

# *Computer vision and deep learning*

Lecture 1

# What is Computer Vision (CV)?

**Goal:** build computer-based vision systems which perform the same functions as the human visual system

**CV** builds the theoretical and algorithmic basis by which useful information about the world can be extracted and analysed from an image, a set of images (e.g. stereo vision) or an image sequence



# Computer vision – multidisciplinary subject

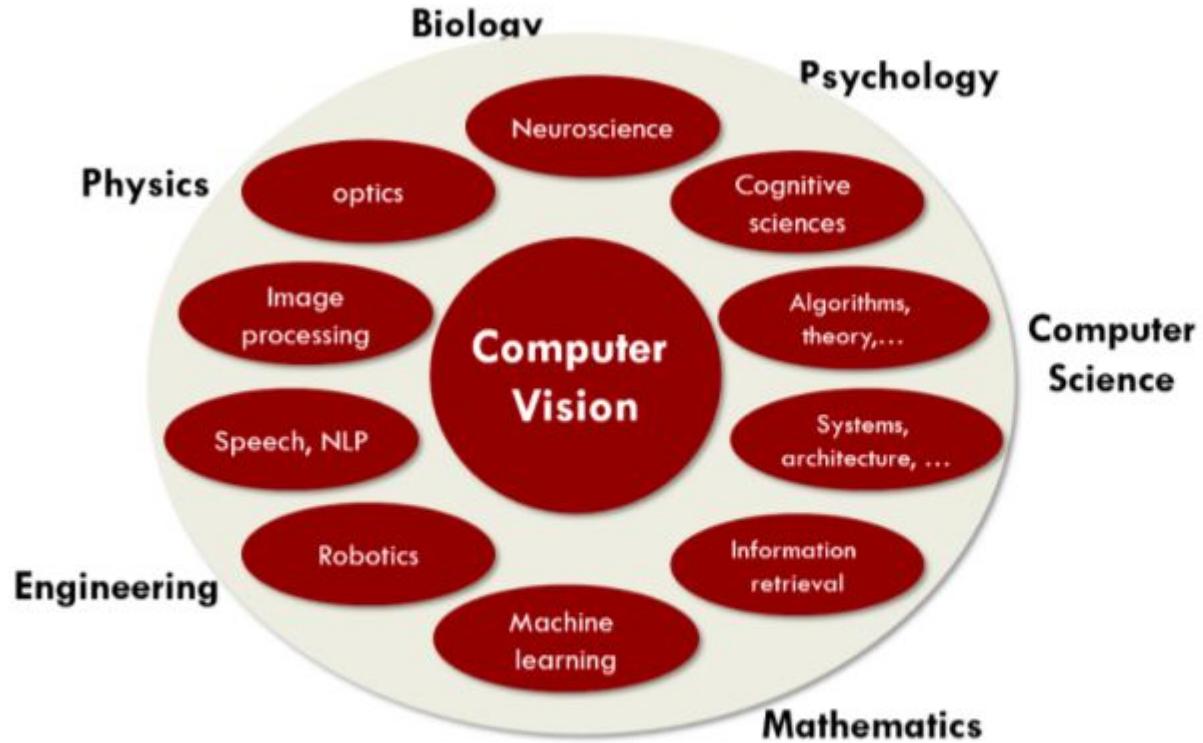
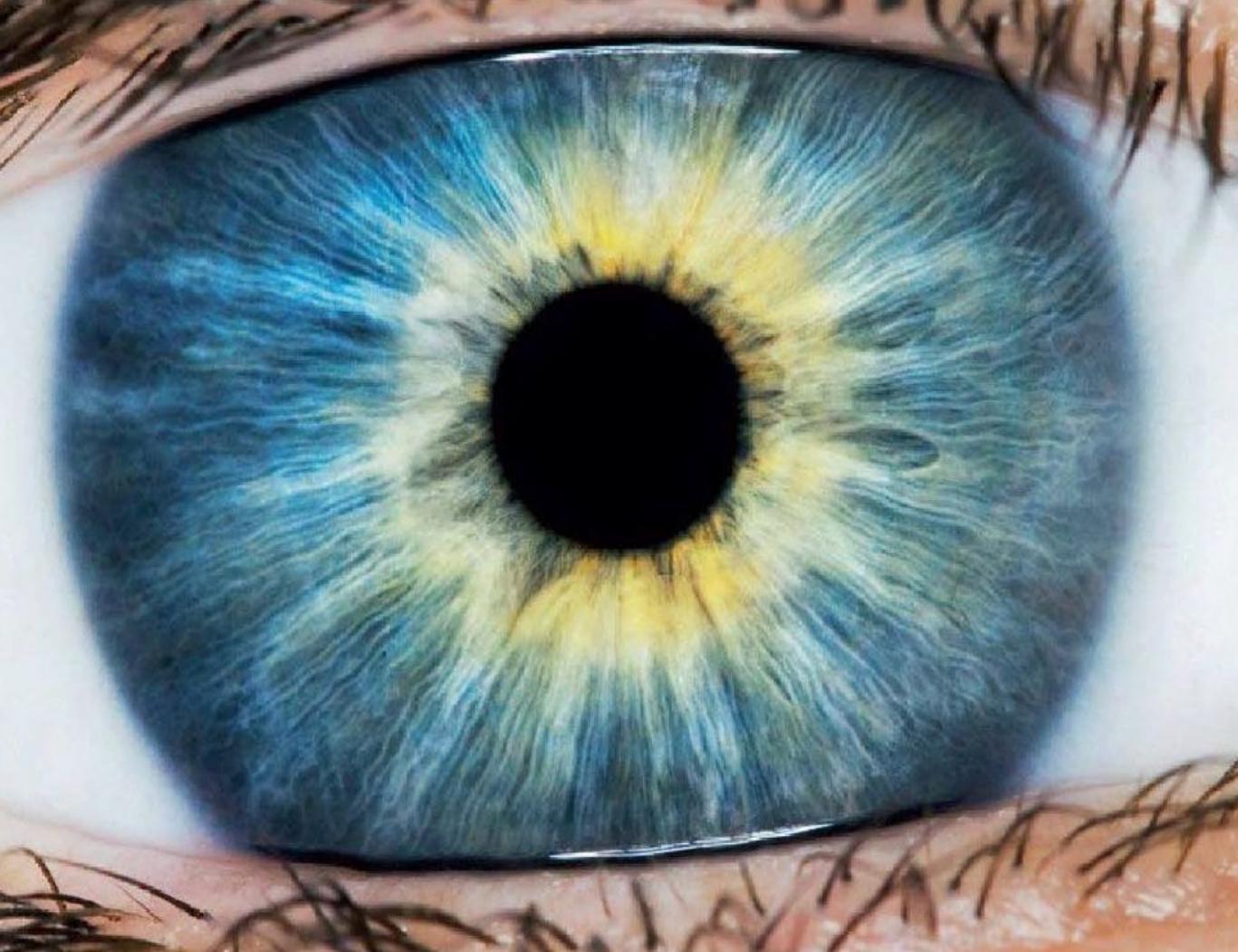
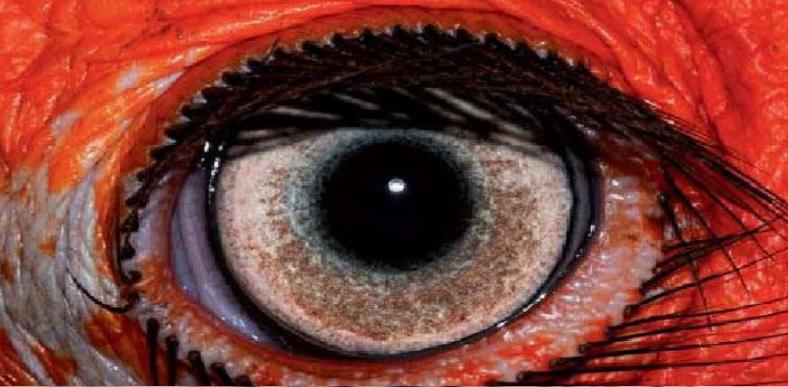
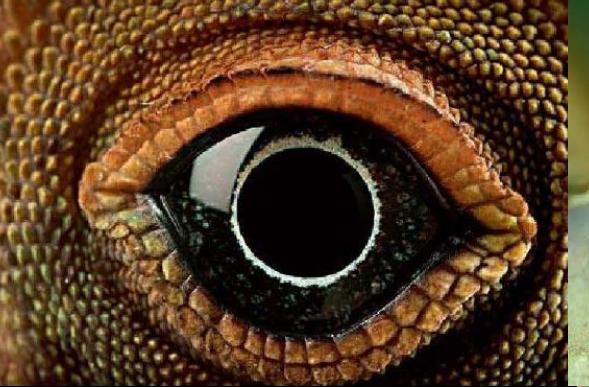
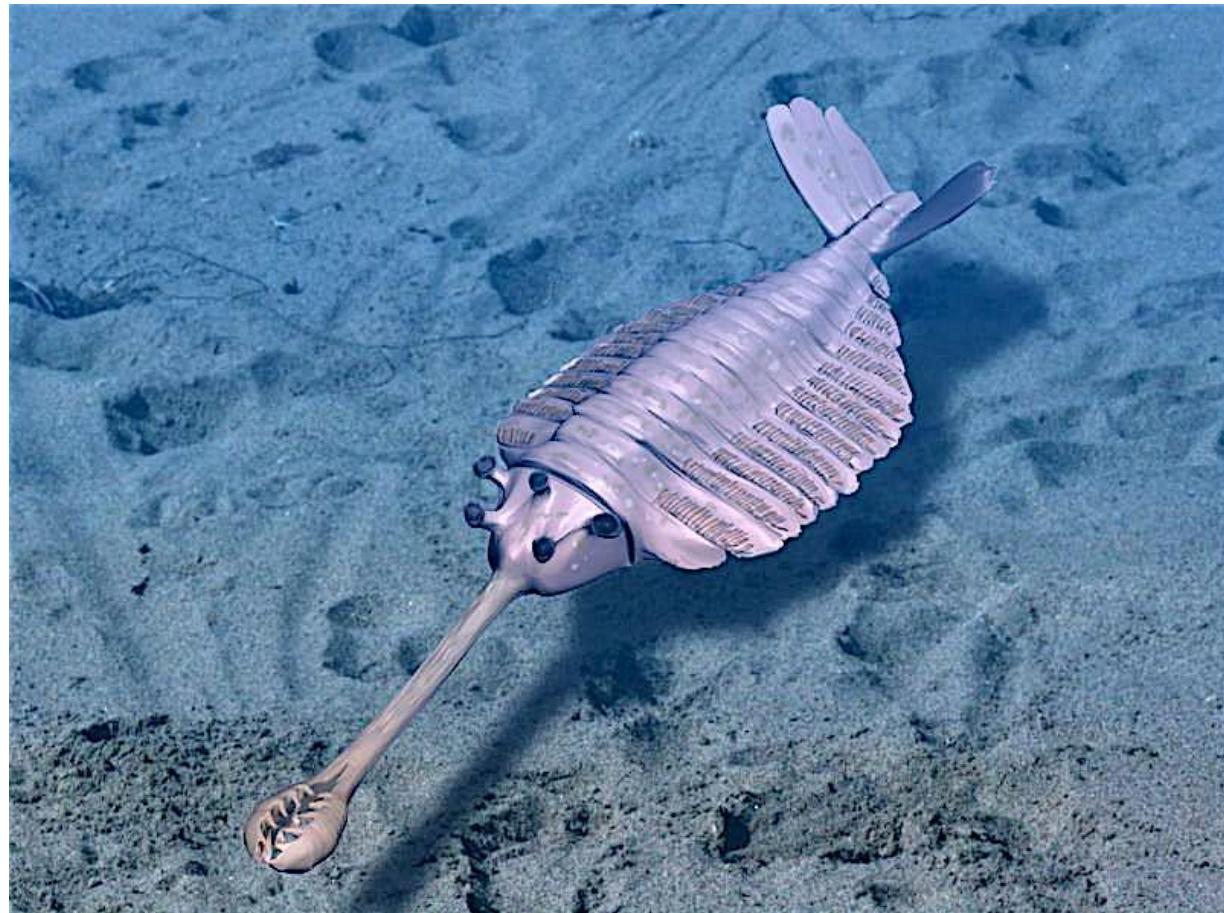


Figure 1: Computer vision at the intersection of multiple scientific fields

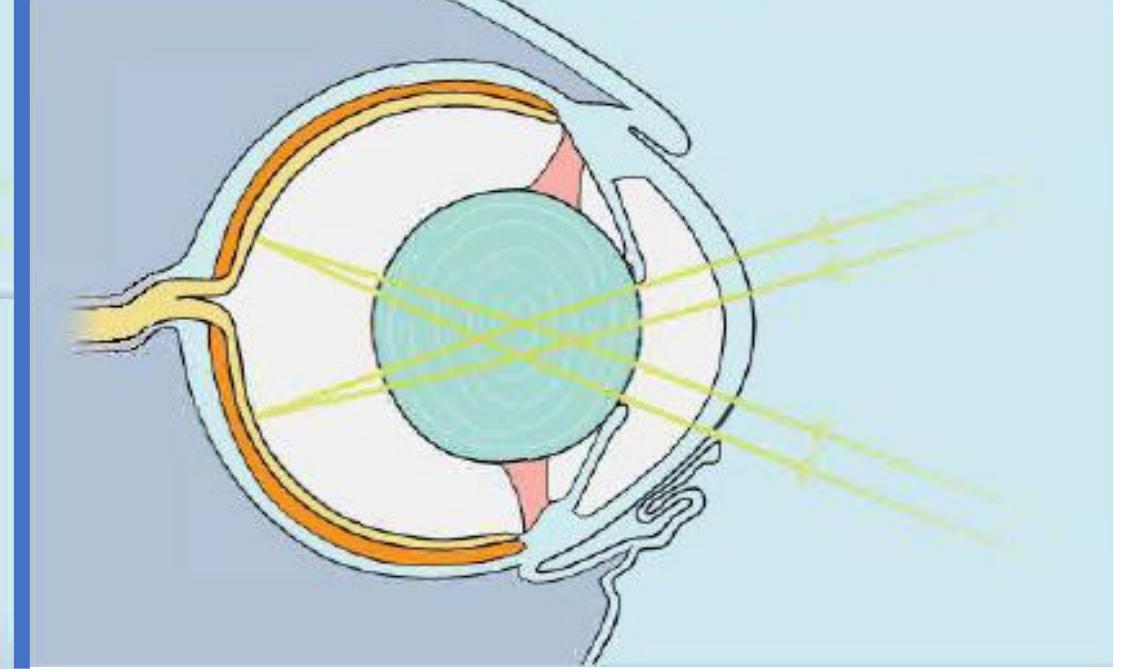
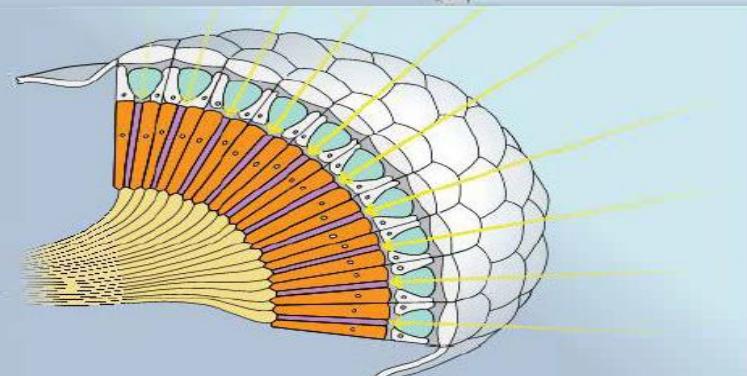
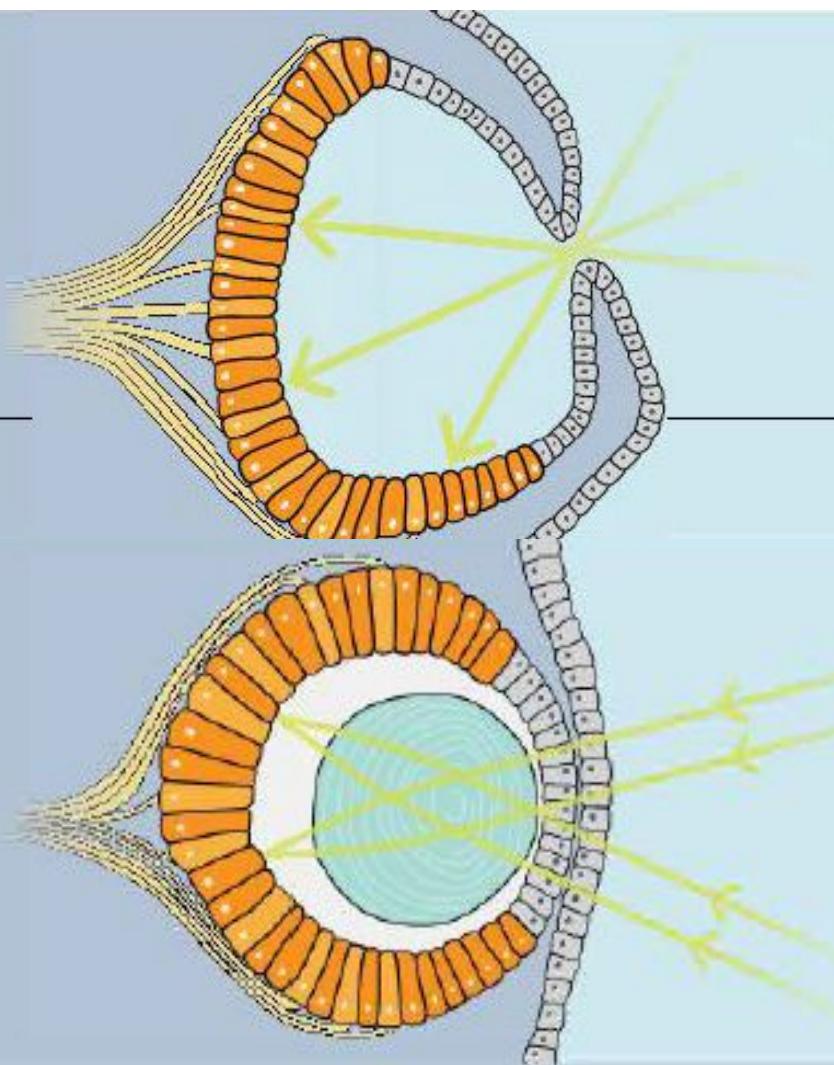
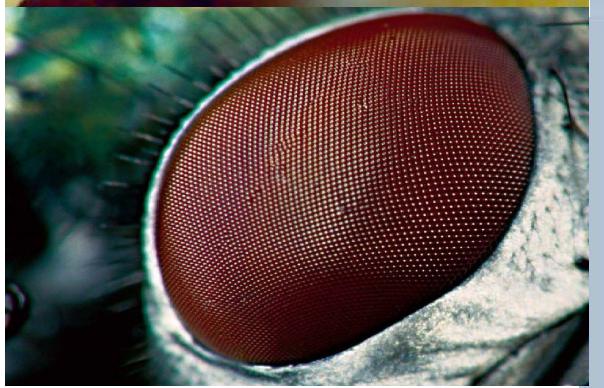


# Evolution of the eye



Cambrian Explosion: 541 million years ago



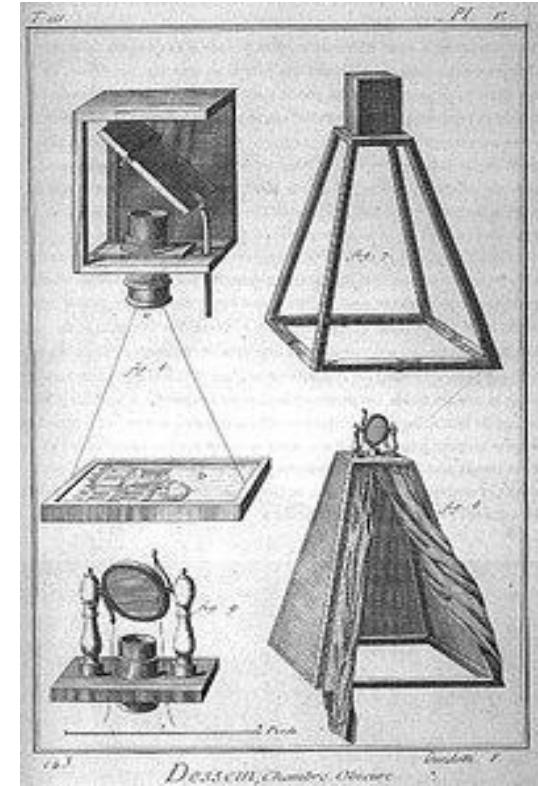


# History of the camera

- Chinese philosopher *Mozi* (470 – 391 BC) first person in history to describe the first to describe the physical principle behind the camera
- Alhazen (Ibn al-Haytham) (965–1040 AD) – considered the inventor of the pinhole camera
- First permanent photograph (1825) by Joseph Nicéphore Niépce (8h exposure time)



*View from the Window at Le Gras*, 1827  
Oldest surviving camera photograph



18th century description of a pinhole camera  
Encyclopedia

# Evolution of the camera

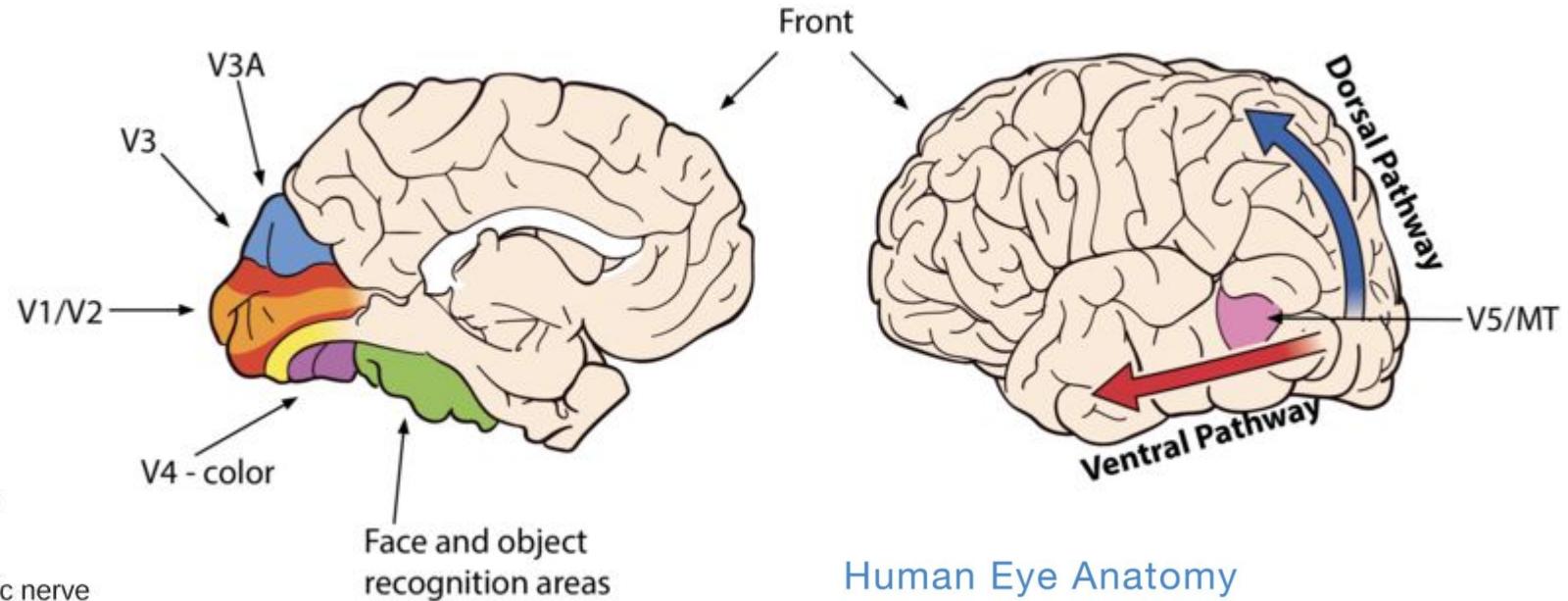
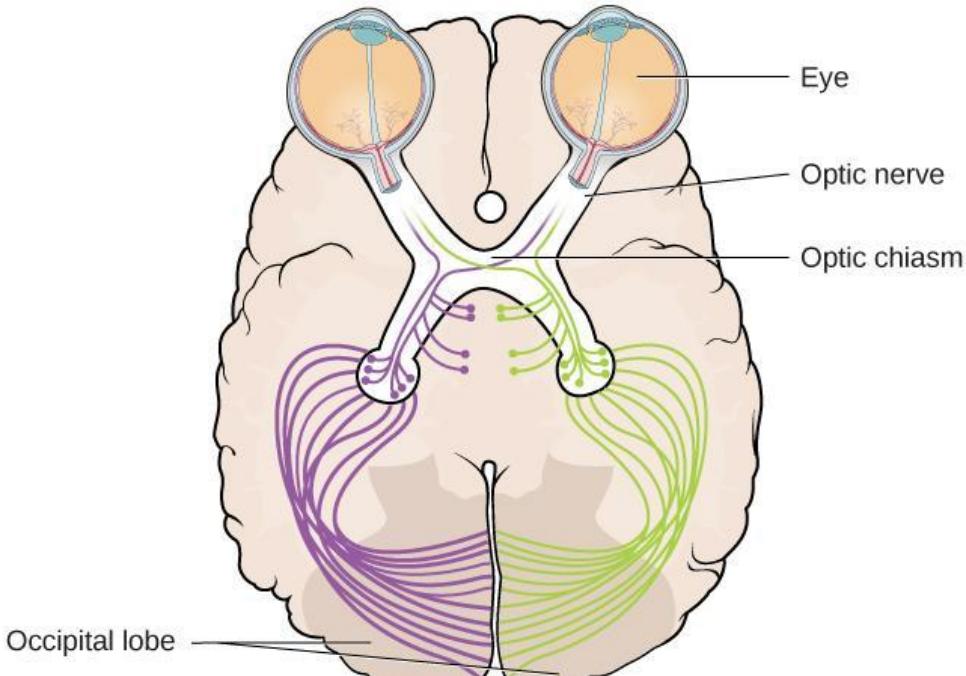


# Nowadays...

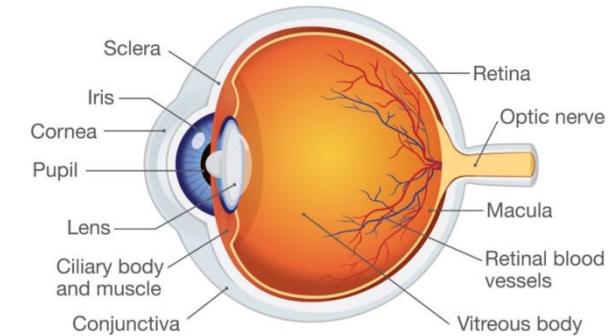
- ~ 1.436.300.000.000 ( > 1.4 trillion) photos will be taken in 2020  
(<https://focus.mylio.com/tech-today/how-many-photos-will-be-taken-in-2020>)
  - If you were to take all these pictures alone, and you took one pic per second = 45.544 years
- As of May 2019, > 500 hours of video are uploaded to YouTube every minute!  
(<https://www.statista.com/statistics/259477/hours-of-video-uploaded-to-youtube-every-minute/>)
- an estimate of 770 million surveillance cameras installed around the world today

# Visual cortex

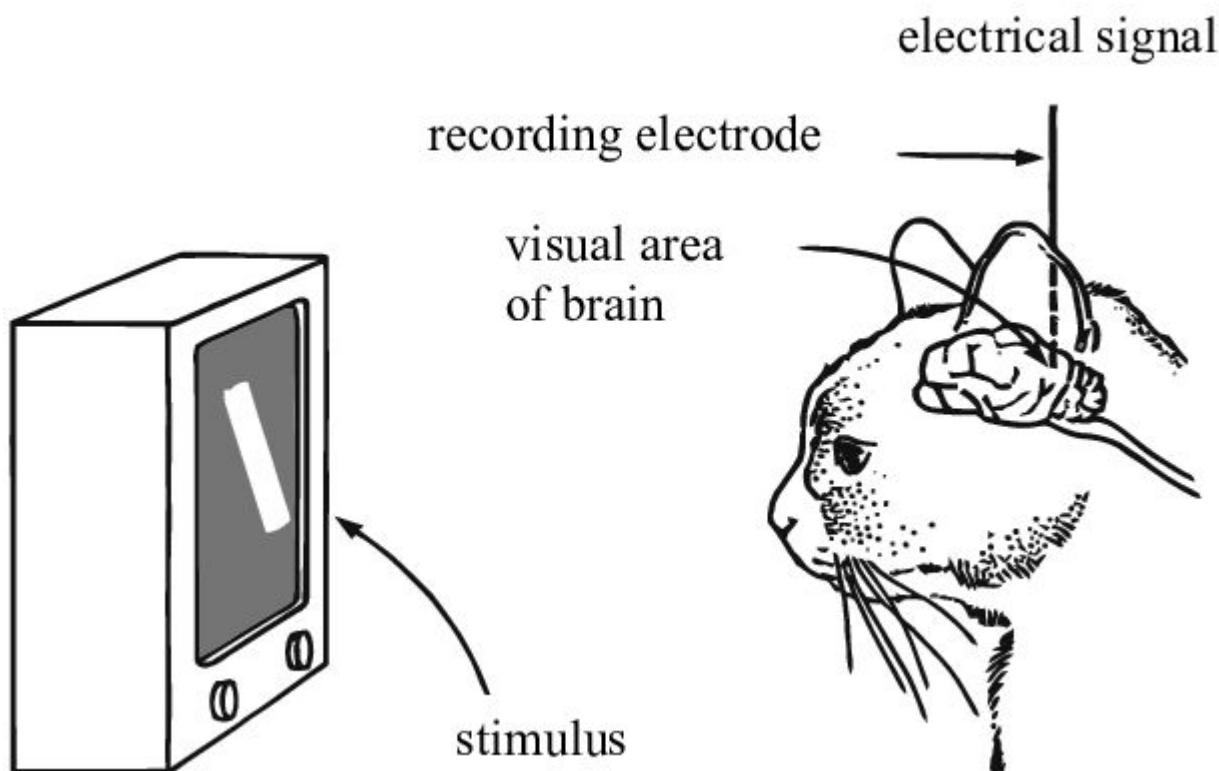
Ventral pathway: “what” pathway  
Dorsal pathway: “where/how” pathway



Human Eye Anatomy



# Understanding the visual cortex



Hubel and Wiesel, 1959

[https://www.youtube.com/watch?v=IOHayh06LJ4&ab\\_channel=PaulLester](https://www.youtube.com/watch?v=IOHayh06LJ4&ab_channel=PaulLester)



Nobel Prize for Physiology or Medicine in 1981:

David Hubel and Torsten Wiesel

**Simple cells:**

orientation, position

**Complex cells:**

orientation, motion, direction

**“Hypercomplex” cells:**

orientation, motion, direction, length

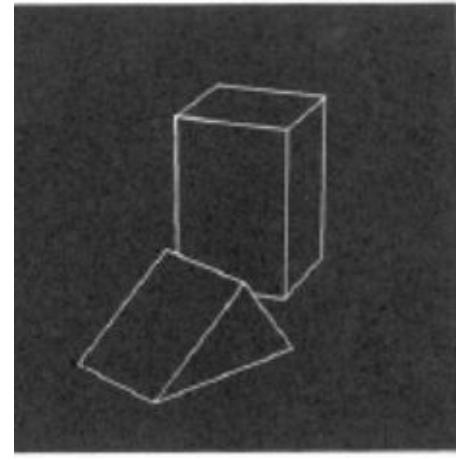
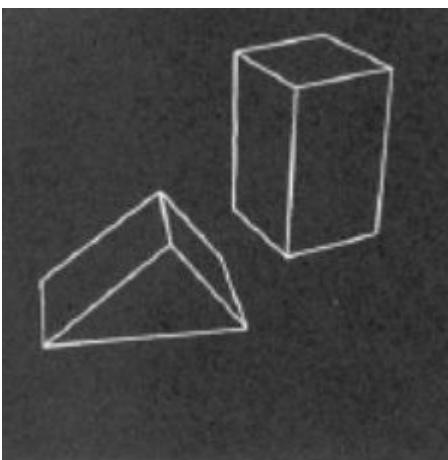
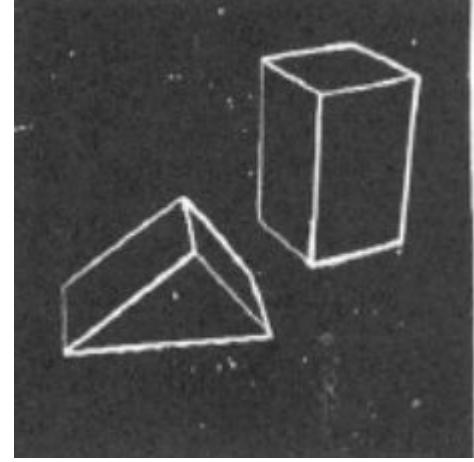
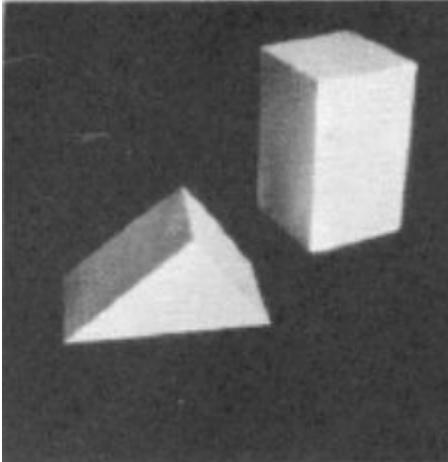
1963

# Larry Roberts



First PhD thesis in the field of Computer Vision: “Machine Perception of Three Dimensional Solids”, Larry Roberts, 1963, MIT Press

- extract 3D information about solid objects from 2D photographs of line drawings



1966

# The Summer Vision Project

MASSACHUSETTS INSTITUTE OF TECHNOLOGY  
PROJECT MAC

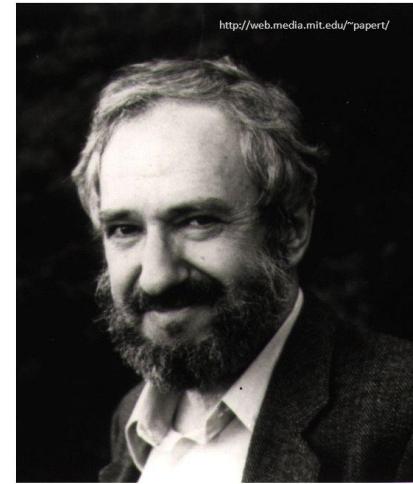
Artificial Intelligence Group  
Vision Memo. No. 100.

July 7, 1966

## THE SUMMER VISION PROJECT

Seymour Papert

The summer vision project is an attempt to use our summer workers effectively in the construction of a significant part of a visual system. The particular task was chosen partly because it can be segmented into sub-problems which will allow individuals to work independently and yet participate in the construction of a system complex enough to be a real landmark in the development of "pattern recognition".



<http://web.media.mit.edu/~paper/>

*"I am convinced that the best learning takes place when the learner takes charge."*

— Seymour Papert

© Steve Wheeler, University of Plymouth, 2015

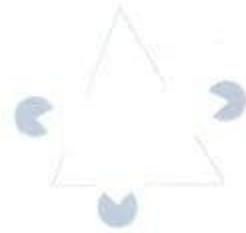


*The primary goal of the project is to construct a system of programs which will divide a vidisector picture into regions such as likely objects, likely background areas and chaos. We shall call this part of its operation FIGURE-GROUND analysis. It will be impossible to do this without considerable analysis of shape and surface properties, so FIGURE-GROUND analysis is really inseparable in practice from the second goal which is REGION DESCRIPTION. The final goal is OBJECT IDENTIFICATION which will actually name objects by matching them with a vocabulary of known objects.*

1981

# David Marr

# VISION



DAVID MARR

Scene

Image

Primal sketch

2.5 Surface Map  
2.5D Symbolic map

Relational  
Structure

Recognition description

Pixel intensities

Edges, blobs, curves  
boundaries

Depth image, surface  
orientations

Object centred  
coordinate system  
(surface and  
geometrical  
primitives)

1986

# Canny edge detector



# Normalized Cuts and Image Segmentation

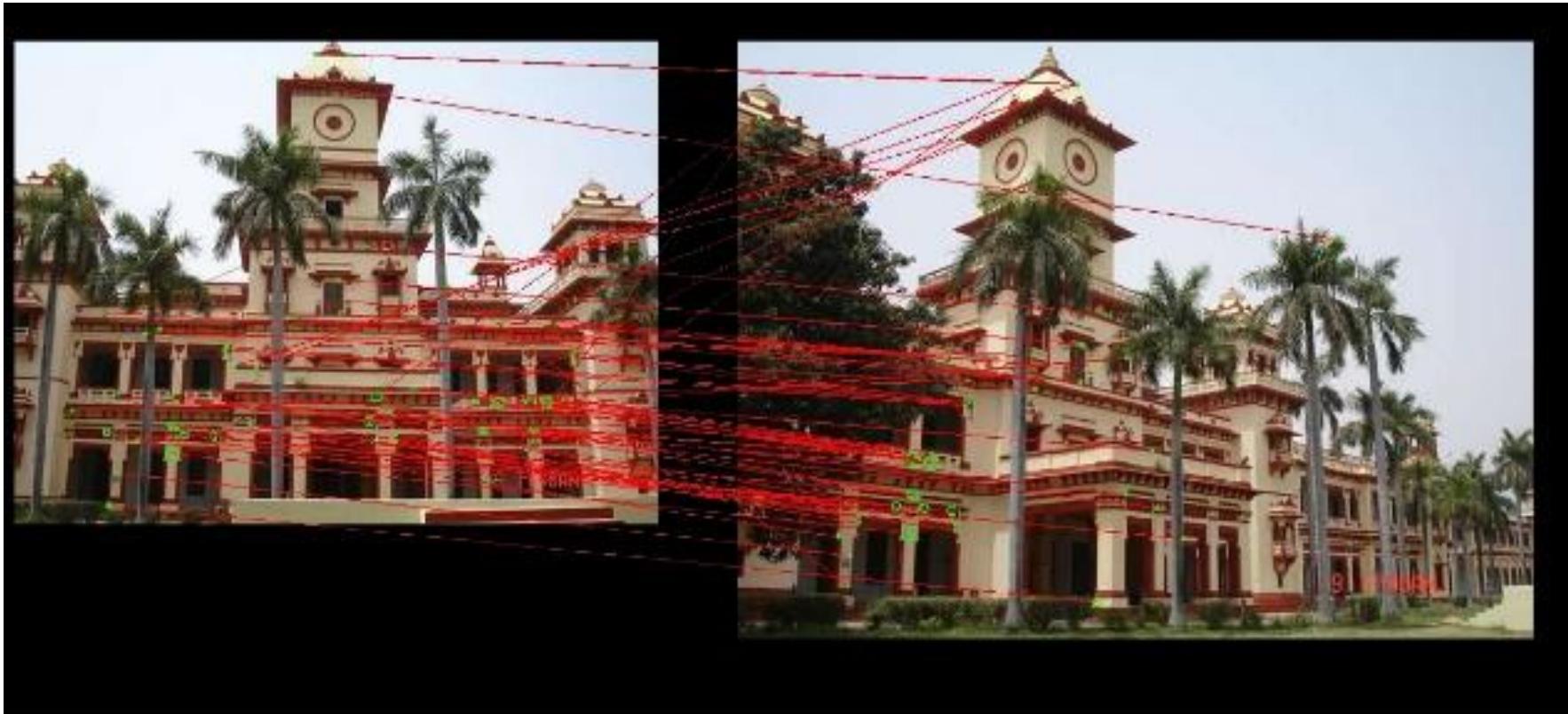
Jianbo Shi and Jitendra Malik



# Scale Invariant Feature Transform

David G. Lowe

Local image features invariant to image scaling, translation, and rotation, and partially invariant to illumination changes and affine or 3D projection

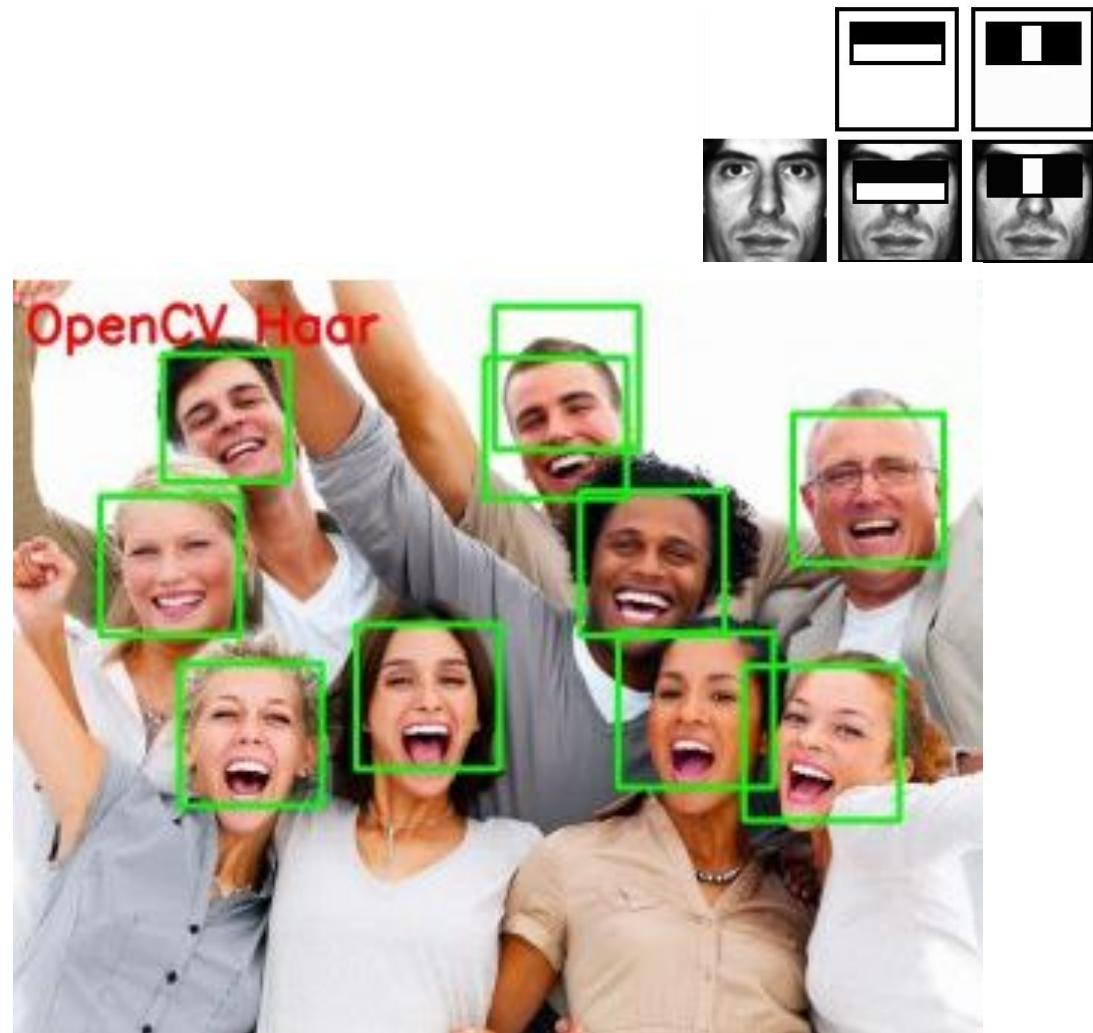


# Rapid object detection using a boosted cascade of simple features

Paul Viola and Michael Jones

## Abstract

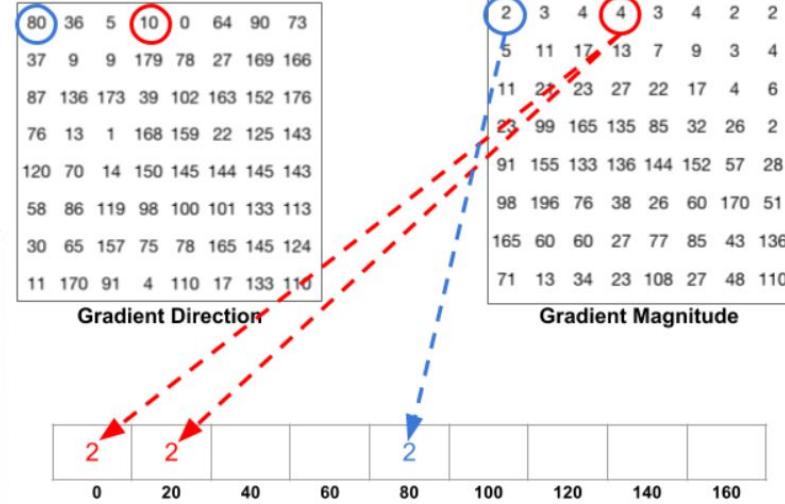
*This paper describes a machine learning approach for visual object detection which is capable of processing images extremely rapidly and achieving high detection rates. This work is distinguished by three key contributions. The first is the introduction of a new image representation called the “Integral Image” which allows the features used by our detector to be computed very quickly. The second is a learning algorithm, based on AdaBoost, which selects a small number of critical visual features from a larger set and yields extremely efficient classifiers[6]. The third contribution is a method for combining increasingly more complex classifiers in a “cascade” which allows background regions of the image to be quickly discarded while spending more computation on promising object-like regions. The cascade can be viewed as an object specific focus-of-attention mechanism which unlike previous approaches provides statistical guarantees that discarded regions are unlikely to contain the object of interest. In the domain of face detection the system yields detection rates comparable to the best previous systems. Used in real-time applications, the detector runs at 15 frames per second without resorting to image differencing or skin color detection.*



# Histogram of Oriented Gradients

2005

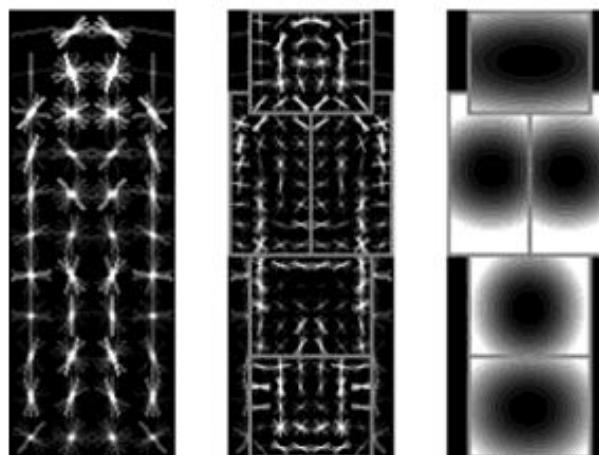
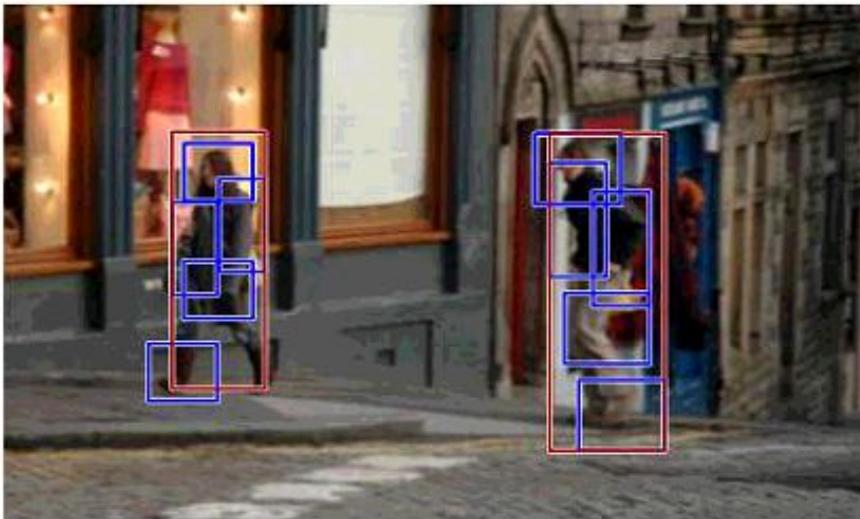
Navneet Dalal and Bill Triggs



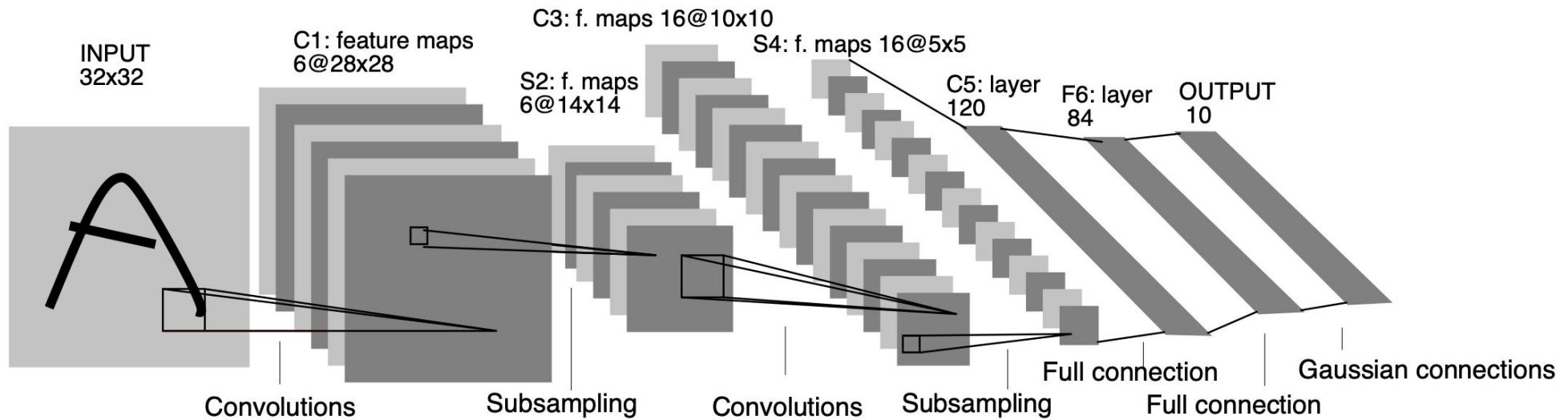
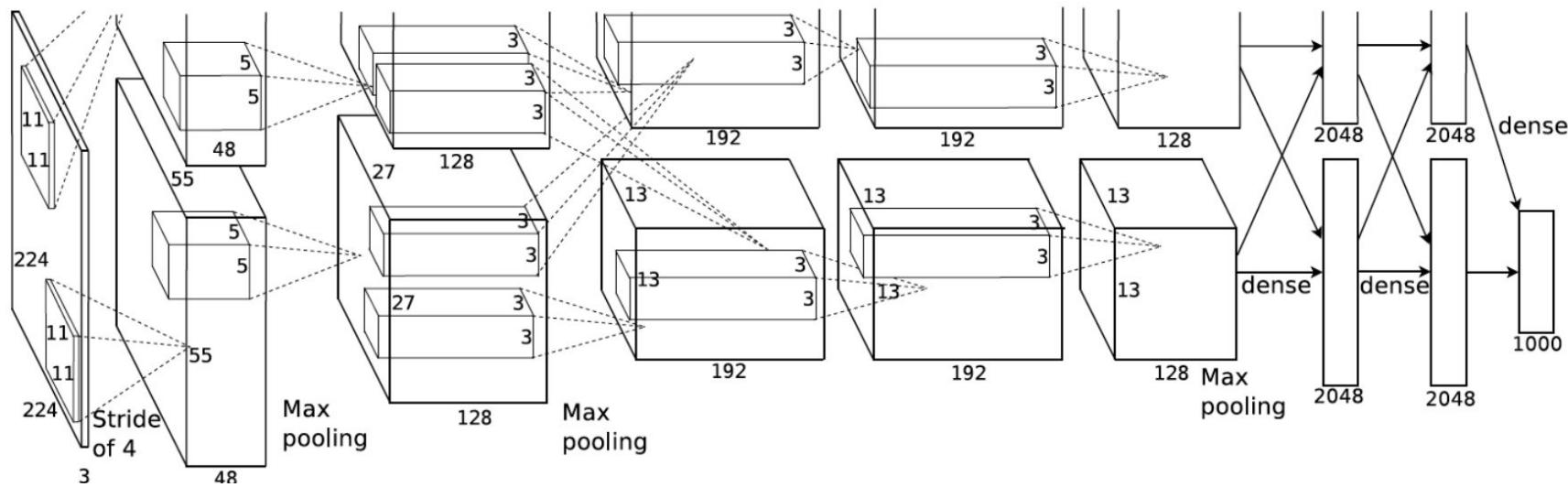
# Deformable part models

2009

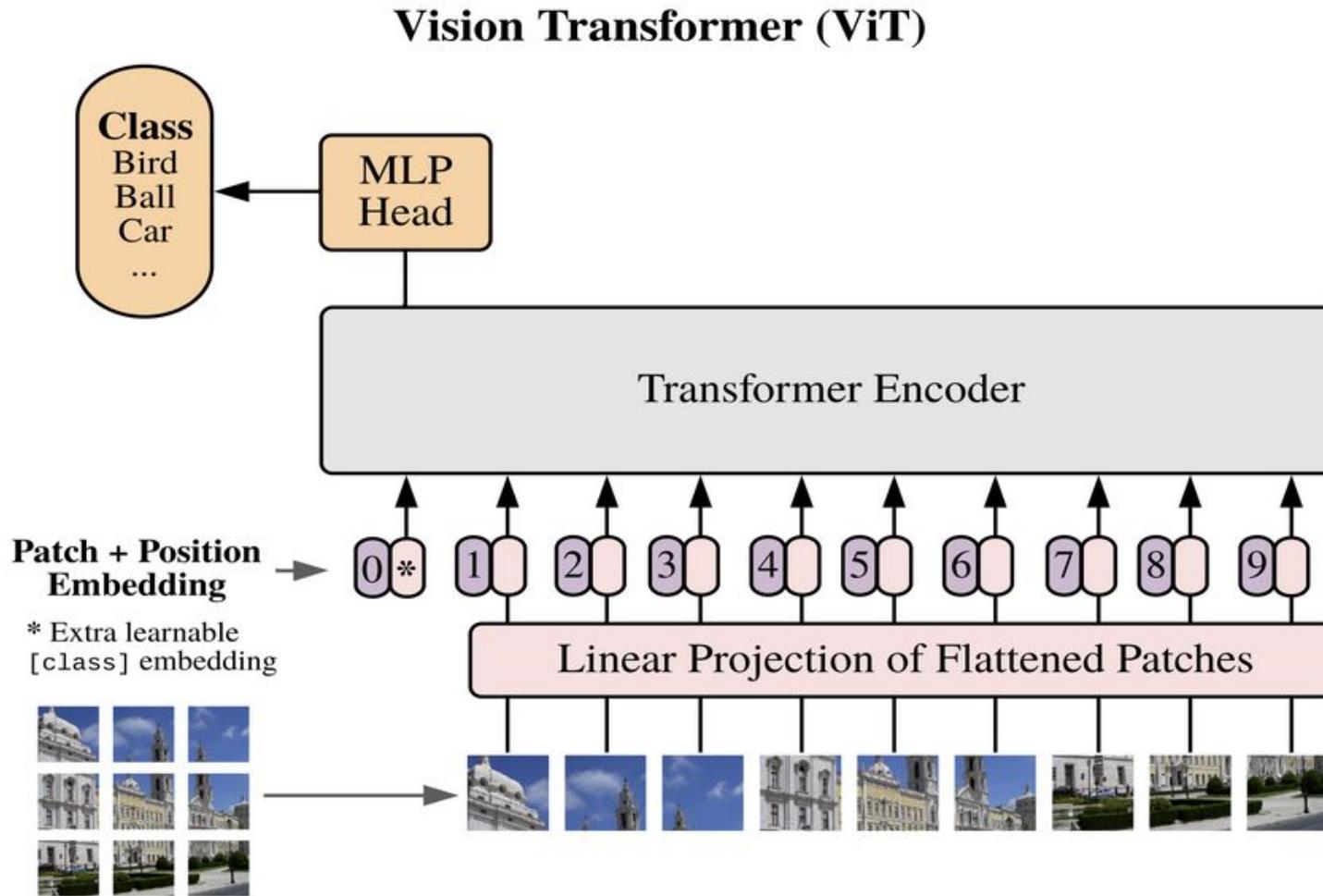
P. Felzenszwalb, R. Girshick, D. McAllester, D. Ramanan



2012

**LeNet (1998)****AlexNet (2012)**

# Visual transformers



# Data



# Caltech 101

- 9,146 images, split between 101 distinct object categories and a background category
- 40 to 800 images per category



# Pascal VOC

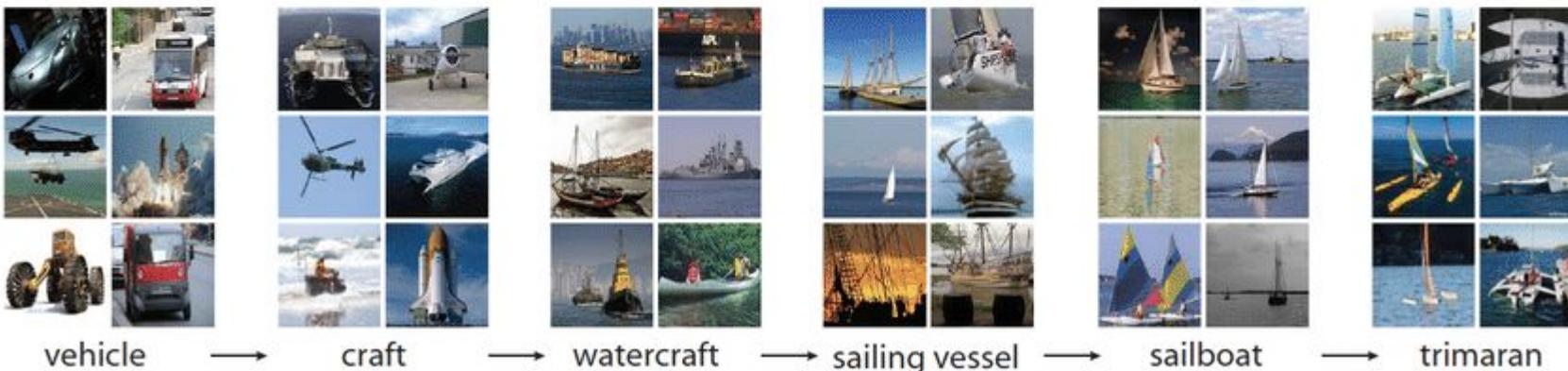
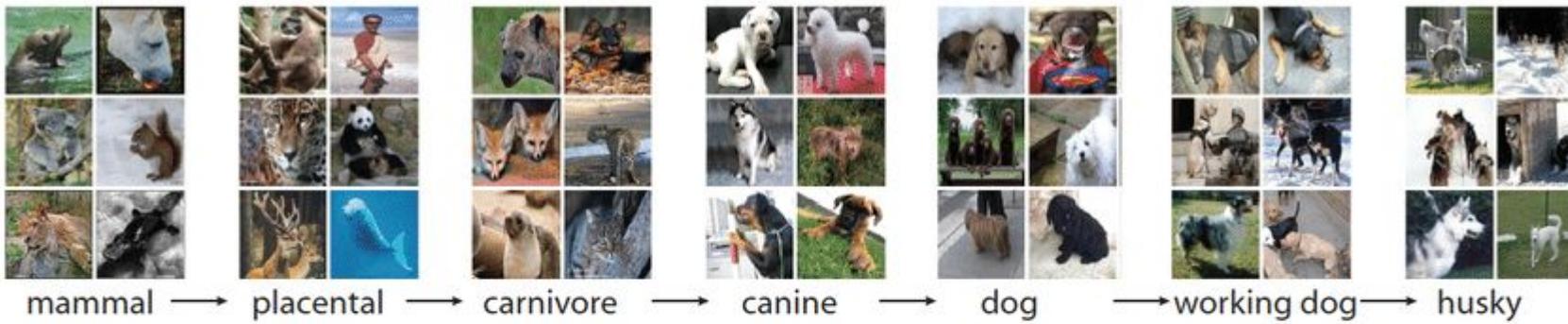
- 9,963 images containing 24,640 object split into 20 classes:
  - airplane, bicycle, boat, bottle, bus, car, cat, chair, cow, dining table, dog, horse, motorbike, person, potted plant, sheep, train, TV



2010

# ImageNet

- > 14 million images organized into over 20,000 categories
- organized according to the WordNet hierarchy (nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images



# JFT

- internal dataset used by Google to train image classification algorithms
- the dataset has 300M images and 375M labels (multiple labels per image, on average each image has 1.26 labels)
- *noisy labels*: no human processing is involved
- labeled using an algorithm that uses complex mixture of raw web signals, connections between webpages and user feedback

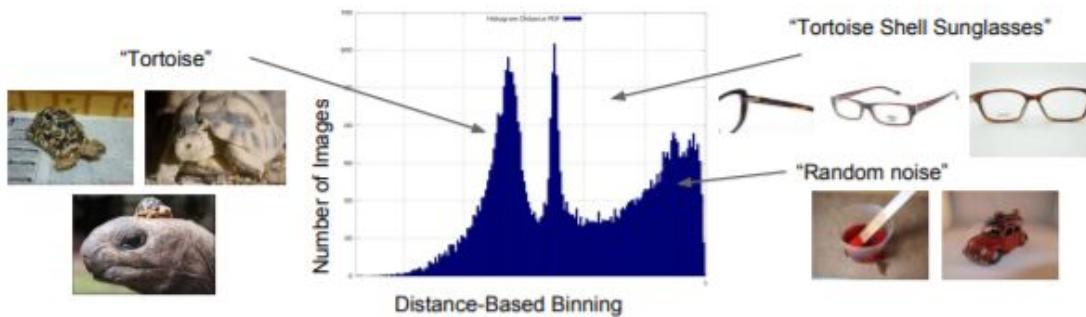


Figure 2. JFT-300M dataset can be noisy in terms of label confusion and incorrect labels. This is because labels are generated via a complex mixture of web signals, and not annotated or cleaned by humans. x-axis corresponds to the quantized distances to K-Means centroids, which are computed based on visual features.

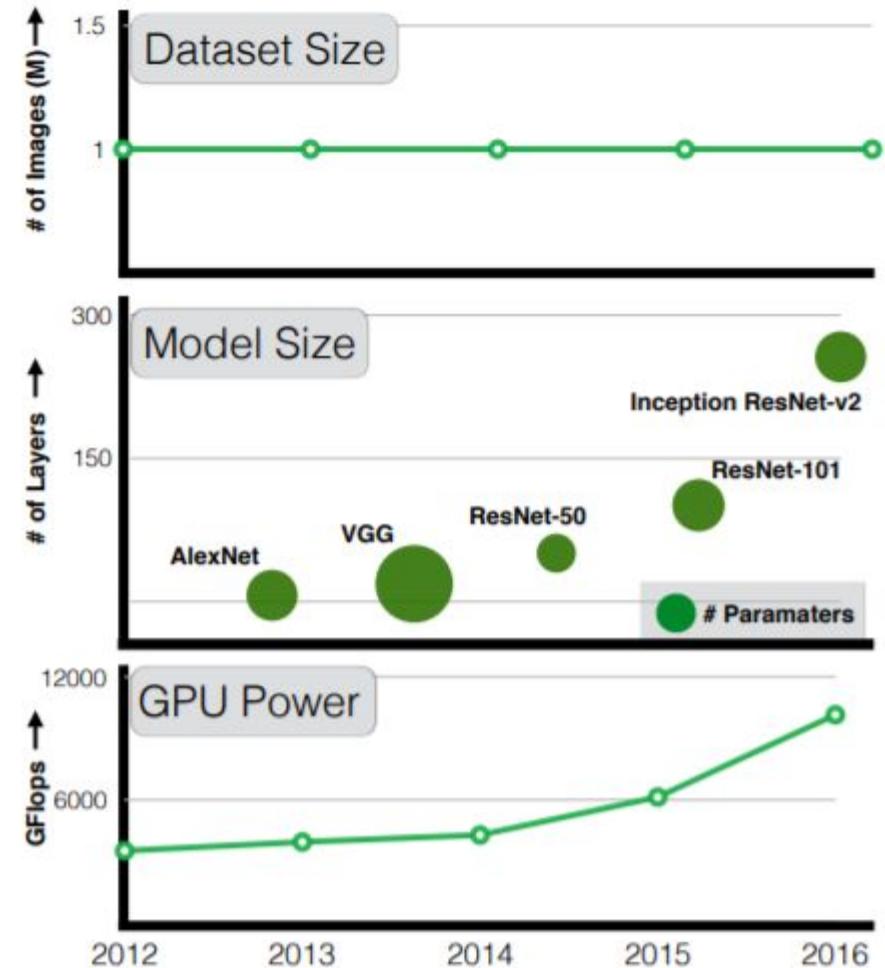
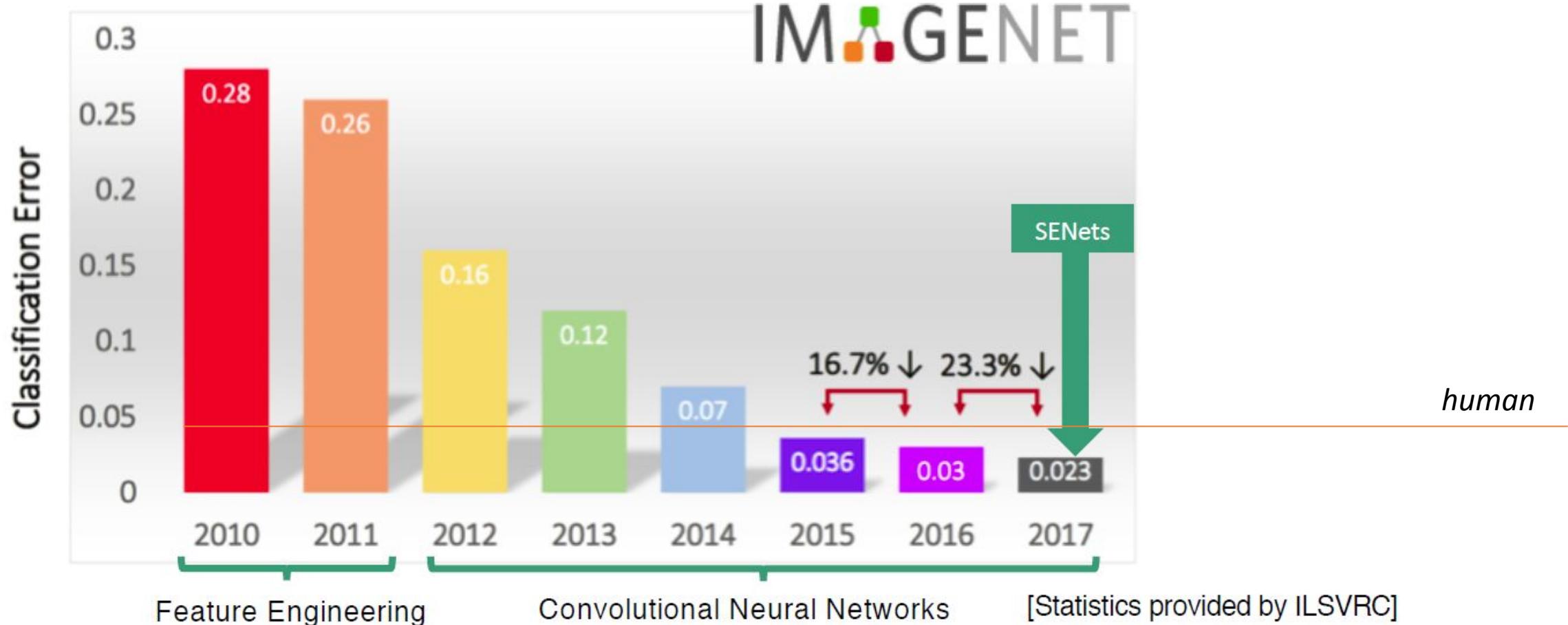
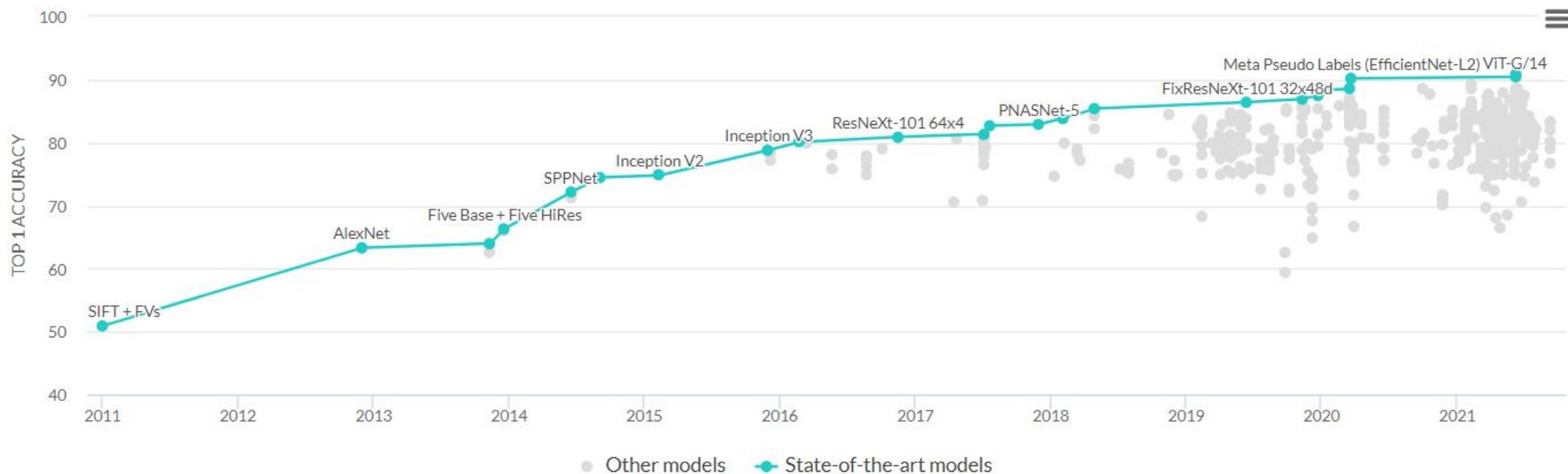


Figure 1. The Curious Case of Vision Datasets: While GPU computation power and model sizes have continued to increase over the last five years, size of the largest training dataset has surprisingly remained constant. Why is that? What would have happened if we have used our resources to increase dataset size as well? This paper provides a sneak-peek into what could be if the dataset sizes are increased dramatically.

# Computer vision - Where are we now?



# ImageNet accuracy



# Computer vision applications I



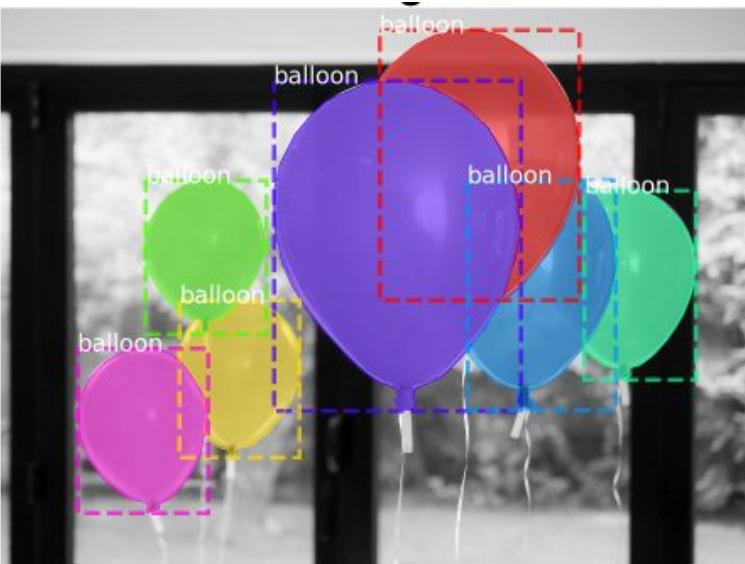
Penguin  
Classification



Object  
Detection



Semantic segmentation

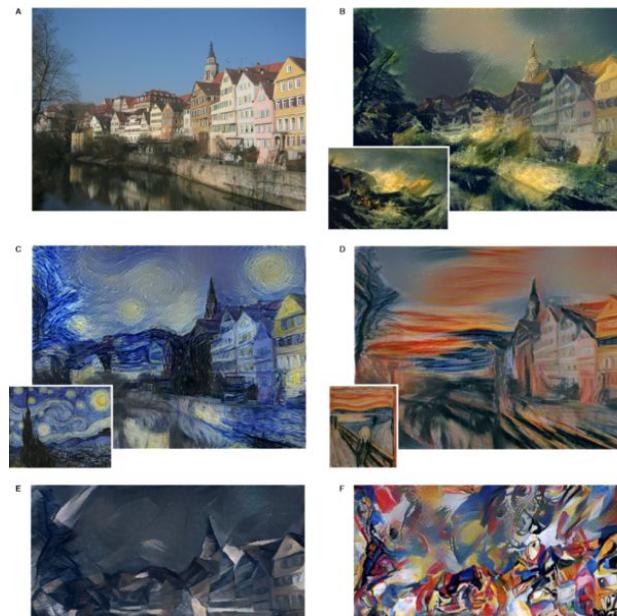


Instance segmentation

# Computer vision applications II



**Image captioning**



**Style transfer**



**Action recognition**

# Computer vision applications III



**Image  
generation**

<https://thispersondoesnotexist.com/>



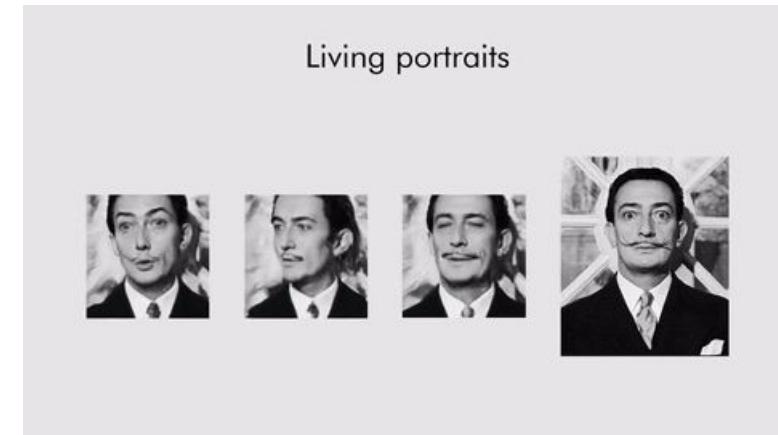
<https://thiscatdoesnotexist.com/>

<https://edition.cnn.com/2020/02/28/tech/fake-twitter-candidate-2020/index.html>

# Computer vision applications IV



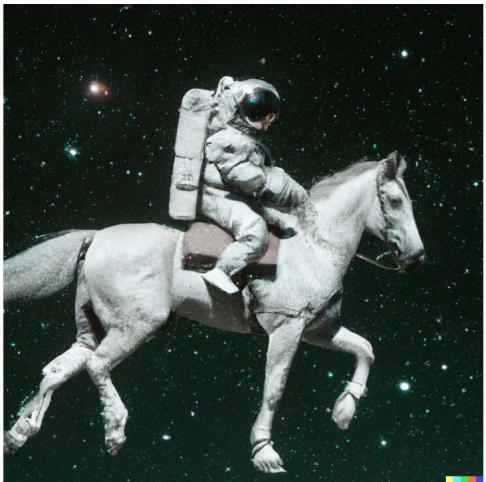
Image translation



Living portraits

Talking head models

# Computer vision applications V



An astronaut riding a horse in a photorealistic style



A painting of a fox sitting in a field at sunrise in the style of Claude Monet

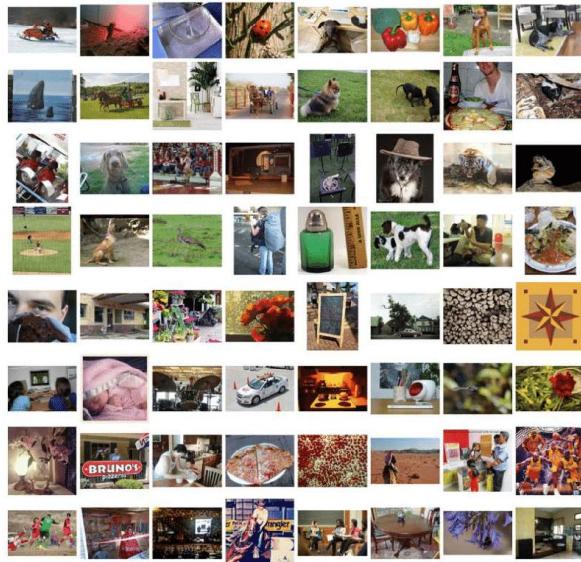


Teddy bears working on new AI research on the moon in the 1980s

# Computer vision surpassed human vision



# Captcha recognition



# Object recognition



# Face recognition

*“We should all be playing games  
and let AI do all the work, instead  
AI plays games and we do all the  
work”*



Andrej Karpathy

# Computer vision vs. human vision



Example from Andrej Karpathy's blog: <http://karpathy.github.io/2012/10/22/state-of-computer-vision/>

# Stanford AI index 2021

Industry has become, by far, the largest consumer of AI talent. 65% of graduating North American PhDs in AI went into industry—up from 44.4% in 2010.

AI investment in drug design and discovery increased significantly

Globally, investment in AI start-ups: from a total of \$1.3B raised in 2010 to over \$40.4B in 2018 (with \$37.4B in 2019 as of November 4th), funding has increased at an average annual growth rate of over 48%

PRIVATE INVESTMENT in FUNDED AI COMPANIES, 2015-20  
Source: CapiQ, Crunchbase, and NetBase Quid, 2020 | Chart: 2021 AI Index Report

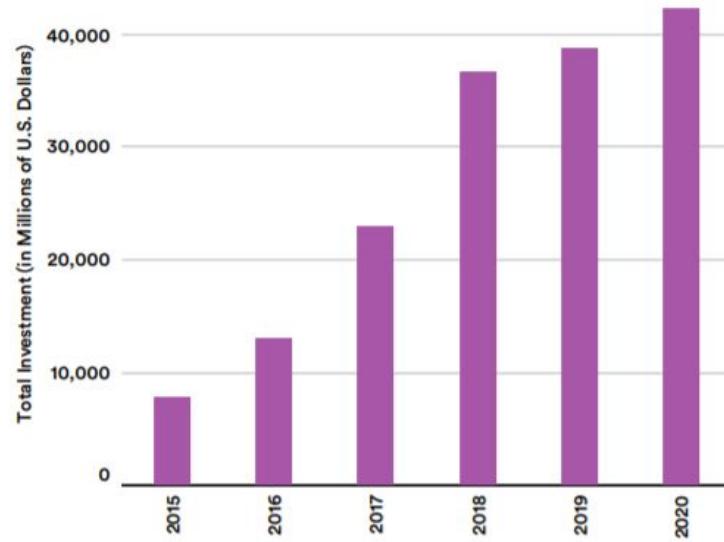
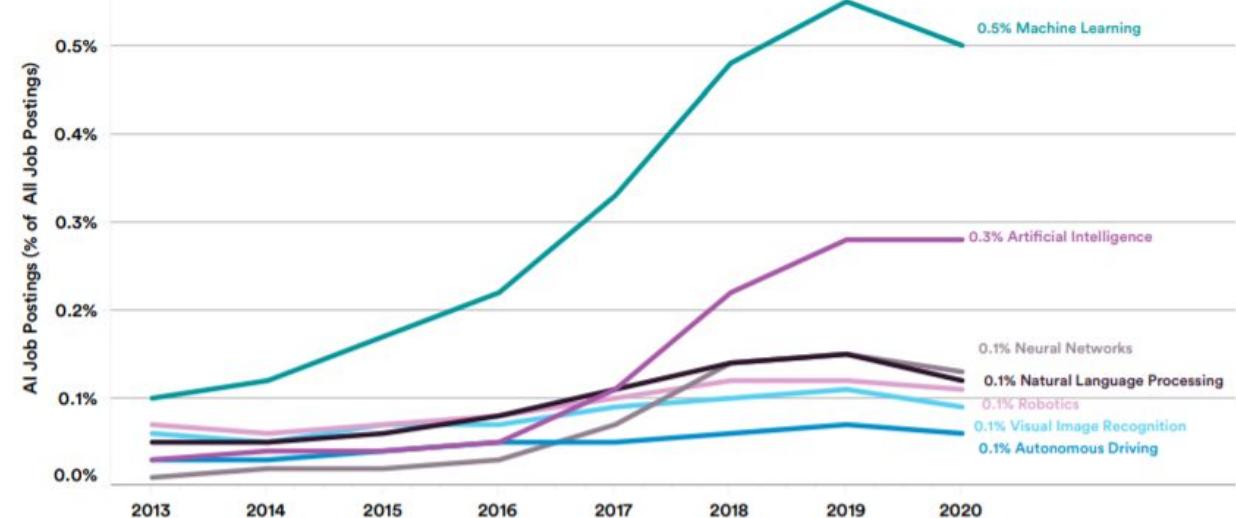


Figure 3.2.2

AI JOB POSTINGS (% of ALL JOB POSTINGS) in the UNITED STATES by SKILL CLUSTER, 2013-20  
Source: Burning Glass, 2020 | Chart: 2021 AI Index Report



# Stanford AI index 2022

## NEW CS PHDS (% of TOTAL) in the UNITED STATES by SPECIALITY, 2020

Source: CRA Taulbee Survey, 2021 | Chart: 2022 AI Index Report

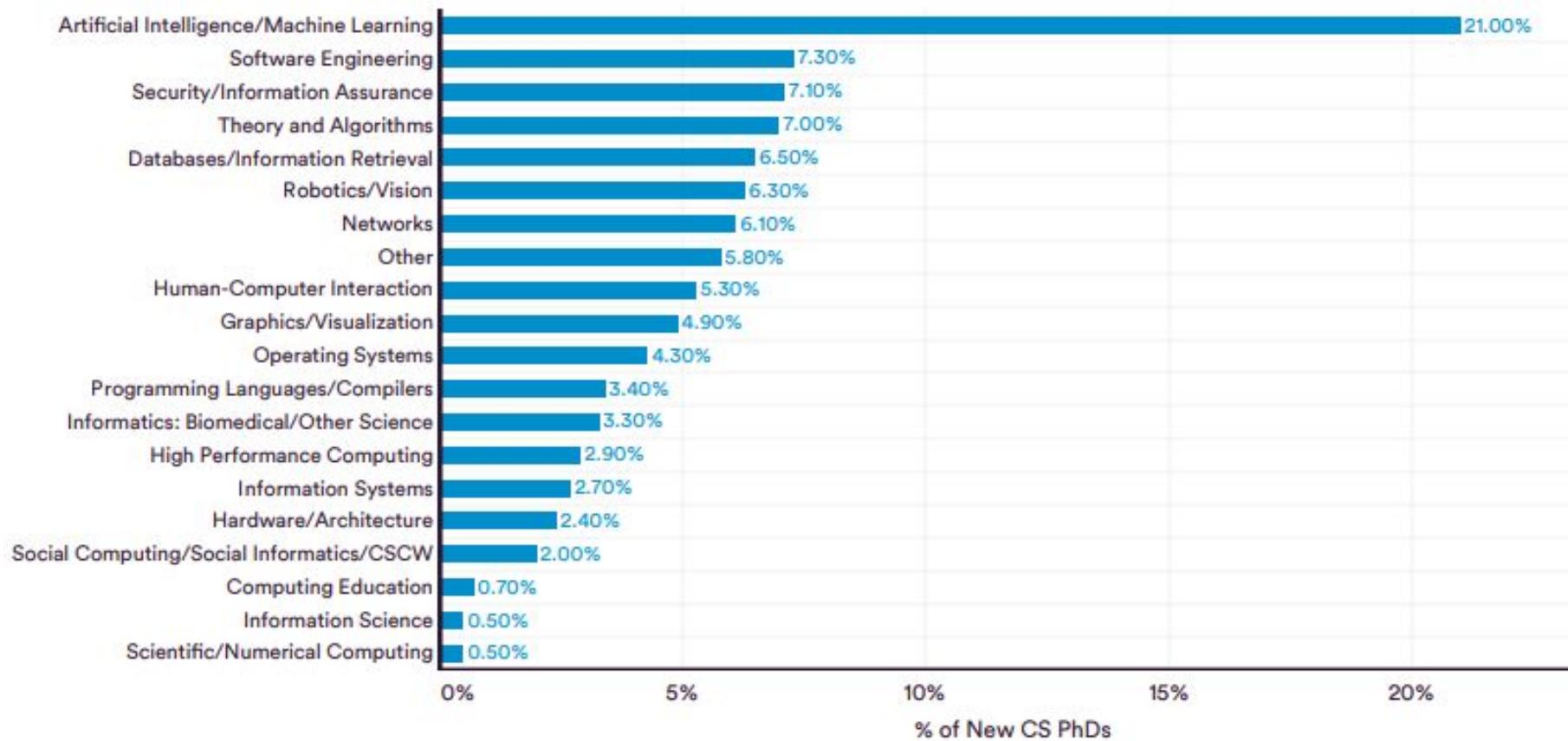


Figure 4.4.2

# Stanford AI index 2022

AI JOB POSTINGS (% of ALL JOB POSTINGS) by GEOGRAPHIC AREA, 2013–21

Source: Emsi Burning Glass, 2021 | Chart: 2022 AI Index Report

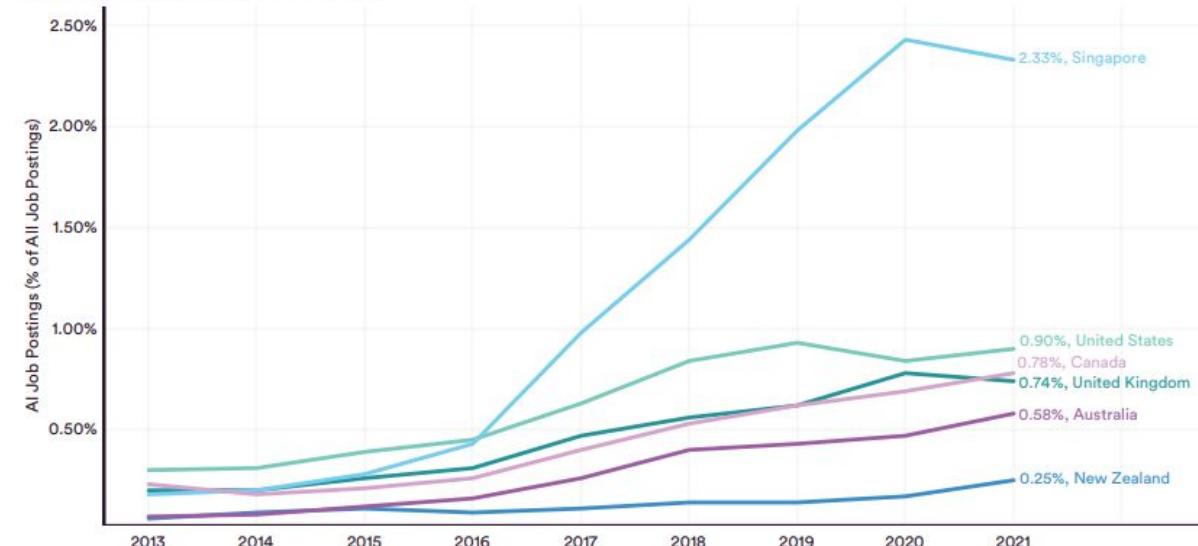


Figure 4.1.3

AI JOB POSTINGS (% of ALL JOB POSTINGS) in the UNITED STATES by SKILL CLUSTER, 2010–21

Source: Emsi Burning Glass, 2021 | Chart: 2022 AI Index Report

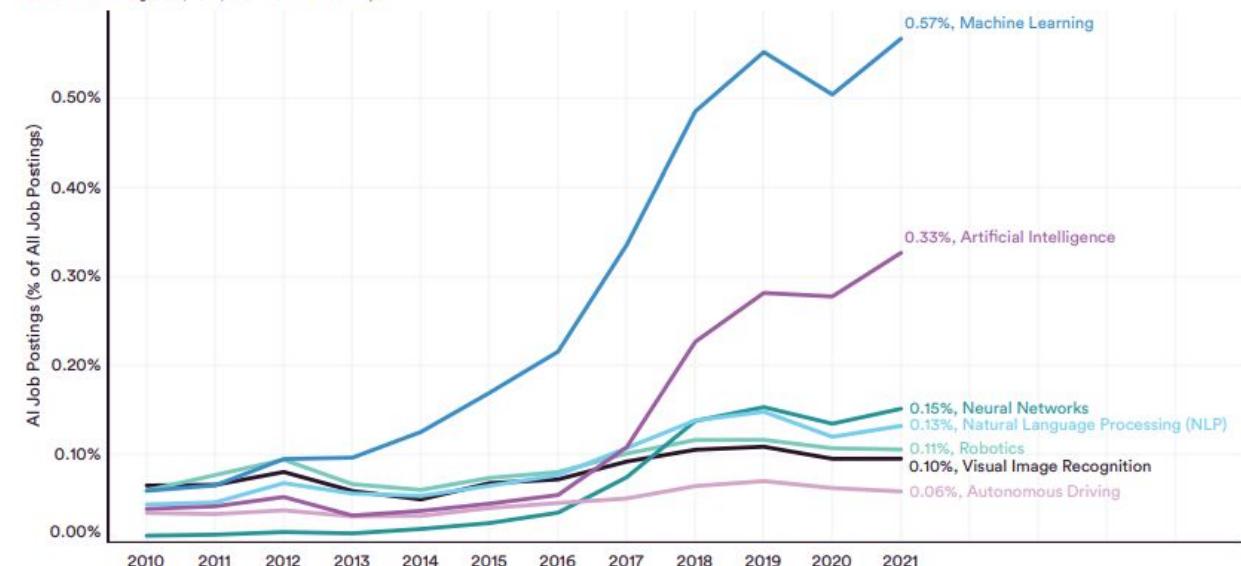


Figure 4.1.4

# Stanford AI index 2022

GLOBAL CORPORATE INVESTMENT in AI by INVESTMENT ACTIVITY, 2013–21

Source: NetBase Quid, 2021 | Chart: 2022 AI Index Report

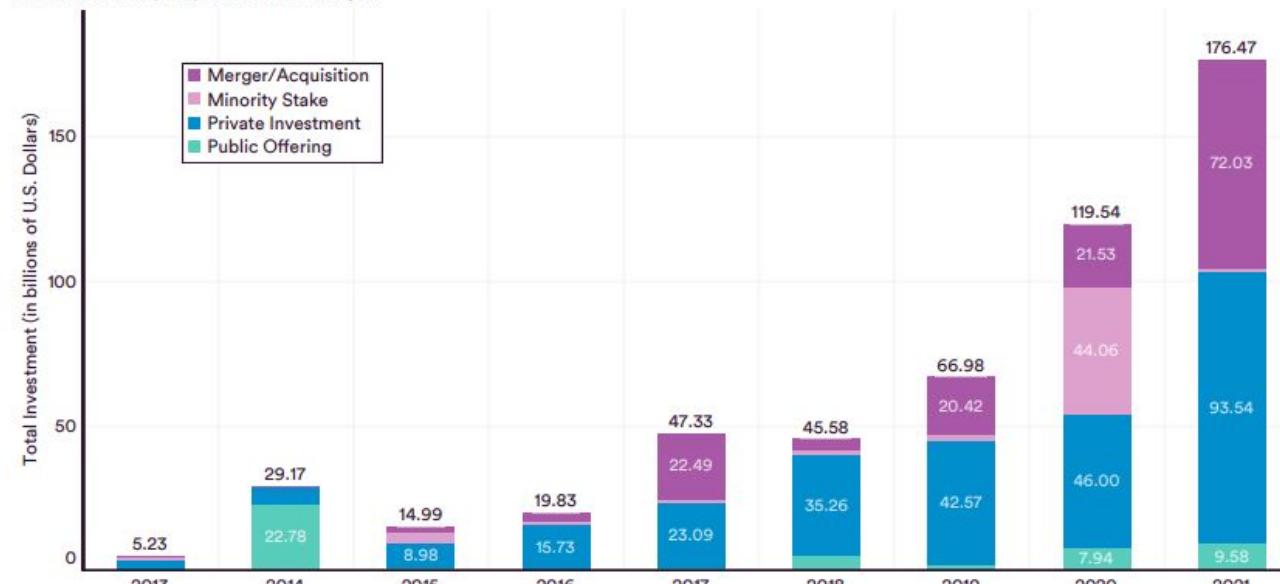


Figure 4.2.1

NUMBER of NEWLY FUNDED AI COMPANIES in the WORLD, 2013–21

Source: NetBase Quid, 2021 | Chart: 2022 AI Index Report

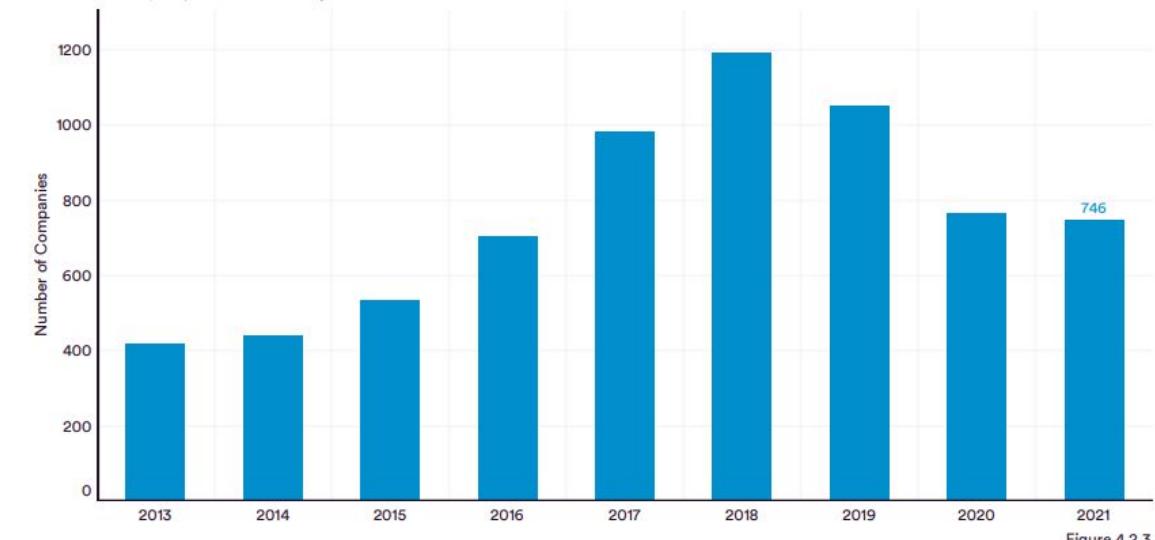
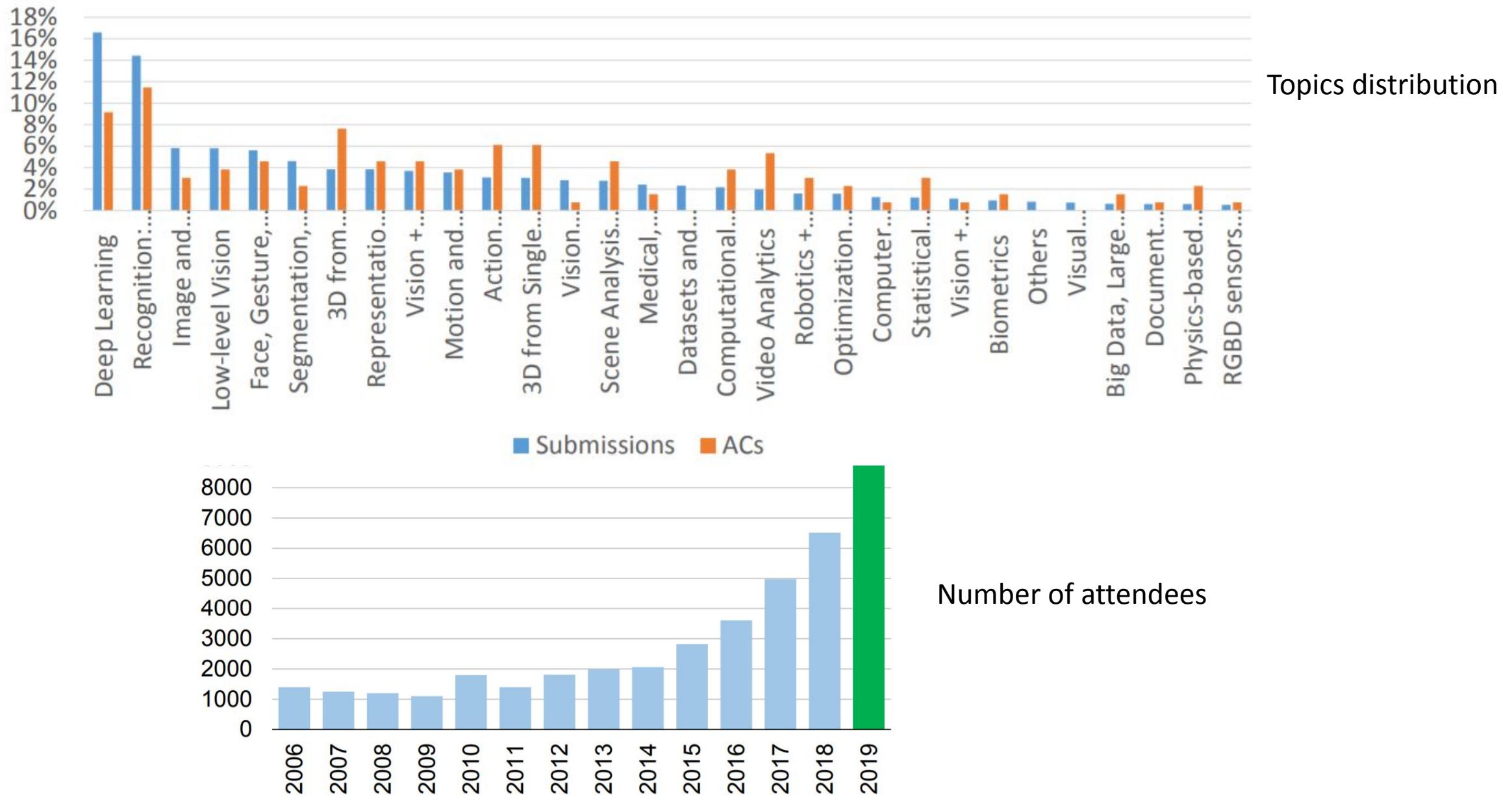
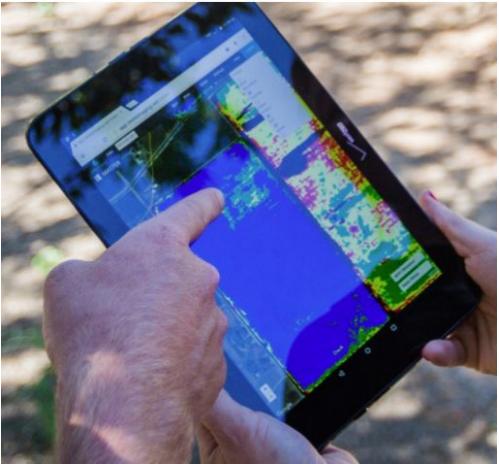


Figure 4.2.3

# CVPR 2019



# Start-ups developing Computer Vision



Ceres Imaging combines high-resolution aerial imagery with advanced analytics to help farms, orchards, and vineyards optimize irrigation strategy and performance



Pixee Medical has created the first computer-assisted orthopaedic surgery solution using augmented reality to support total knee arthroplasty.



"Out-of-stocks" detection - FocalSystems



OrCAM: Wearable device, for people who are blind or visually impaired, that reads text, recognizes faces, identifies products & more

TURNING IMAGERY INTO PROPERTY SPECIFIC INTELLIGENCE

**Property Profile**  
456 CENTER AVENUE, USA

BY PERIL ATTRIBUTES:	Roof Covering	Shingle
Hail	Roof Condition	Poor
	Roof Area	1,420 sq ft
PROPERTY CHARACTERISTICS	Building Extension	Yes
	Yard Debris	Yes

CHANGE DETECTED: BUILDING EXTENSION

An aerial photograph of a residential neighborhood. A specific house on Center Avenue is highlighted with a green circle, indicating a detected change such as a building extension.

Valuable property attributes combined with a simple API allow investors and insurers to better select properties, evaluate risk, and streamline underwriting processes.

# Lecture

## **Part 1: Fundamentals. Artificial Neural Networks**

Multi layer perceptron

Optimization. Loss functions

Neural Networks

## **Part 2: Convolutional neural networks, visual transformers, graph convolutional neural networks**

Convolutions. Convolutional layers. Pooling layers.

Attention mechanisms

Visual transformers

Graph convolutional neural networks

## **Part 3: Applications and case studies**

Object detection

Segmentation. Instance segmentation

Style transfer. Image generation

Visualizing and understanding

## **Part 4: Ethics of Artificial Intelligence and Robotics**

# Laboratories

1. Introduction to python, numpy.
2. Pytorch fundamentals. Implement a softmax classifier. Evaluating a model.
3. Convolutional neural networks. Fine tuning. Transfer learning. Hyperparameter tuning.
4. Semantic segmentation. Evaluating a semantic segmentation model.

# Projects

- Work either independently or in a team (2-3 students)
- From week 7 the lab will be related to the project
- Deadlines
  - Week 5: project proposal
  - Week 6: state of the art, find datasets to train and test on, (maybe) run some of the state of the art works and analyse their results
  - Week 9: first implementation.
  - Week 12: improve your first solution, experiments, documentation.
  - Week 14: have our own CV conference. 3 min teaser presentations, poster, vote for the best projects.
- **Documentation:**  
<https://www.overleaf.com/latex/templates/cvpr-2018-template/qgmrftfbqns>
- **Useful tips (Zachary Lipton):**  
<https://www.approximatelycorrect.com/2018/01/29/heuristics-technical-scientific-writing-machine-learning-perspective/>

# Grading

- Written examination: 40%
- Project and lab: 60 %

The final grade (average between written exam and project) should be at least 5

# Recommended resources

- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.
  - <http://www.deeplearningbook.org/>
- Langr, Jakub, and Vladimir Bok. GANs in Action. (2018)
- deeplearning.ai
  - <https://www.deeplearning.ai/>

*I hope you'll  
enjoy this course*

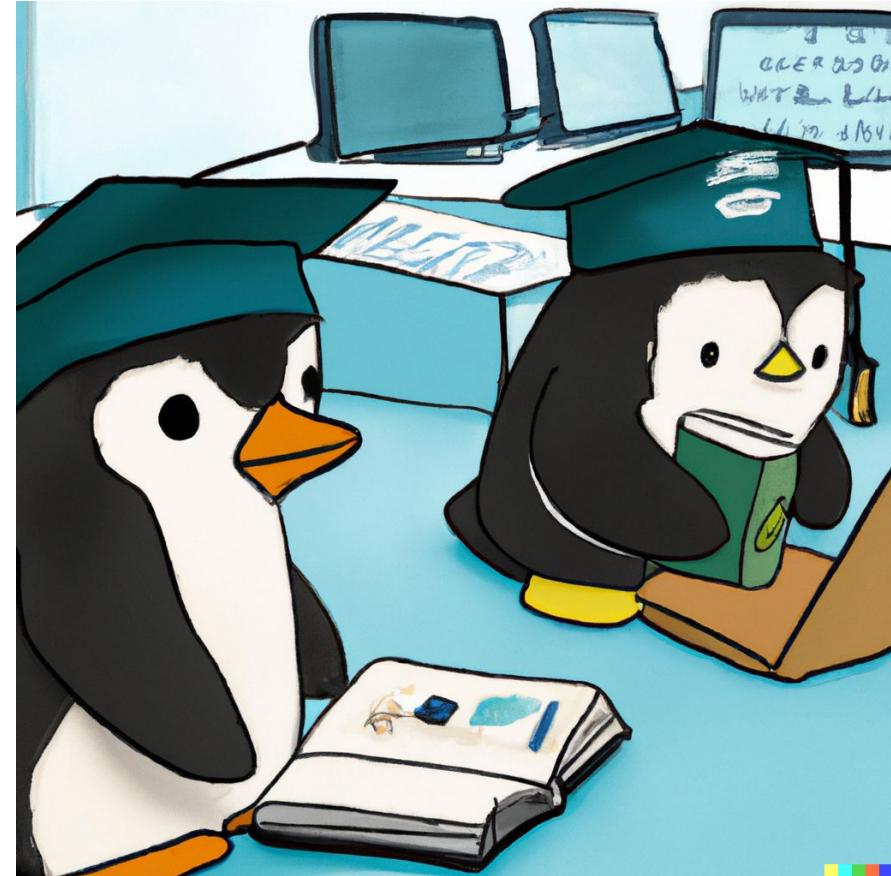


Image generated with DALL-E



Image generated with DALL-E: Teacher saying thank you

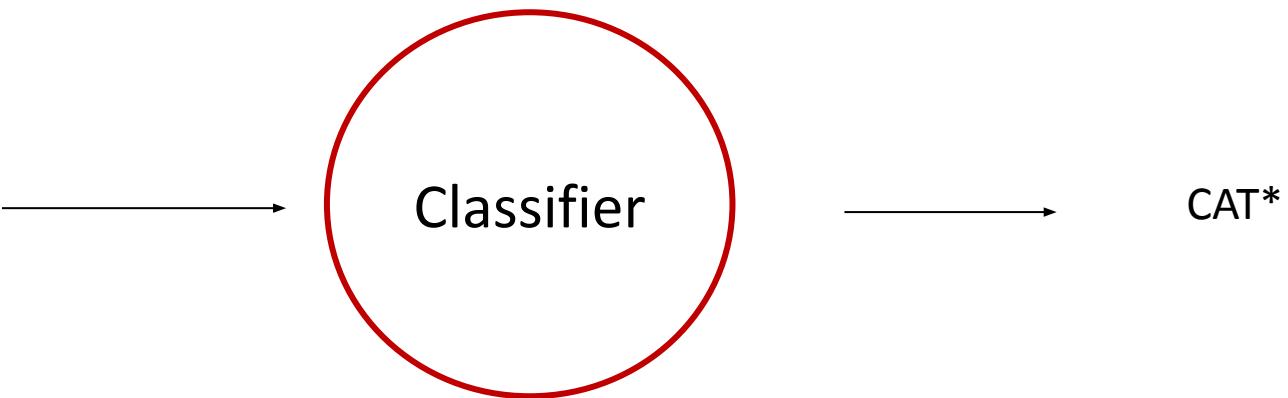


Image generated with DALL-E: Happy students learning about AI

# Computer vision and deep learning

Lecture 2

# Image classification I



\* assuming that we  
have a set of labels  $L =$   
 $\{cat, dog, ship \dots\}$

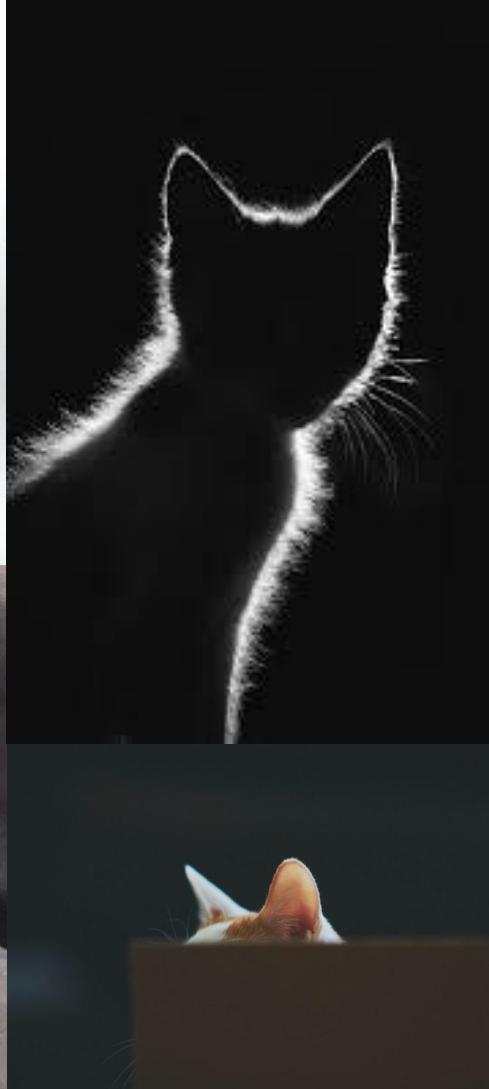
# Image classification II

- Unlike other computer science problems:
  - Find the maximum value in an array
  - Sort an array
  - Find the shortest distance between two nodes in a graph
- **No obvious way** to implement an algorithm for image classification

```
def image_classifier(image):  
    # ???  
    return class_label
```

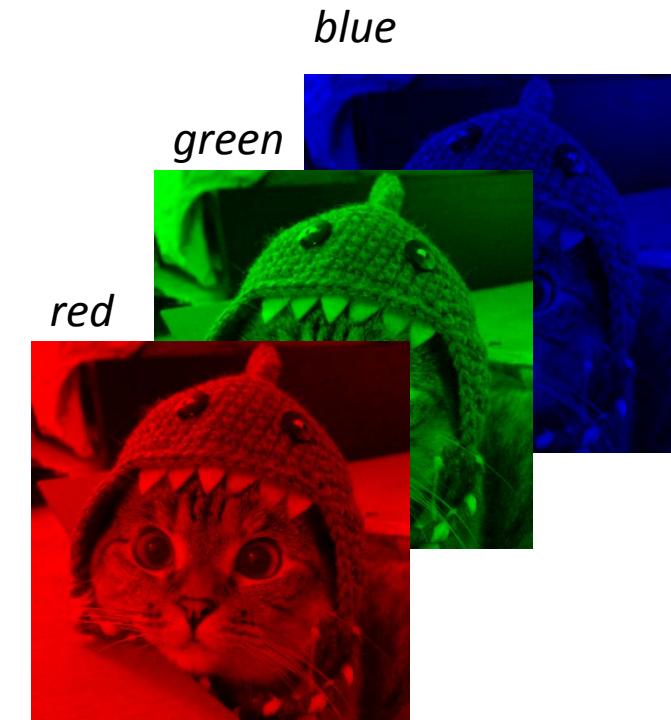
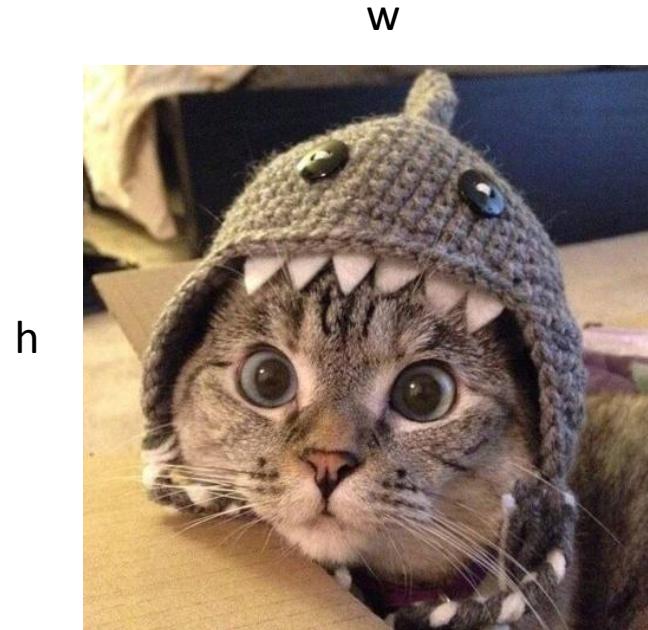
# Classification challenges

- View point variations
- Occlusions
- Scale variation
- Deformation
- Lightning conditions
- Background clutter
- Intra-class variation



# Linear classification

# Image representation



$w \times h \times 3$  values between [0, 255]

Flatten the image to a  
1D array

# Linear classification

- Define a function which maps image pixels  $x_i$  to class scores

$$f(x_i, W, b) = x_i \cdot W + b$$

weights    bias



Classifier  
 $f(x_i, W, b)$



N class scores

# Linear classification

CIFAR 10 example:

- 3 channel images, 32x32 pixels  $x_i \in \mathbb{R}^D$
- N = 10 class scores

$$f: \mathbb{R}^D \rightarrow \mathbb{R}^N \quad , f(x_i, W, b) = x_i \cdot W + b$$



Classifier  
 $f(x_i, W, b)$



N class scores

N = 10

$$D = 32 \times 32 \times 3 = 3072$$

# Linear classification

CIFAR 10 example:

- 3 channel images, 32x32 pixels,  $x_i$  size  $[1 \times 3072]$
- $N = 10$  class scores

$$x_i \in \mathbb{R}^D$$

$$f(x_i, W, b) = \boxed{x_i} \cdot \textcolor{red}{W} + b$$

$[1 \times D]$

$[1 \times 3072]$

$$f: \mathbb{R}^D \rightarrow \mathbb{R}^N$$



Classifier  
 $f(x_i, W, b)$



$N$  class scores

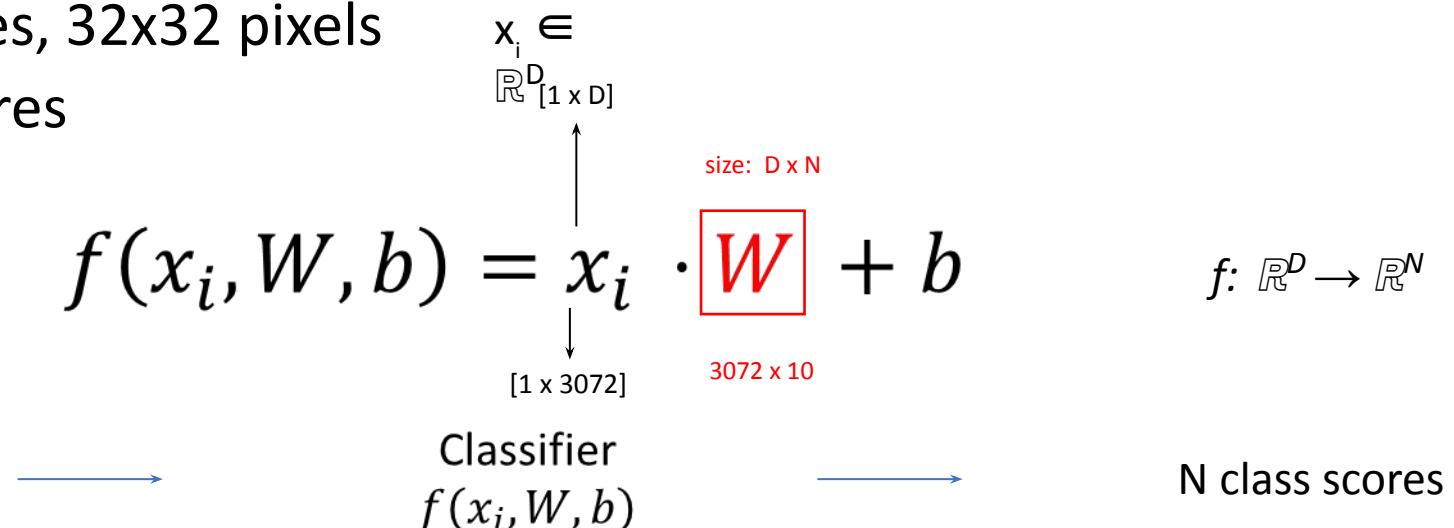
$N = 10$

$$D = 32 \times 32 \times 3 = 3072$$

# Linear classification

CIFAR 10 example:

- 3 channel images, 32x32 pixels
- N = 10 class scores

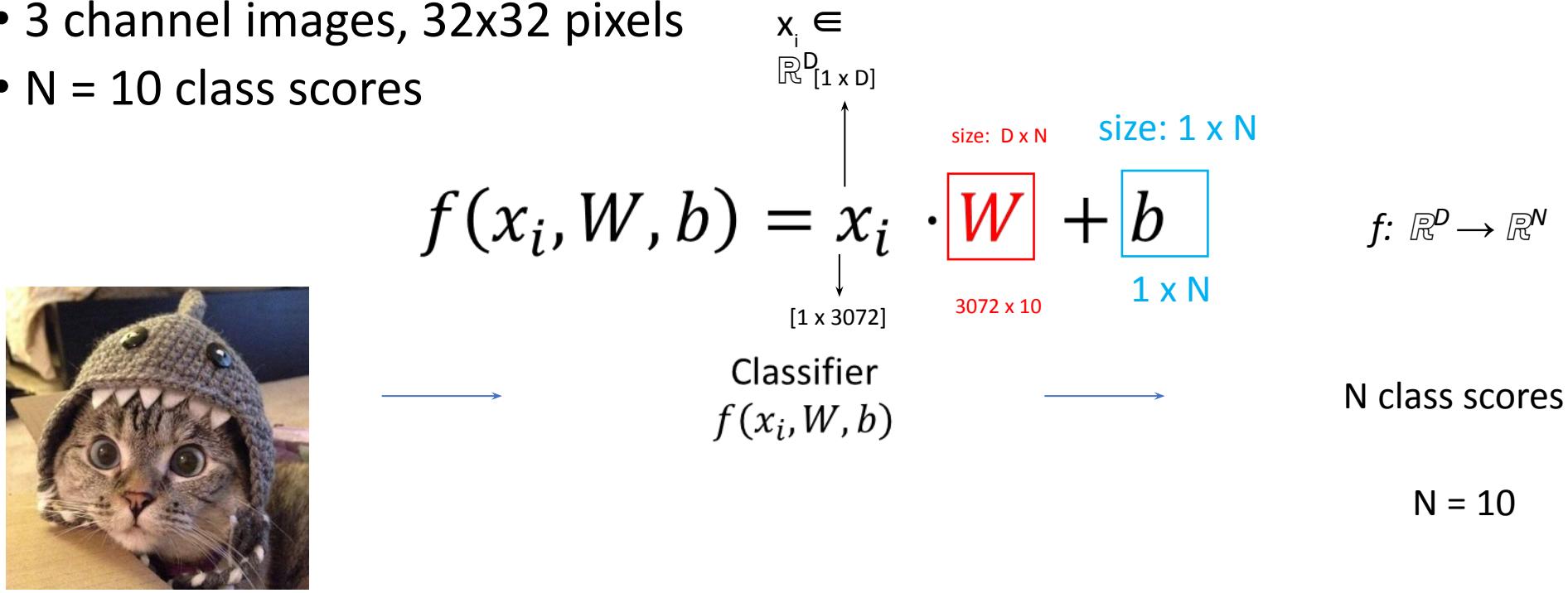


$$D = 32 \times 32 \times 3 = 3072$$

# Linear classification

# CIFAR 10 example:

- 3 channel images, 32x32 pixels
  - $N = 10$  class scores



$$D = 32 \times 32 \times 3 = 3072$$

# Linear classification - simplified example

weights W

x

57	77	200	128
----	----	-----	-----

.

0.7	-0.3	0
-0.3	0.5	0.25
2.5	3.1	0.4
1	-3	0.1

bias b

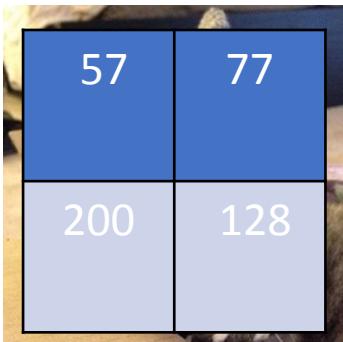
+

1.7	-2	3
-----	----	---

=

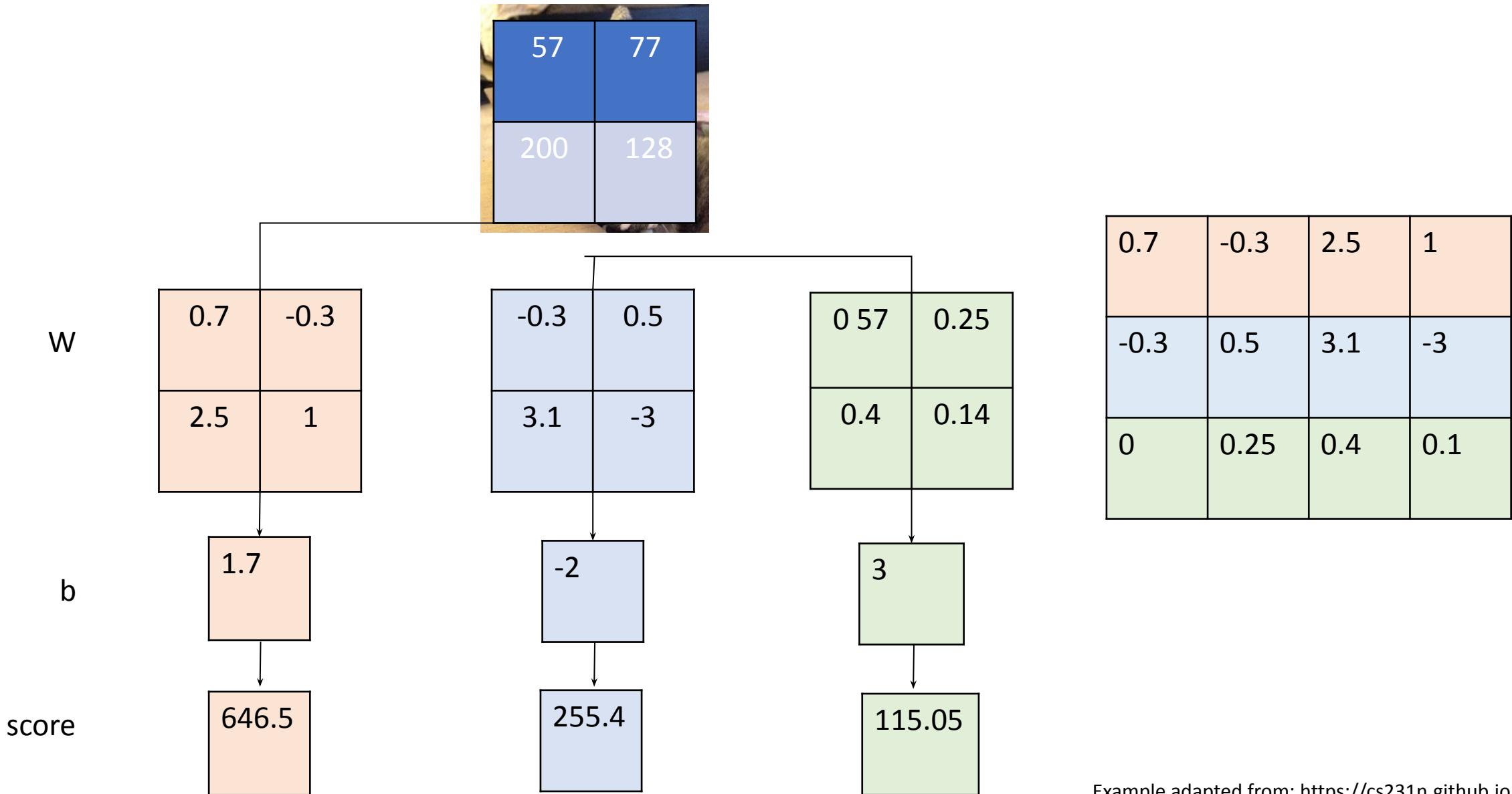
646.5	255.4	115.05
-------	-------	--------

cat      boat      shark  
score    score    score





# Linear classification - simplified example

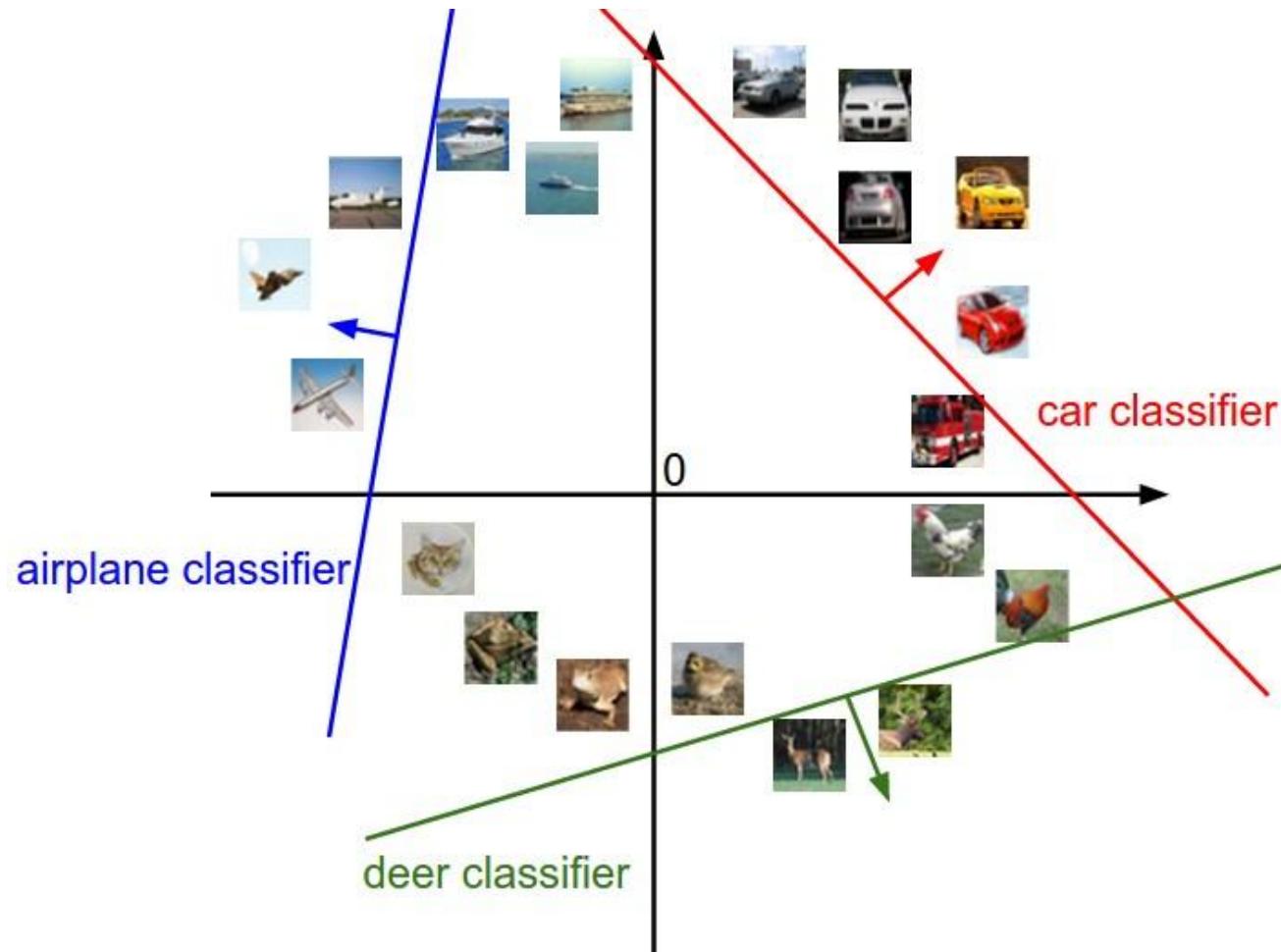


# Linear classification – visual interpretation



- Linear classification can be seen as a *template matching* process, where the weights are learned
- Each row of the weights matrix is a *prototype* (template) for one of the classes

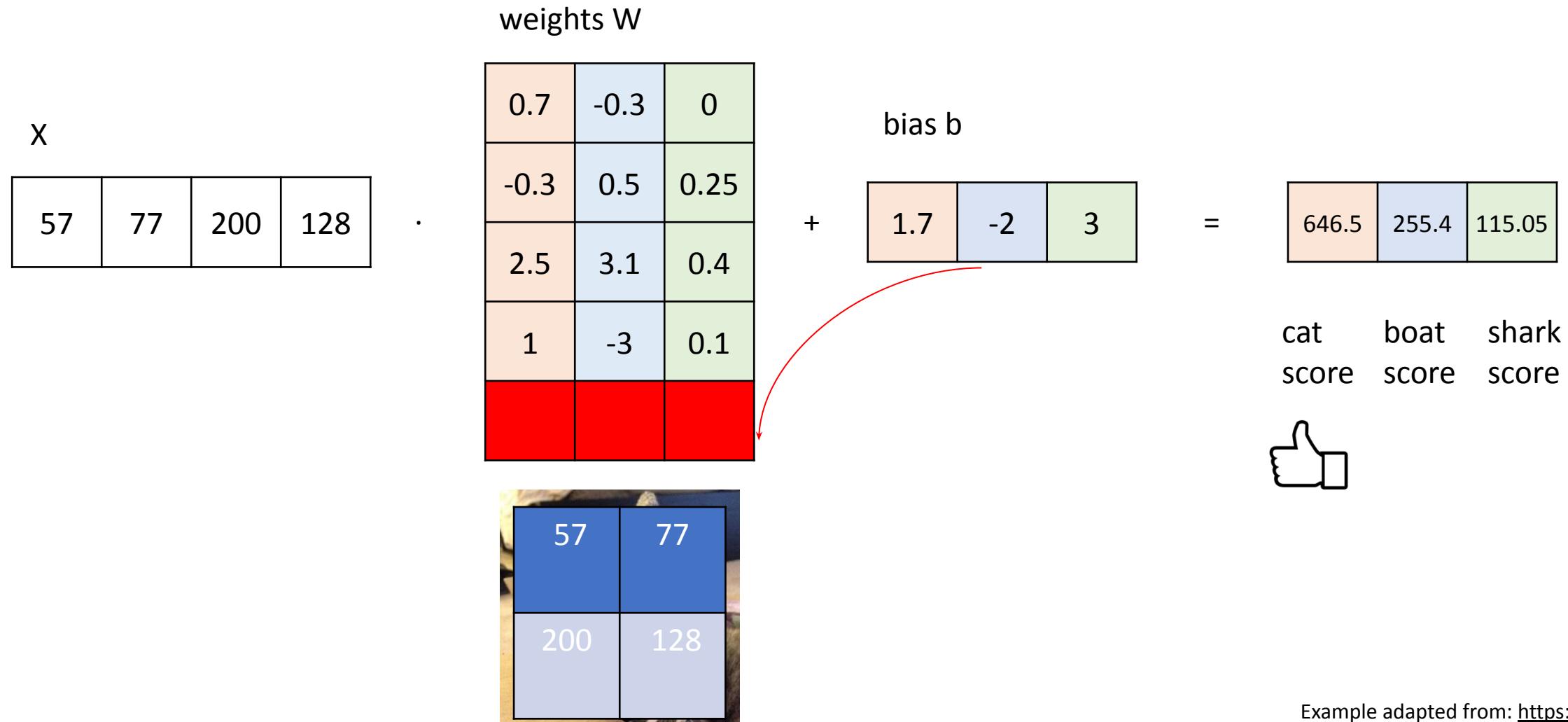
# Linear classification – geometric interpretation



Hyper-plane cutting up the space

# Linear classification – implementation hints

## The bias trick



# Linear classification – implementation hints

## The bias trick

X

57	77	200	128	1
----	----	-----	-----	---

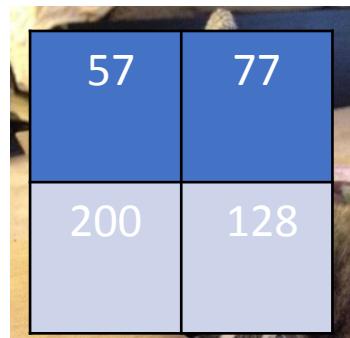
weights W

0.7	-0.3	0
-0.3	0.5	0.25
2.5	3.1	0.4
1	-3	0.1
1.7	-2	3

=

646.5	255.4	115.05
-------	-------	--------

cat      boat      shark  
score    score    score



bias b

# Linear classification

$$f(X, W, b) = X \cdot W + b$$

- Input  $X \rightarrow$  fixed  $x_i \in X$
- $N$ : number of classes  $\rightarrow$  fixed
- $W$  – weights  $\rightarrow$  we have control over their setting
- $b$  – bias  $\rightarrow$  we have control over their setting
  - $b$  doesn't actually interact with the data
- $f$  – ***score function*** - map raw pixels to class scores

*Goal: set  $W$  and  $b$  such that the correct class has the a higher value than in incorrect ones*

# Next steps

- Define a **loss function  $L$** 
  - $L$  - measures the quality of  $W$  and  $b$  based on how well the predicted scores agree with the ground truth labels
  - $L$  – how happy are we with the scores across the training data? How good is our (current) classifier?
- Find an efficient way to determine the parameters ( $W$  and  $b$ ) such that the loss function is minimized

# Loss function

$$L = \frac{1}{N} \sum_i^N L_i(f(x_i, W), y_i)$$

- N – number of training examples
- $x_i$  - i<sup>th</sup> image in our training dataset
- $y_i$  - ground truth label for the i<sup>th</sup> image in our training set (i.e. cat, boat or shark)

# Example – hinge loss or SVM loss

- $s_j = f(x_j, W)$  - the prediction of the  $i^{\text{th}}$  image in our training dataset
- m – margin

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + m)$$

# Example – hinge loss or SVM loss

- $s_j = f(x_j, W)$  - the prediction of the  $i^{\text{th}}$  image in our training dataset
- m – margin

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + m)$$

```
max(0,  $s_j - s_{y_i} + m$ )
if  $s_j - s_{y_i} + m < 0$ :
    return 0
else:
    return  $s_j - s_{y_i} + m$ 
```

# Example – hinge loss

- $s_j = f(x_j, W)$  - the prediction of the  $i^{\text{th}}$  image in our training dataset
- $m$  – margin

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + m)$$

$$= \sum_{j \neq y_i} \begin{cases} 0, & s_{y_i} \geq s_j + m \\ s_j - s_{y_i}, & \text{otherwise} \end{cases}$$

```
max(0, sj - syi + m)
  if syi ≥ sj + m
    return 0
  else:
    return sj - syi + m
```

# Example – hinge loss

- $s_j = f(x_j, W)$  - the prediction of the  $i^{\text{th}}$  image in our training dataset
- $m$  – margin

$$\begin{aligned} L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + m) \\ &= \sum_{j \neq y_i} \begin{cases} 0, & s_{y_i} \geq s_j + m \\ s_j - s_{y_i}, & \text{otherwise} \end{cases} \end{aligned}$$

```
max(0,  $s_j - s_{y_i} + m$ )
if  $s_{y_i} \geq s_j + m$ 
    return 0
else:
    return  $s_j - s_{y_i} + m$ 
```

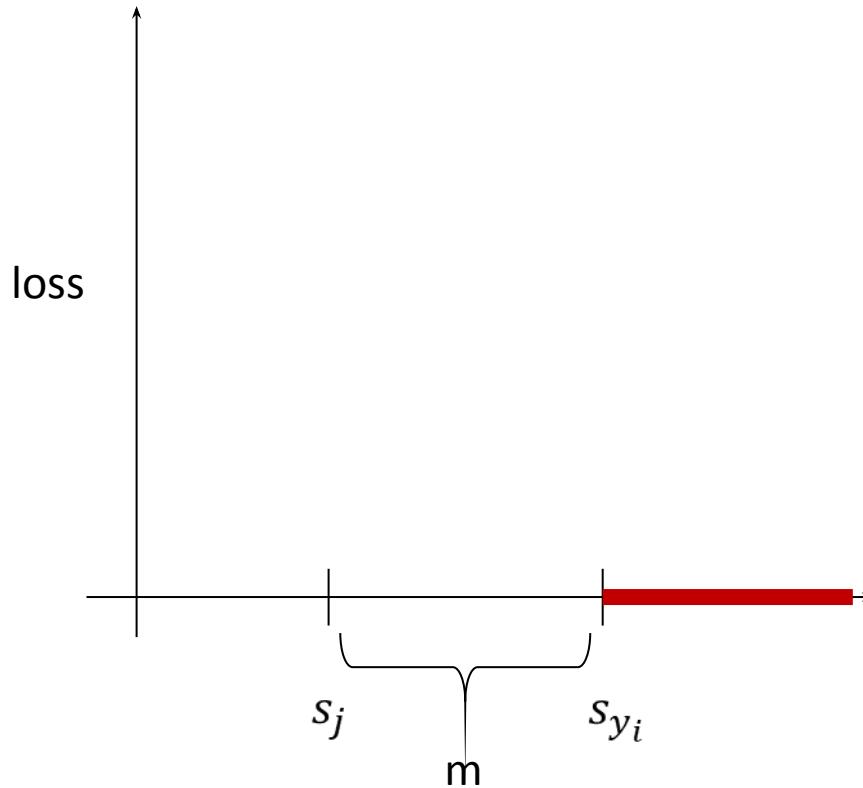
Intuition: we are “happy” if the score of the correct class is larger, by a margin of at least  $m$ , than the score of the other classes

# Example – hinge loss

- $s_j = f(x_j, W)$  - the prediction of the  $i^{\text{th}}$  image in our training dataset
- $m$  – margin

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + m)$$

$$= \sum_{j \neq y_i} \begin{cases} 0, & \underline{s_{y_i} \geq s_j + m} \\ s_j - s_{y_i}, & \text{otherwise} \end{cases}$$

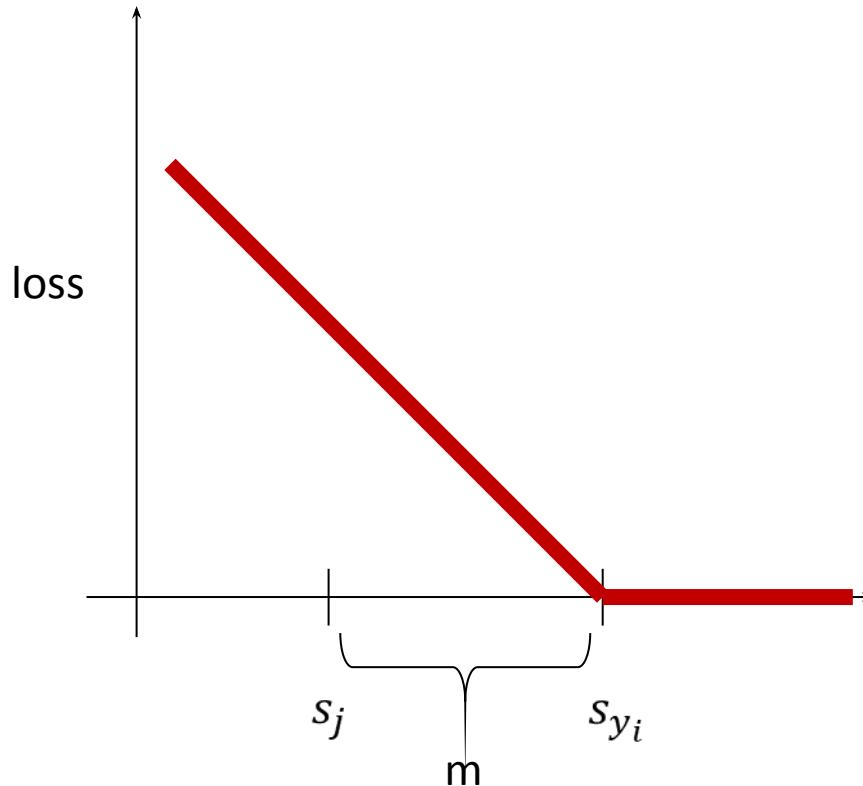


# Example – hinge loss

- $s_j = f(x_j, W)$  - the prediction of the  $i^{\text{th}}$  image in our training dataset
- $m$  – margin

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + m)$$

$$= \sum_{j \neq y_i} \begin{cases} 0, & s_{y_i} \geq s_j + m \\ s_j - s_{y_i}, & \text{otherwise} \end{cases}$$



# Example – hinge loss

Class	A cat wearing a shark hood.	A sailboat on the water.	A dolphin in the ocean.
cat	3.2	0.1	0.3
boat	-5	14	8
shark	10	5	14
Hinge loss			

$$L_{H_i} = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + m)$$

Let's set  $m = 1$

For image 1



$$\begin{aligned} &= \max(0, -5 - 3.2 + 1) + \max(0, 10 - 3.2 + 1) \\ &= 0 + 7.8 \\ &= 7.8 \end{aligned}$$

# Example – hinge loss

Class	A cat's face with a shark mouth hood.	A sailboat on the water.	A dolphin in the ocean.
cat	3.2	0.1	0.3
boat	-5	14	8
shark	10	5	14
<b>Hinge loss</b>	<b>7.8</b>		

$$L_{H_i} = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + m)$$

Let's set  $m = 1$

For image 1



$$\begin{aligned} &= \max(0, -5 - 3.2 + 1) + \max(0, 10 - 3.2 + 1) \\ &= 0 + 7.8 \\ &= 7.8 \end{aligned}$$

# Example – hinge loss

Class			
cat	3.2	0.1	0.3
boat	-5	14	8
shark	10	5	14
<b>Hinge loss</b>	<b>7.8</b>	<b>0</b>	<b>0</b>

Total loss L

$$L = \frac{1}{N} \sum_{i=1}^N L_i$$

$$L = (7.8 + 0 + 0) / 3 = 2.6$$

$$L_{Hi} = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + m)$$

Let's set m = 1



For image 2

$$= \max(0, 0.1 - 14 + 1) + \max(0, 5 - 14 + 1)$$

$$= 0 + 0$$

$$= 0$$



For image 3

$$= \max(0, 0.3 - 14 + 1) + \max(0, 8 - 14 + 1)$$

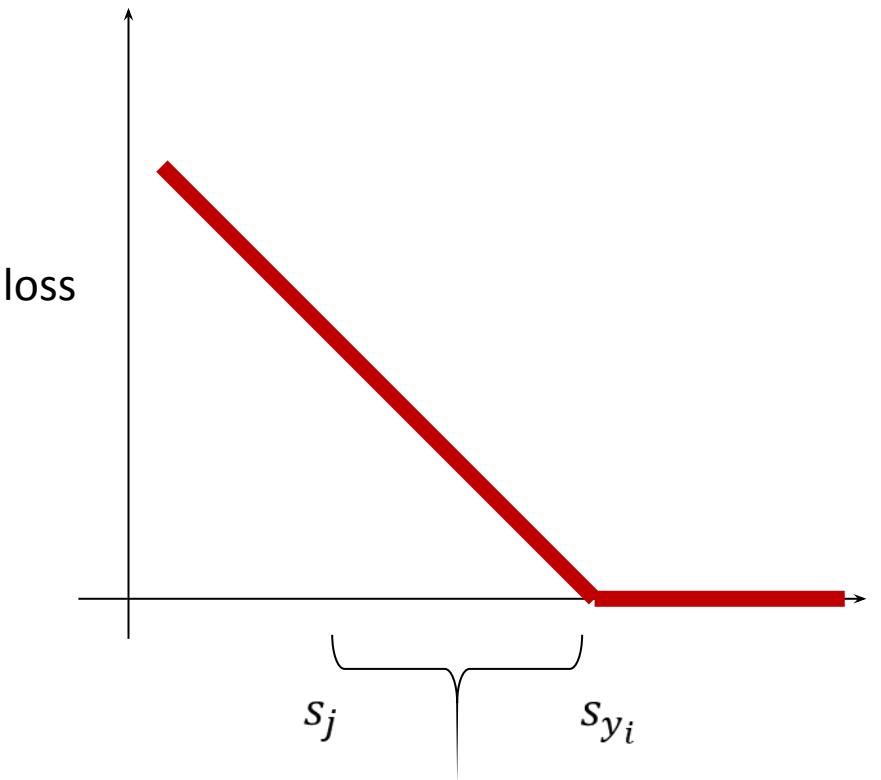
$$= 0 + 0$$

$$= 0$$

# Hinge loss

- What is the min value and the max value of hinge loss?

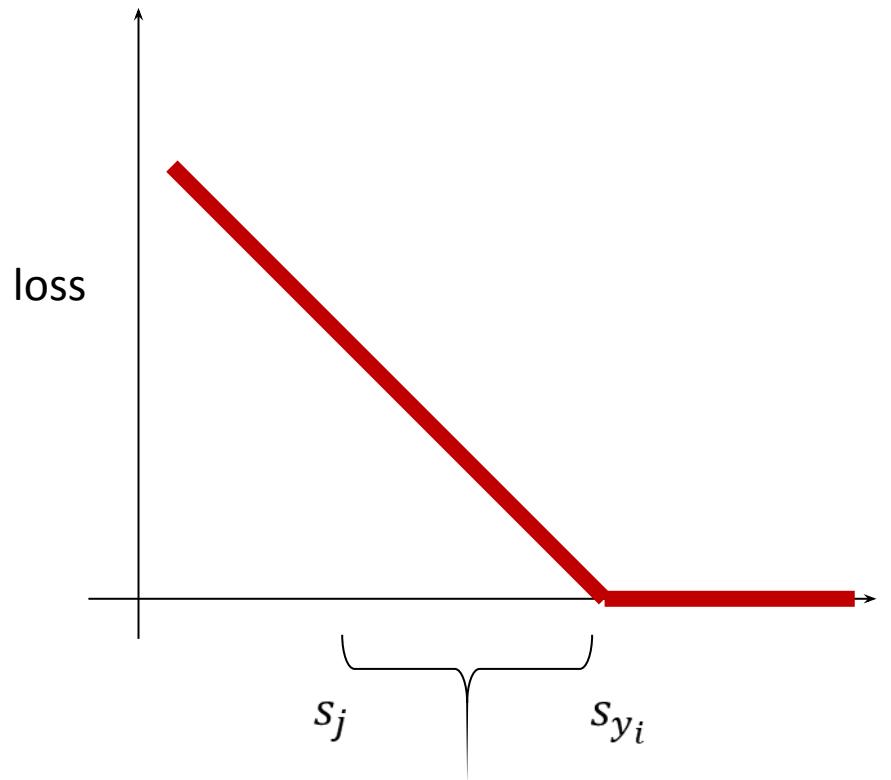
$$L_{H_i} = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + m)$$



# Hinge loss

- What is the min value and the max value of hinge loss?
  - min is 0: the correct score is much larger ( $\geq m$ ) than the incorrect scores
  - max is infinity
- In the beginning of the training ( $W \approx 0$ ) so all scores  $\approx 0$ . What is the value of L ?

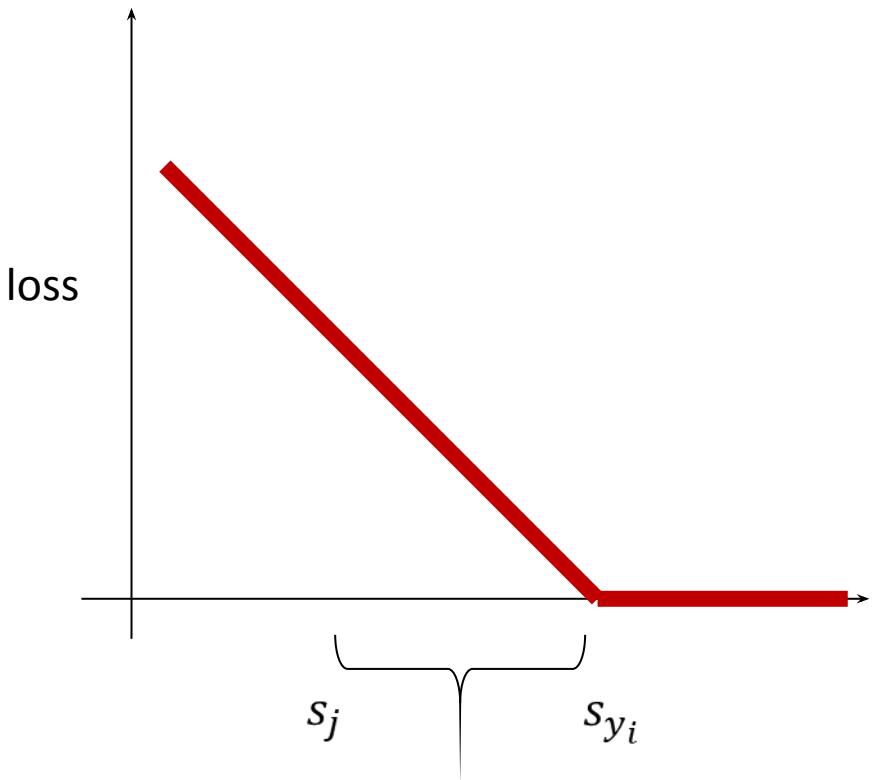
$$L_{H_i} = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + m)$$



# Hinge loss

- What is the min value and the max value of hinge loss?
- In the beginning of the training ( $W \approx 0$ ) so all scores  $\approx 0$ . What is the value of  $L$ ?
  - $N - 1$

$$L_{H_i} = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + m)$$



# Regularization

- Suppose that you found some values for the parameters  $W$  such that the loss function is 0. Do you think there is a single setting for the parameters  $W$ , such that the loss is 0?

# Regularization

- Suppose that you found some values for the parameters  $W$  such that the loss function is 0. Do you think there is a single setting for the parameters  $W$ , such that the loss is 0?
  - $W, 2W, 3W, \alpha W$
  - extend the loss function with a **regularization penalty** to discourage large weights

# Regularization

$$L = \frac{1}{N} \sum_{i=1}^N L_i + \frac{\text{Regularization loss}}{\lambda R(W)}$$

Data loss

- Prevent overfitting (the model will perform "too" well on the training data)
- Make the model simpler



Occam's razor: "the simplest explanation is most likely the right one"

# Regularization

## L2 regularization

- 

$$R(W) = \sum_r \sum_c W_{r,c}^2$$

$$x = [2, 0.1, 0.1, 0.5]$$

$$W1 = [0.2, 0.8, 0.2, 0.1] \rightarrow R(W) = 0.73$$

$$W1 \cdot X = 2$$

$$W2 \cdot X = 2$$

$$W2 = [1, 0, 0, 0] \rightarrow R(W) = 1$$

# Regularization

## L2 regularization

- 

$$R(W) = \sum_r \sum_c W_{r,c}^2$$

$x = [2, 0.1, 0.1, 0.5]$

$W1 = [0.2, 0.8, 0.2, 0.1] \rightarrow R(W) = 0.73$



$W2 = [1, 0, 0, 0] \rightarrow R(W) = 1$

$W1 \cdot X = 2$

$W2 \cdot X = 2$

- Prefers to “spread out” the weights such that each element has the same influence on the prediction

# Regularization

## L1 regularization

- $R(W) = \sum_r \sum_c |W_{r,c}|$

$$x = [2, 0.1, 0.1, 0.5]$$

$$W1 \cdot X = 2$$

$$W1 = [0.2, 0.8, 0.2, 0.1] \rightarrow R(W) = 1.4$$

$$W2 \cdot X = 2$$

$$W2 = [1, 0, 0, 0] \rightarrow R(W) = 1$$

# Regularization

## L1 regularization

- 

$$R(W) = \sum_r \sum_c |W_{r,c}|$$

$$x = [2, 0.1, 0.1, 0.5]$$

$$W_1 \cdot X = 2$$

$$W_1 = [0.2, 0.8, 0.2, 0.1] \rightarrow R(W) = 1.4$$

$$W_2 \cdot X = 2$$

$$W_2 = [1, 0, 0, 0] \rightarrow R(W) = 1$$



- Prefers more sparse weight matrices, most of the values in W should be close to 0, except for some values where the weights are allowed to deviate
- Feature selection

# Example – softmax classifier

So far, we don't have interpretation for the outputs of the score function

Class	
cat	
boat	-5
shark	10

What if we would like to determine the probability of each class?

# Example – softmax classifier

So far, we don't have interpretation for the outputs of the score function

What if we would like to determine the probability of each class?

Class	
cat	3.2
boat	-5
shark	10



$$s = f(x_i, W)$$

Softmax function

$$P(Y = c | X = x_i) = \frac{e^{s_c}}{\sum_j e^{s_j}}$$

# Example – softmax classifier

So far, we don't have interpretation for the outputs of the score function

What if we would like to determine the probability of each class?

Class	
cat	3.2
boat	-5
shark	10

$$s = f(x_i, W)$$

Softmax function

$$P(Y = c | X = x_i) = \frac{e^{s_c}}{\sum_j e^{s_j}}$$

normalize

# Example – softmax classifier

So far, we don't have interpretation for the outputs of the score function

What if we would like to determine the probability of each class?

Class	
cat	3.2
boat	-5
shark	10

$$s = f(x_i, W)$$

Softmax function

$$P(Y = c | X = x_i) = \frac{e^{s_c}}{\sum_j e^{s_j}}$$

normalize

Class	
cat	3.2
boat	-5
shark	10

$s =$   
 $[$   
 3.2  
 -5  
 10  
 $]$

$$e^s =$$

$$[$$

$$24.5325$$

$$0.0067$$

$$22026.4658$$

$$]$$

$$\frac{e^{s_c}}{\sum_j e^{s_j}} =$$

$$[$$

$$0.0011125$$

$$0.0000003$$

$$0.9988872$$

$$]$$

softmax function

$$\sum_j e^{s_j} = 22026.4658$$

softmax function

$[$   
 0.0011125  
 0.0000003  
 0.9988872  
 $]$

vs

hardmax function, one hot encoding

$[$   
 0  
 0  
 1  
 $]$

$$L_i = - \sum_{j=1}^C y_j \log(s_j)$$

$s_j$  - prediction for class j

$$= -y_c \log(s_c)$$

$$= -\log(s_c)$$

Intuition:

- The loss  $L_i = -\log(s_c)$  should be as small as possible  
→ the correct probability class should be as big as possible

$$L_i = - \sum_{j=1}^C y_j \log(s_j)$$

$s_j$  - prediction for class j

$$= -y_c \log(s_c)$$

$$= -\log(s_c)$$

Intuition:

- The loss  $L_i = -\log(s_c)$  should be as small as possible  
→ the correct probability class should be as big as possible

This loss function just looks at the correct class from the ground truth and tries to make the predicted probability of that class as big as possible

$$L_i = -\log(s_c)$$

$$L = \frac{1}{N} \sum_i^N L_i$$

- What is the minimum and maximum value of  $L_i$ ?

$$L_i = -\log(s_c)$$

$$L = \frac{1}{N} \sum_i^N L_i$$

- What is the minimum and maximum value of  $L_i$ ?
  - Min 0
  - Max infinity

$$L_i = -\log(s_c)$$

$$L = \frac{1}{N} \sum_i^N L_i$$

- What is the minimum and maximum value of the  $L_i$ ?
  - Min 0
  - Max infinity
- At initialization, when all the predicted scores will be approximately equal, what is the value of  $L_i$ ?

$$L_i = -\log(s_c)$$

$$L = \frac{1}{N} \sum_i^N L_i$$

- What is the minimum and maximum value of  $L_i$ ?
  - Min 0
  - Max infinity
- At initialization, when all the predicted scores will be approximately equal, what is the value of  $L_i$ ?
  - $-\log(1/C) = \log(C)$

# Optimization

How can we find the best value for W?

# Gradient descent



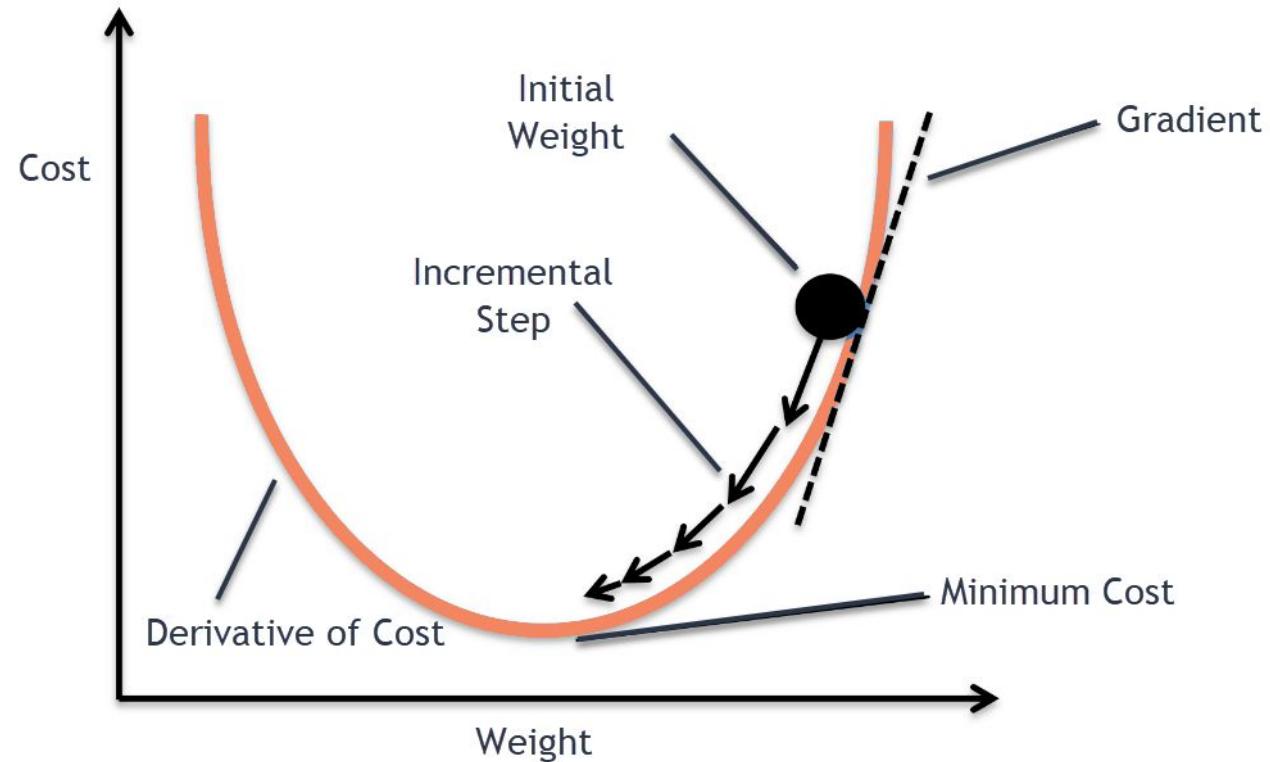
Follow the slope!

# Follow the slope

- In 1D the derivative of a function is:
  - Slope of the “line”

$$f'(a) = \frac{\partial f(x)}{\partial x} = \lim_{h \rightarrow 0} \frac{f(a + h) - f(a)}{h}$$

# Follow the slope



# Follow the slope

- In N dimensions, the gradient is a vector containing the partial derivative along each dimensions

## Numerical gradient

w [0.3, 0.24, -1.9, 0.22, 0.3, 0.9, -1.7, 1.5, 0.29, ...]

**loss 1.2345**

gradient

?

# Numerical gradient

w

[0.3, 0.24, -1.9, 0.22, 0.3, 0.9, -1.7, 1.5, 0.29, ...]

**loss 1.2345**

w + h

h = 0.0001

[0.3001, 0.24, -1.9, 0.22, 0.3, 0.9, -1.7, 1.5, 0.29, ...]

**loss 1.2331**

= 0.3 + h  
= 0.3 + 0.0001

gradient

?

$$f'(a) = \frac{\partial f(x)}{\partial x} = \lim_{h \rightarrow 0} \frac{f(a + h) - f(a)}{h}$$

# Numerical gradient

w

[0.3, 0.24, -1.9, 0.22, 0.3, 0.9, -1.7, 1.5, 0.29, ...]

**loss 1.2345**

w + h

h = 0.0001

[0.3001, 0.24, -1.9, 0.22, 0.3, 0.9, -1.7, 1.5, 0.29, ...]

**loss 1.2331**

$$= 0.3 + h$$

$$= 0.3 + 0.0001$$

gradient

**$[(1.2345 - 1.2331)/0.0001, \dots]$**

# Numerical gradient

w

[0.3, 0.24, -1.9, 0.22, 0.3, 0.9, -1.7, 1.5, 0.29, ...]

**loss 1.2345**

w + h

h = 0.0001

[0.3001, 0.24, -1.9, 0.22, 0.3, 0.9, -1.7, 1.5, 0.29, ...]

**loss 1.2331**

$$\begin{aligned} &= 0.3 + h \\ &= 0.3 + 0.0001 \end{aligned}$$

gradient

**[23.5, ...]**

# Numerical gradient

w

[0.3, 0.24, -1.9, 0.22, 0.3, 0.9, -1.7, 1.5, 0.29, ...]

**loss 1.2345**

w + h

h = 0.0001

[0.3, 0.2401, -1.9, 0.22, 0.3, 0.9, -1.7, 1.5, 0.29, ...]

**loss 1.2345**

= 0.24 + h

gradient

**[23.5, ... ]**

# Numerical gradient

w

[0.3, 0.24, -1.9, 0.22, 0.3, 0.9, -1.7, 1.5, 0.29, ...]

**loss 1.2345**

w + h

h = 0.0001

[0.3001, 0.2401, -1.9, 0.22, 0.3, 0.9, -1.7, 1.5, 0.29, ...]

**loss 1.2345**

= 0.24 + h

gradient

[23.5, (1.2345 – 1.2345)/0.0001, ....]

# Analytical gradient

- Numerical gradient: slow and approximate
- **Analytical gradient.** Given:

$$L = \frac{1}{N} \sum_{i=1}^N L_i(x_i, y_i, W) + \lambda R(W)$$
$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + m)$$

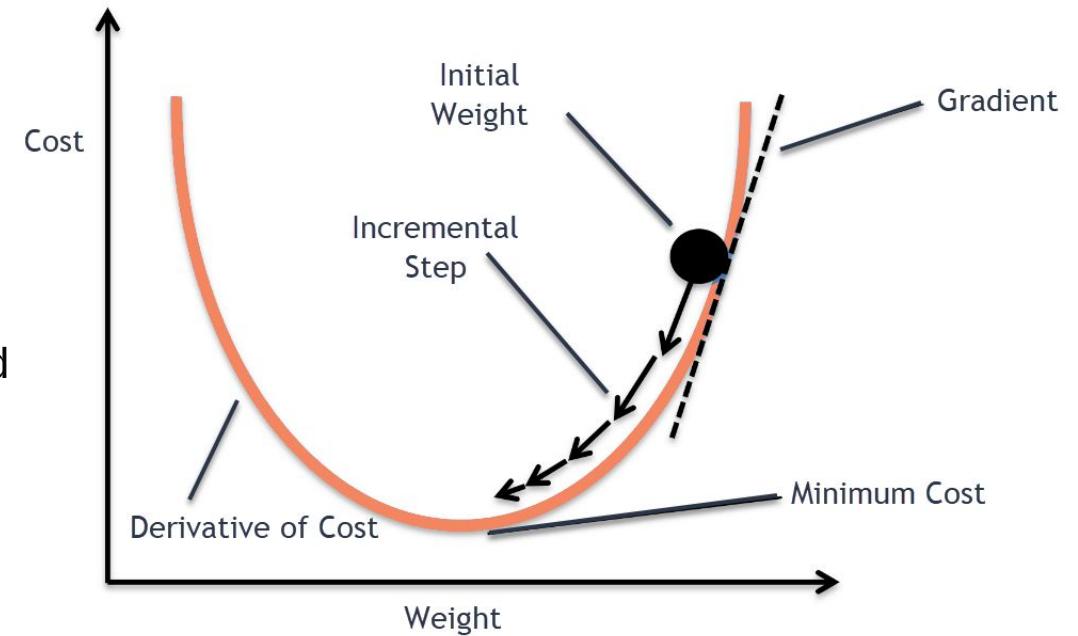
, we need to compute

$$\frac{\partial L}{\partial W} = \frac{1}{N} \sum_1^N \frac{\partial L_i(x_i, y_i, W)}{\partial W} + \lambda \frac{\partial R(W)}{\partial W}$$

and we can use calculus for this!

# Gradient descent

```
while True:  
    W_gradient = compute_gradient(loss_func, d)  
    # update parameters  
    W += -lr*W_gradient
```



# Mini-batch gradient descent

Gradient descent is infeasible to compute if we have a large dataset

- Update the weights using a **mini-batch** of examples
- Common sizes for the batch: 32, 64, 128

```
while True:  
    batch = sample_batch(data)  
    W_gradient = compute_gradient(loss_func, batch, W)  
    # update parameters  
    W += -lr*W_gradient
```

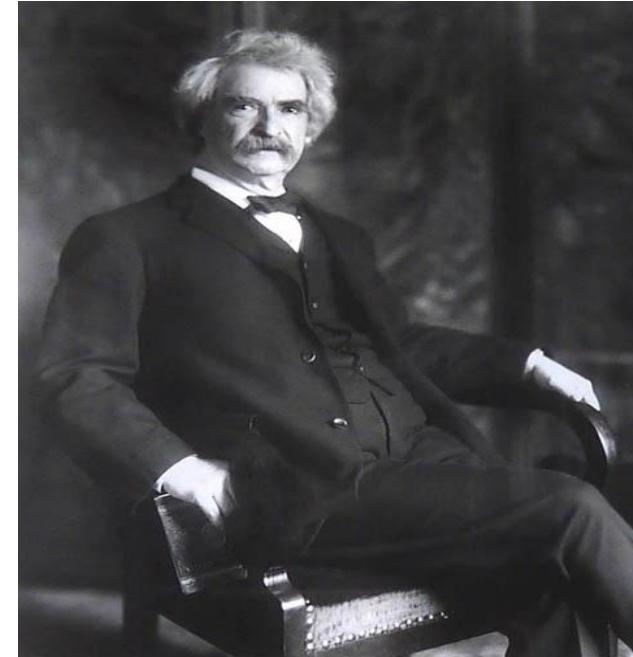
# What did we learn today?

- Score function
- Loss function
  - Hinge loss
  - Softmax loss
- Regularization
- Optimization
  - Gradient descent
  - Mini batch gradient descent

# Evaluating a model's performance

*"Facts are stubborn things, but statistics are more pliable. "* – Mark Twain

*"There are three kinds of lies: lies, damned lies, and statistics. "* – popularized by  
Mark Twain



Mark Twain

## Evaluation metrics

- if  $s_{y_i} \geq s_j + m$   
    return 0
- else:  
    return  $s_j - s_{y_i} + m$

# Evaluation metrics

- True positives (TP)
- False positives (FP)
- True negatives (TN)
- False negatives (FN)
- Accuracy: The percent of predictions that the model got right
  - **Accuracy will yield misleading results if the data set is unbalanced!!!**

$$AC = \frac{TP + TN}{TP + TN + FP + FN}$$

# Tumour classification

<b>True Positive (TP):</b> <ul style="list-style-type: none"><li>• Reality: Malignant</li><li>• ML model predicted: Malignant</li><li>• Number of TP results: 1</li></ul>	<b>False Positive (FP):</b> <ul style="list-style-type: none"><li>• Reality: Benign</li><li>• ML model predicted: Malignant</li><li>• Number of FP results: 1</li></ul>
<b>False Negative (FN):</b> <ul style="list-style-type: none"><li>• Reality: Malignant</li><li>• ML model predicted: Benign</li><li>• Number of FN results: 8</li></ul>	<b>True Negative (TN):</b> <ul style="list-style-type: none"><li>• Reality: Benign</li><li>• ML model predicted: Benign</li><li>• Number of TN results: 90</li></ul>

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{1 + 90}{1 + 90 + 1 + 8} = 0.91$$

# Confusion matrix

		True condition			
Total population	Condition positive	Condition negative	Prevalence = $\frac{\sum \text{Condition positive}}{\sum \text{Total population}}$	Accuracy (ACC) = $\frac{\sum \text{True positive} + \sum \text{True negative}}{\sum \text{Total population}}$	
Predicted condition	Predicted condition positive	True positive	False positive, Type I error	Positive predictive value (PPV), Precision = $\frac{\sum \text{True positive}}{\sum \text{Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\sum \text{False positive}}{\sum \text{Predicted condition positive}}$
	Predicted condition negative	False negative, Type II error	True negative	False omission rate (FOR) = $\frac{\sum \text{False negative}}{\sum \text{Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\sum \text{True negative}}{\sum \text{Predicted condition negative}}$
	True positive rate (TPR), Recall, Sensitivity, probability of detection, Power = $\frac{\sum \text{True positive}}{\sum \text{Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\sum \text{False positive}}{\sum \text{Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR}^+}{\text{LR}^-}$	$F_1$ score = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
	False negative rate (FNR), Miss rate = $\frac{\sum \text{False negative}}{\sum \text{Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) = $\frac{\sum \text{True negative}}{\sum \text{Condition negative}}$	Negative likelihood ratio (LR-) = $\frac{\text{FNR}}{\text{TNR}}$		

# Classification metrics

a

		ac	sn	pr	$F_1$	MCC
1	○○○○○	0.8	0	-	0	-1
2	○○○○○	0.8	1	0.8	0.9	-
3	○○○○○	0.8	0.8	0.8	0.8	0.6

b

		ac	sn	pr	$F_1$	MCC
1	○○○○○	0.7	0.8	0.8	0.8	-0.2
2	○○○○○	0.8	0.8	1	0.9	0.6
3	○○○○○	0.8	0.8	0.8	0.8	0.6

c

		ac	sn	pr	$F_1$	MCC
1	○○○○○	0.8	1	0.8	0.9	-
2	○○○○○	0.4	0.4	0.8	0.6	-0.3
3	○○○○○	0.9	1	0.8	0.9	0.8

d

Actual - ● + ○ Predicted - ■ + □ TN ○ FN ○ FP ○ TP ○

observations that do not contribute to the corresponding metric are struck through with a red line

$$MCC = \phi = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

The relevance of TP, FP, TN, FN will depend on the purpose of the classifier and motivate the choice of metric

$$ac = \frac{TP + TN}{TP + TN + FP + FN}$$

$$rc = sn = \frac{TP}{TP + FN}$$

$$pr = \frac{TP}{TP + FP}$$

$$F_\beta = (1 + \beta)^2 \frac{pr \cdot rc}{\beta^2 \cdot pr + rc}$$

$$F_1 = 2 \frac{pr \cdot rc}{pr + rc} = \frac{2 \cdot TP}{2TP + FP + FN}$$

# Precision vs. recall

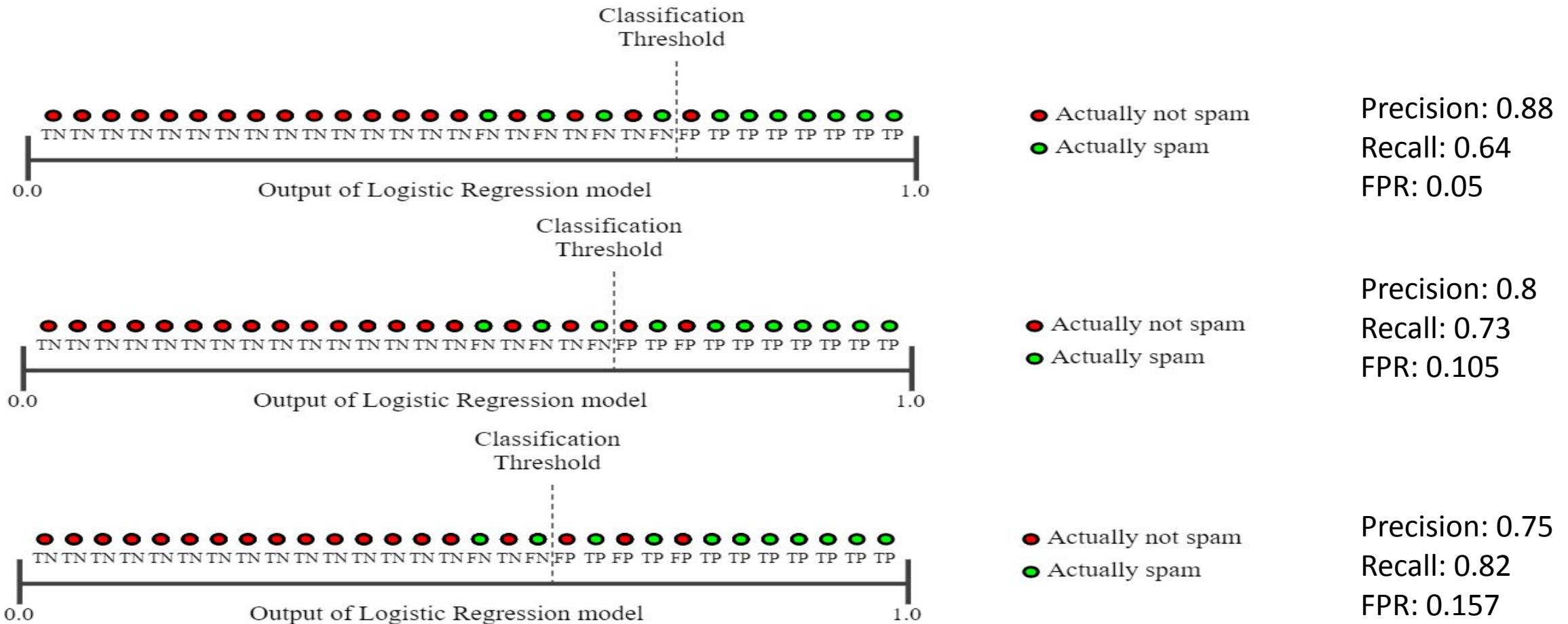
- *Precision*: spam detection, predict when to launch a satellite, pregnancy tests etc.
- *Recall*: airport security (make sure that every suspicious event is investigated), cancer prediction, detecting credit card frauds etc.

# Precision, Recall, F1 score

- Accuracy
  - The percent of prediction that the model got right
- Precision
  - The ability of a model to identify **only** relevant instances
- Recall
  - The ability of a model to identify **all** relevant instances
- F1 score
  - Harmonic mean between precision and recall

$$F_1 = 2 \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

# Precision vs. recall

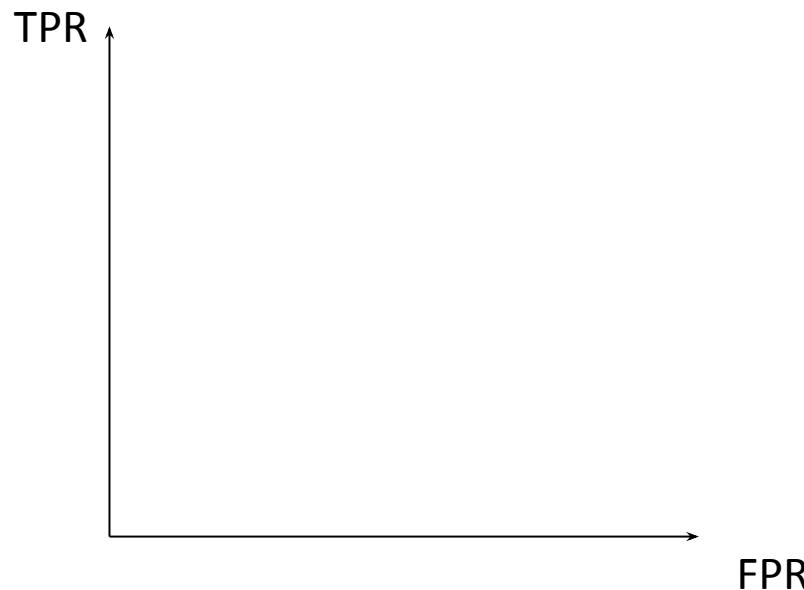


# Receiver Operating characteristic curve

## ROC curve

- Shows the performance of a classifier at different classification thresholds

False Positive Rate (1- specificity) – x axis  
True Positive Rate (sensitivity) – y axis



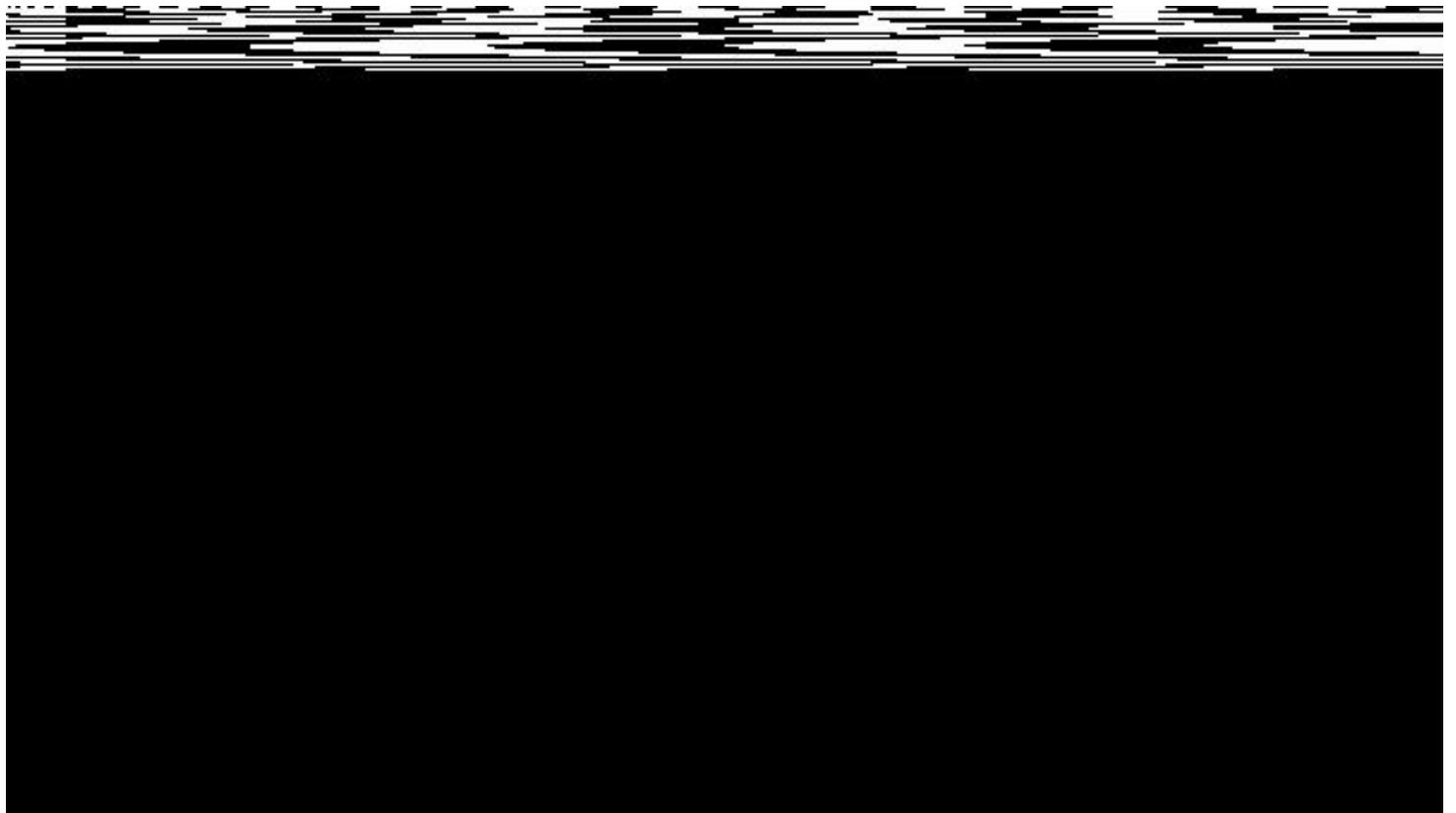

The table illustrates a 2x2 confusion matrix for a binary classification model. The columns represent the "Predicted" state (Positive or Negative) and the rows represent the "Actual" state (Positive or Negative). The matrix elements are labeled as follows:

- True Positive (TP): The number of actual positives correctly identified as positive.
- False Positive (FP): The number of actual negatives incorrectly identified as positive.
- True Negative (TN): The number of actual negatives correctly identified as negative.
- False Negative (FN): The number of actual positives incorrectly identified as negative.

$$TPR = \frac{TP}{TP + FP}$$
$$FPR = \frac{FP}{FP + TN}$$

# Receiver Operating characteristic curve

## ROC curve



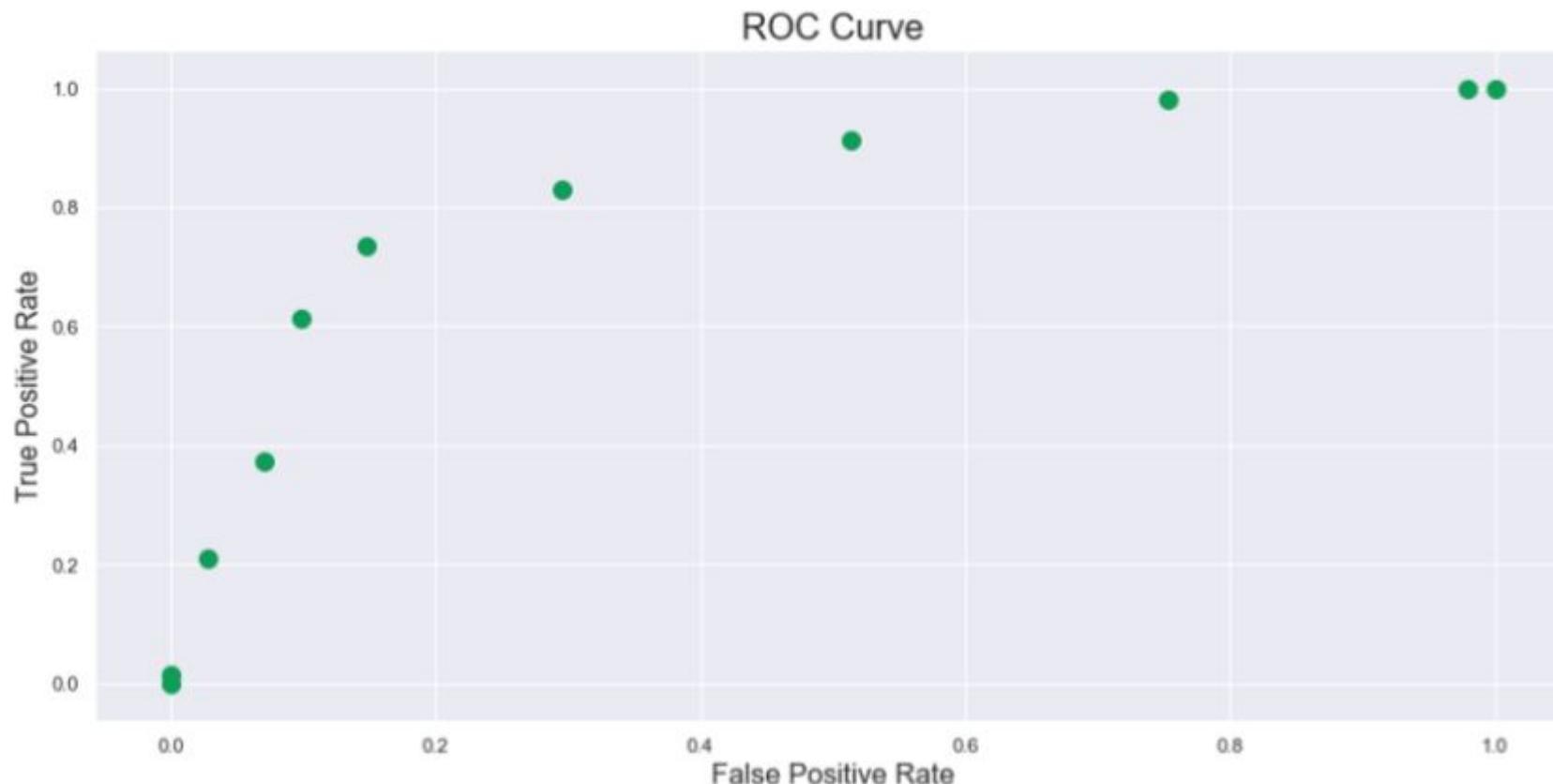
# Receiver Operating characteristic curve

## ROC curve



# Receiver Operating characteristic curve

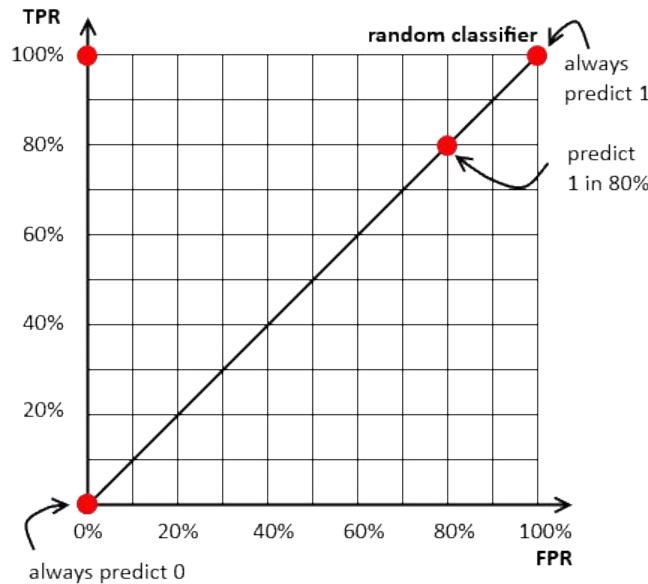
## ROC curve



# Receiver Operating characteristic curve

## ROC curve

- Shows the performance of a classifier at different classification thresholds



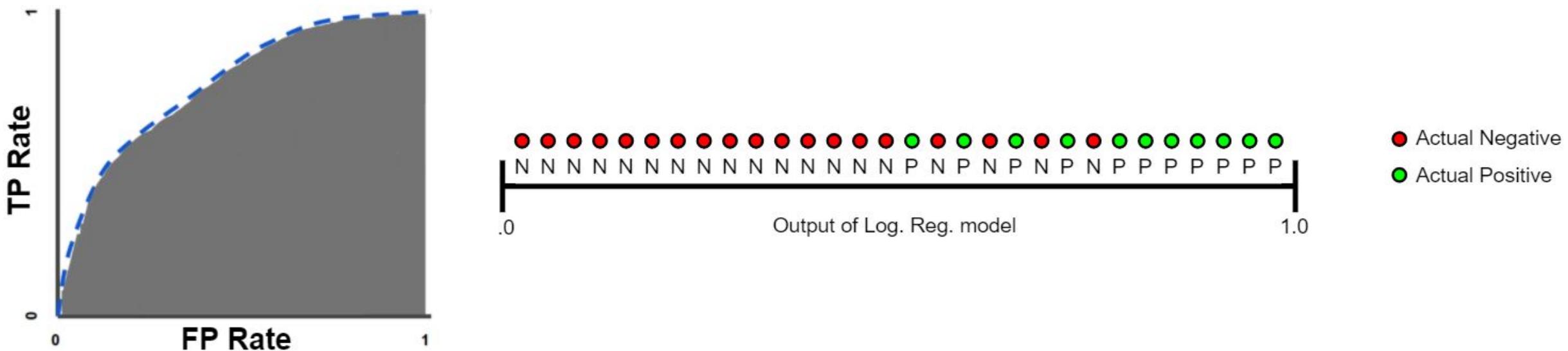
$$TPR = \frac{TP}{TP + FP}$$

$$FPR = \frac{FP}{FP + TN}$$

# Area under the curve

## AUC

- Measures the entire two-dimensional area underneath the entire ROC curve
- Makes it easier to compare one ROC curve to another one



# Recommended resources

- Loss functions (Andrew Ng's great explanations):
  - <https://www.youtube.com/watch?v=LLux1SW--oM>
  - [https://www.youtube.com/watch?v=ueO\\_Ph0Pyqk](https://www.youtube.com/watch?v=ueO_Ph0Pyqk)
- [www.coursera.org](http://www.coursera.org) – Deep Learning specialization: Neural Networks and Deep Learning (week 1 and 2)
- <https://cs231n.github.io/linear-classify/#intro>

# Computer Vision and Deep Learning

Lecture 3

# Last time – linear classification

**Score function:** to map inputs (images) to class scores

**Loss function:** to evaluate a classifier

- Hinge loss
- Softmax loss

**Optimization:** gradient descent

- Numerical gradient
- Analytical gradient

**Classifier evaluation**

# Today's agenda

- Evaluating a model's performance
- Computational graphs
- Backpropagation
- Artificial neural networks
  - Activation functions
- Image convolutions (one step forward towards convolutional neural networks)

# Recap

## Linear classifier

- Linear classifiers are defined by *parameters* (consisting of weights and biases)
- Their training happens in two steps:
  - **Forward Propagation:** It runs the input data through the score function to make its best guess about the correct output.
  - **Backward Propagation:** adjusts its parameters proportionate to the error in its guess.
    - Traverse backwards from the output, collecting the derivatives of the error with respect to the parameters of the functions (*gradients*), and optimizing the parameters using gradient descent.

# How do we compute the gradients?

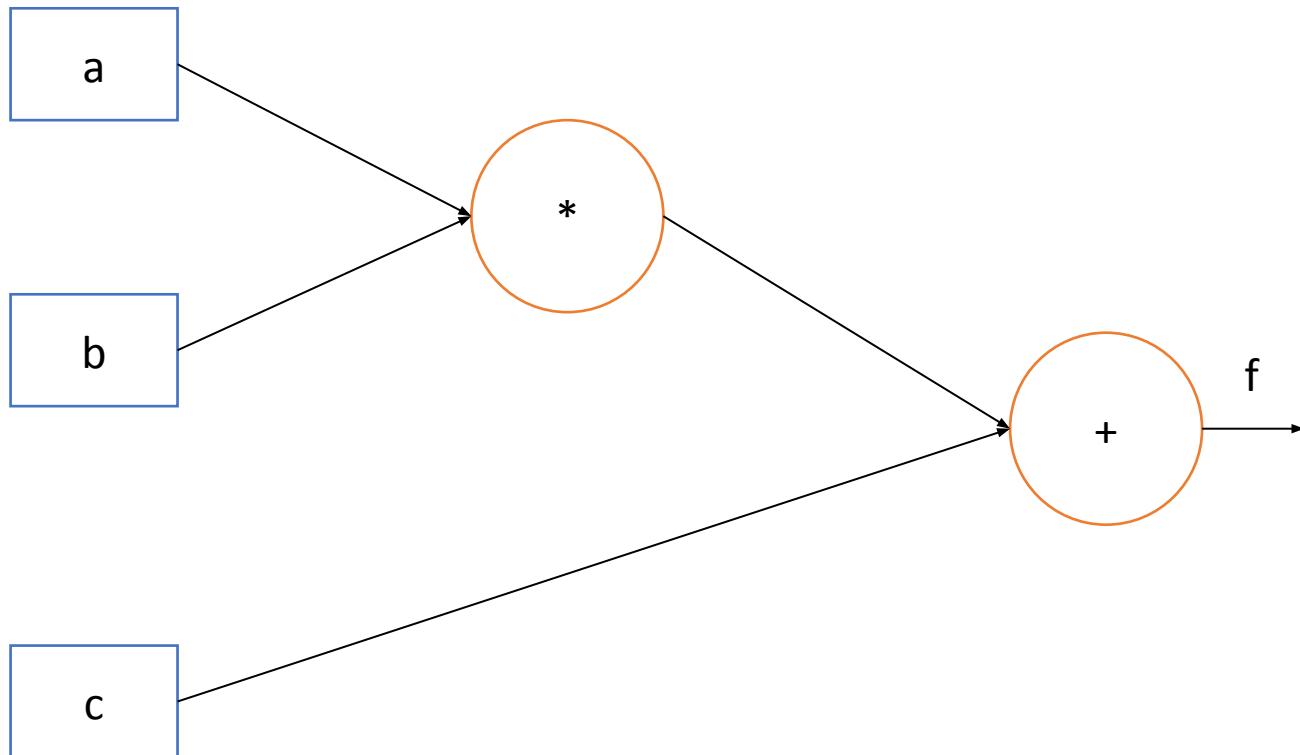
- Numerical gradient
- Analytical gradient
- Better idea: use computational graphs and back-propagation



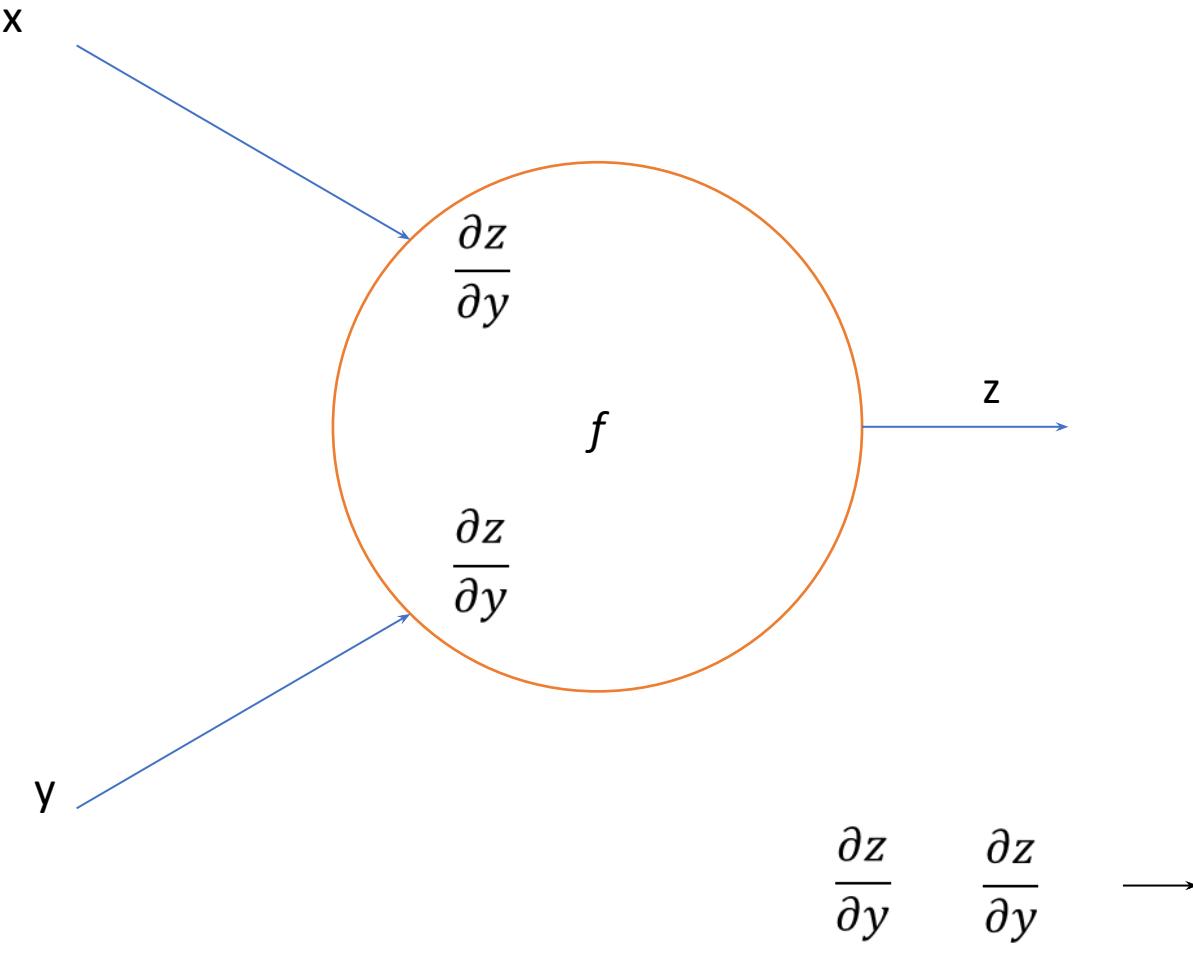
$$f(x) = \frac{(1 + 2x^2)(x^3 - x)^2}{\sqrt{x + 5x^2}(4x)^{3/2}} + \frac{\sqrt{1 + 2x}}{x + \sqrt{1 + x^2}}$$

# Computational graphs

$$f(a, b, c) = a * b + c$$



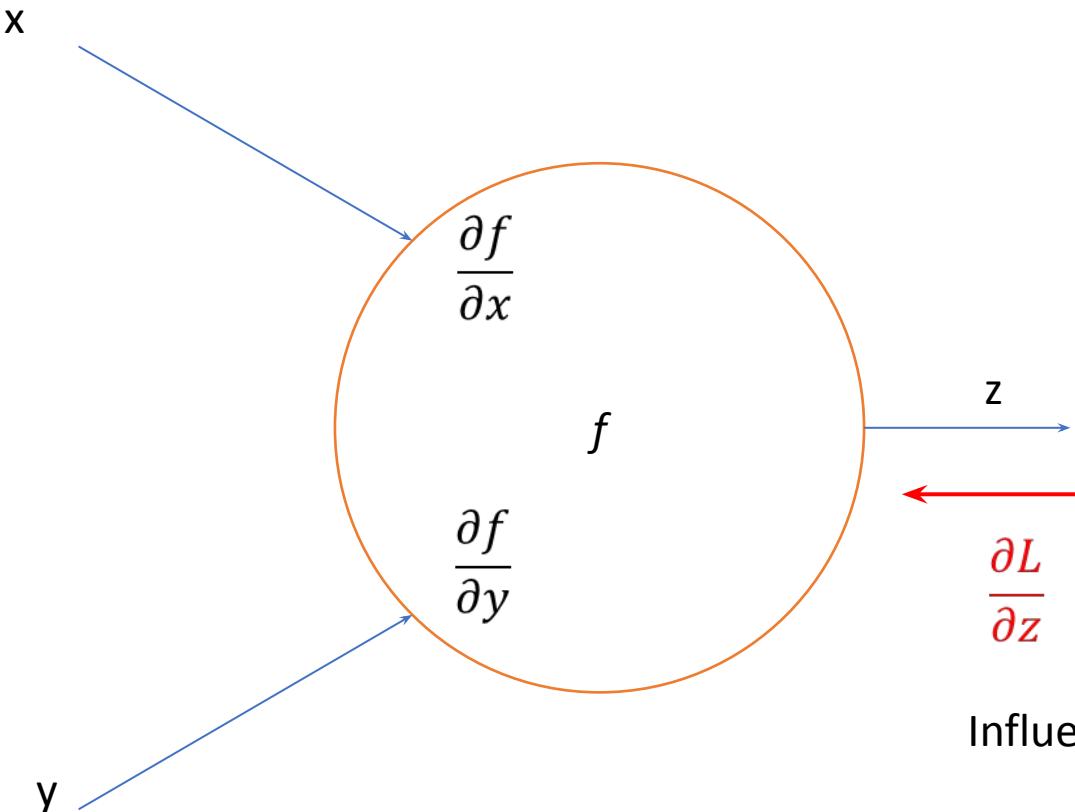
# Back propagation



Local gradients

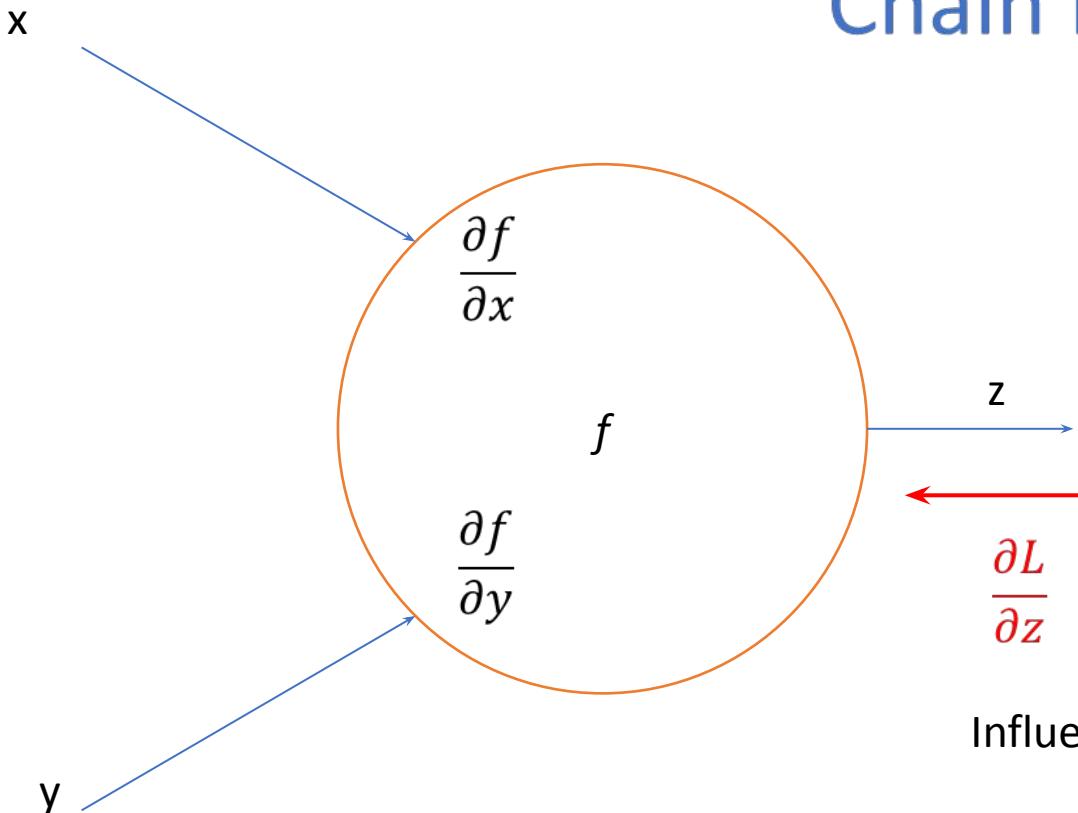
- Influence of  $x$  and  $y$  on the nodes output  $z$

# Back propagation



Influence of  $z$  to the final output of the graph (loss)

# Back propagation

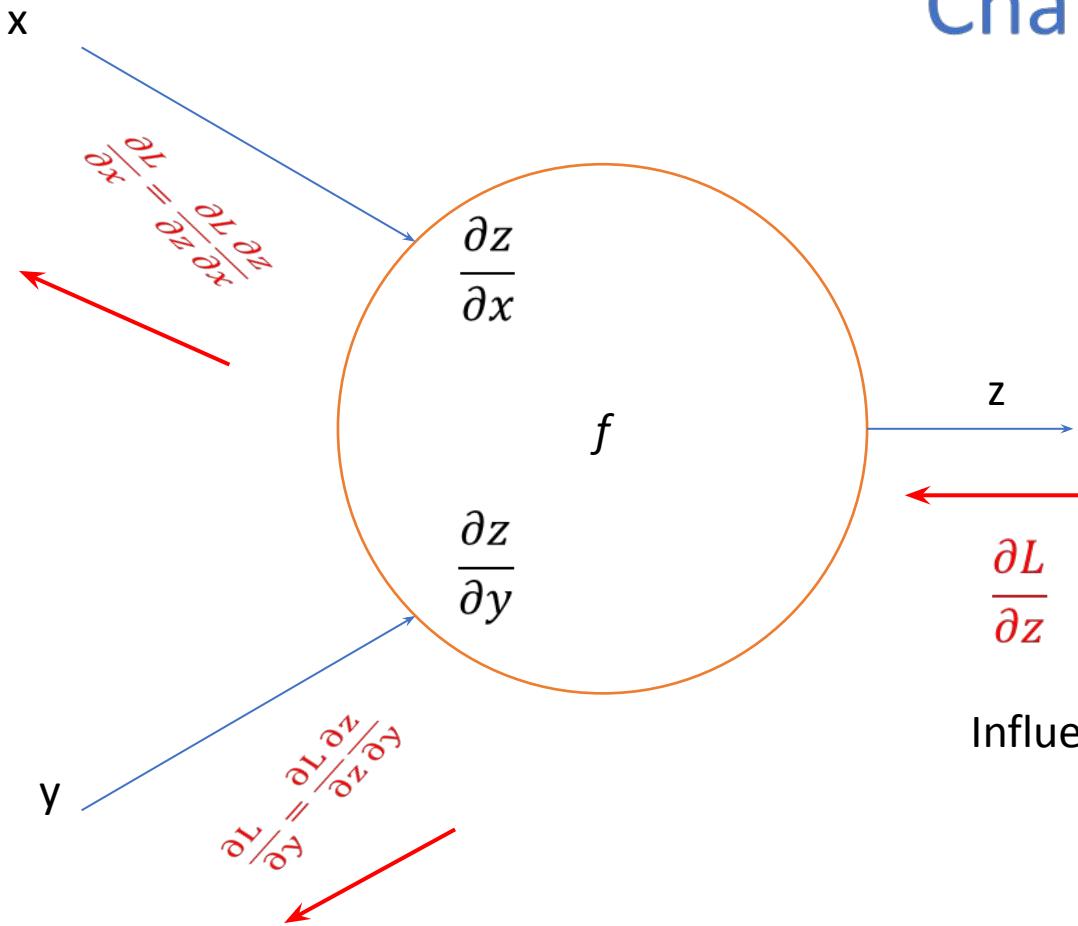


Chain rule  $\frac{\partial f}{\partial a} = \frac{\partial f}{\partial b} \frac{\partial b}{\partial z}$



Influence of  $z$  to the final output of the graph (loss)

# Back propagation



Chain rule  $\frac{\partial f}{\partial a} = \frac{\partial f}{\partial b} \frac{\partial b}{\partial a}$



Influence of  $z$  to the final output of the graph (loss)

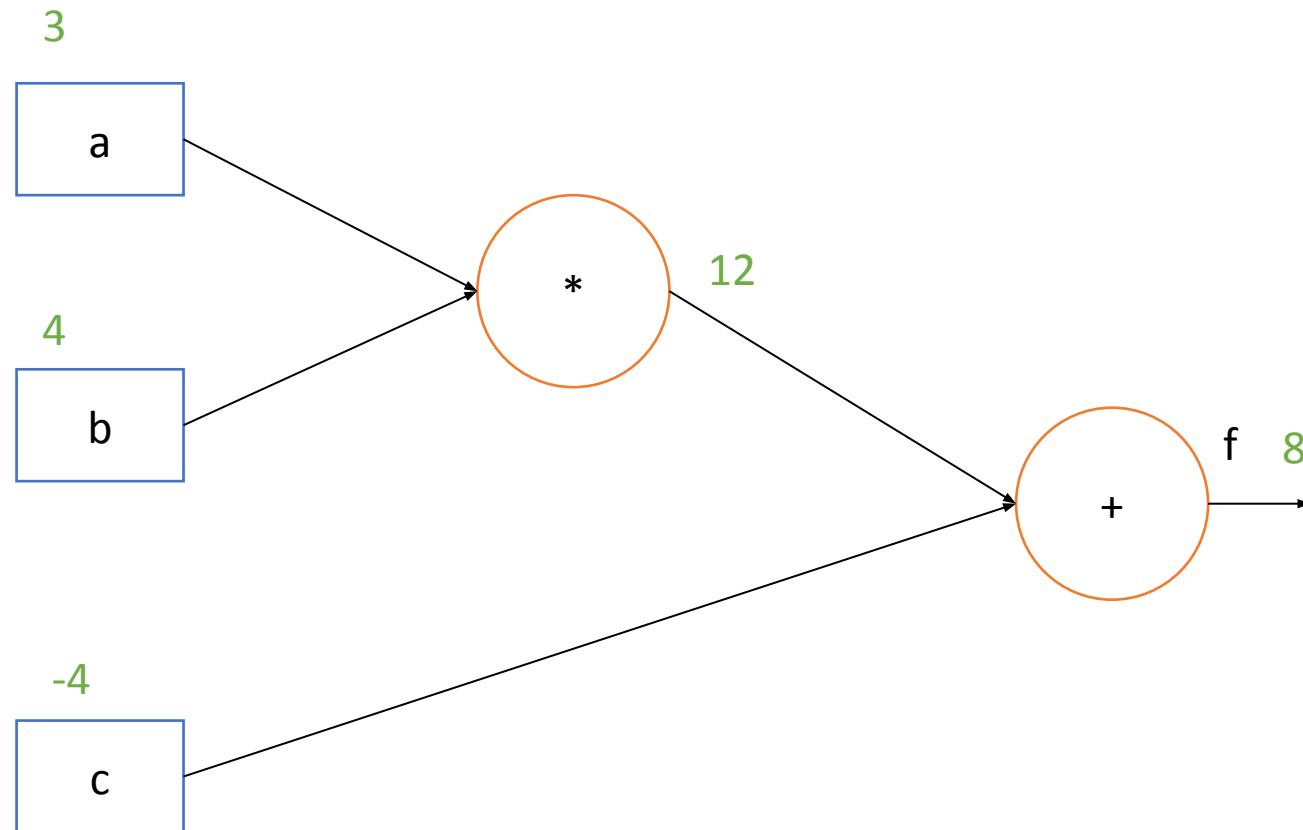
# Back propagation

Use a recursive application of the chain rule on every node in the graph to compute influence of all the intermediate nodes on the output of the graph

# Computational graphs – forward pass

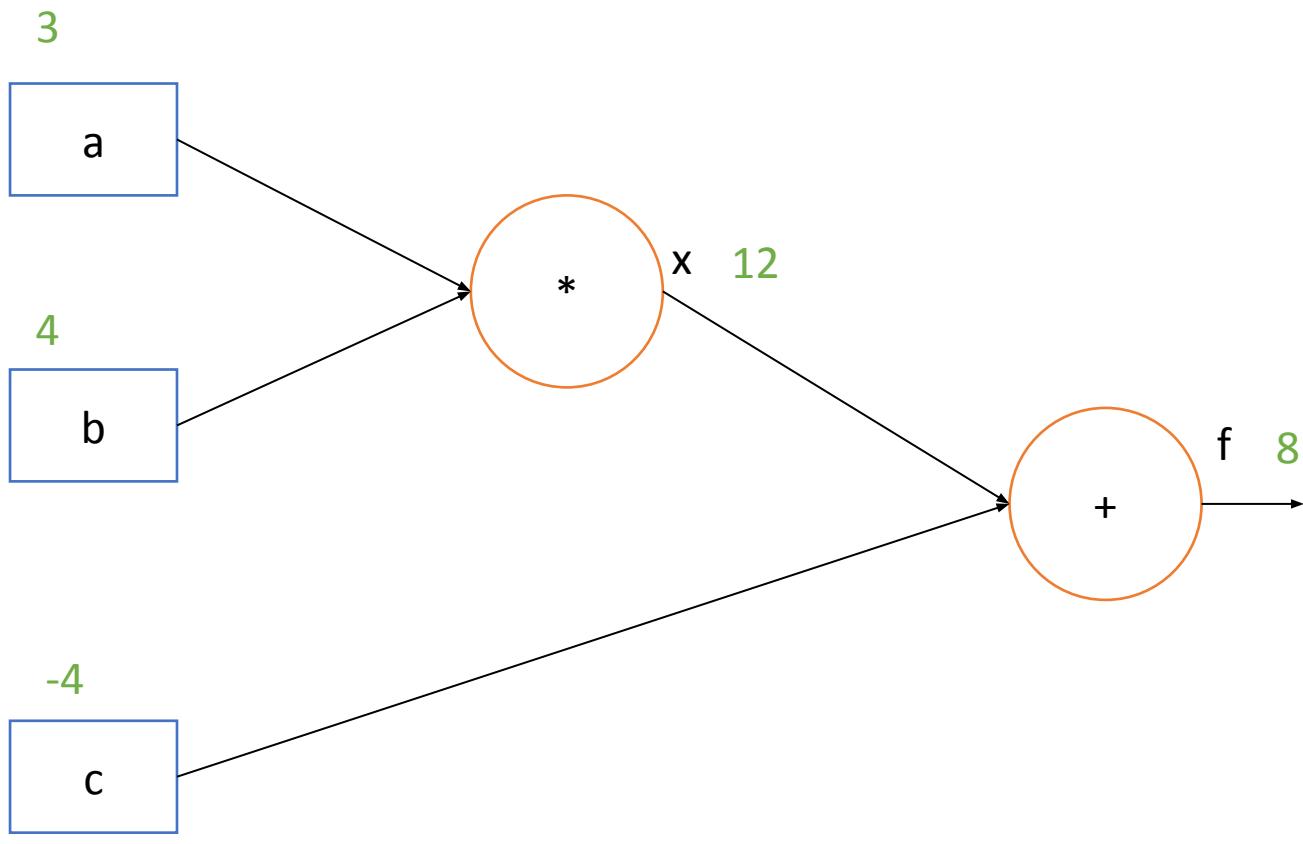
$$f(a, b, c) = a * b + c$$

$a = 3$   
 $b = 2$   
 $c = -4$



# Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$$f(a, b, c) = a \cdot b + c$$

$$\begin{aligned} x &= a \cdot b & \frac{\partial x}{\partial a} &= b \\ && \frac{\partial x}{\partial b} &= a \end{aligned}$$

$$\begin{aligned} f &= x + c & \frac{\partial f}{\partial c} &= 1 \\ && \frac{\partial f}{\partial x} &= 1 \end{aligned}$$

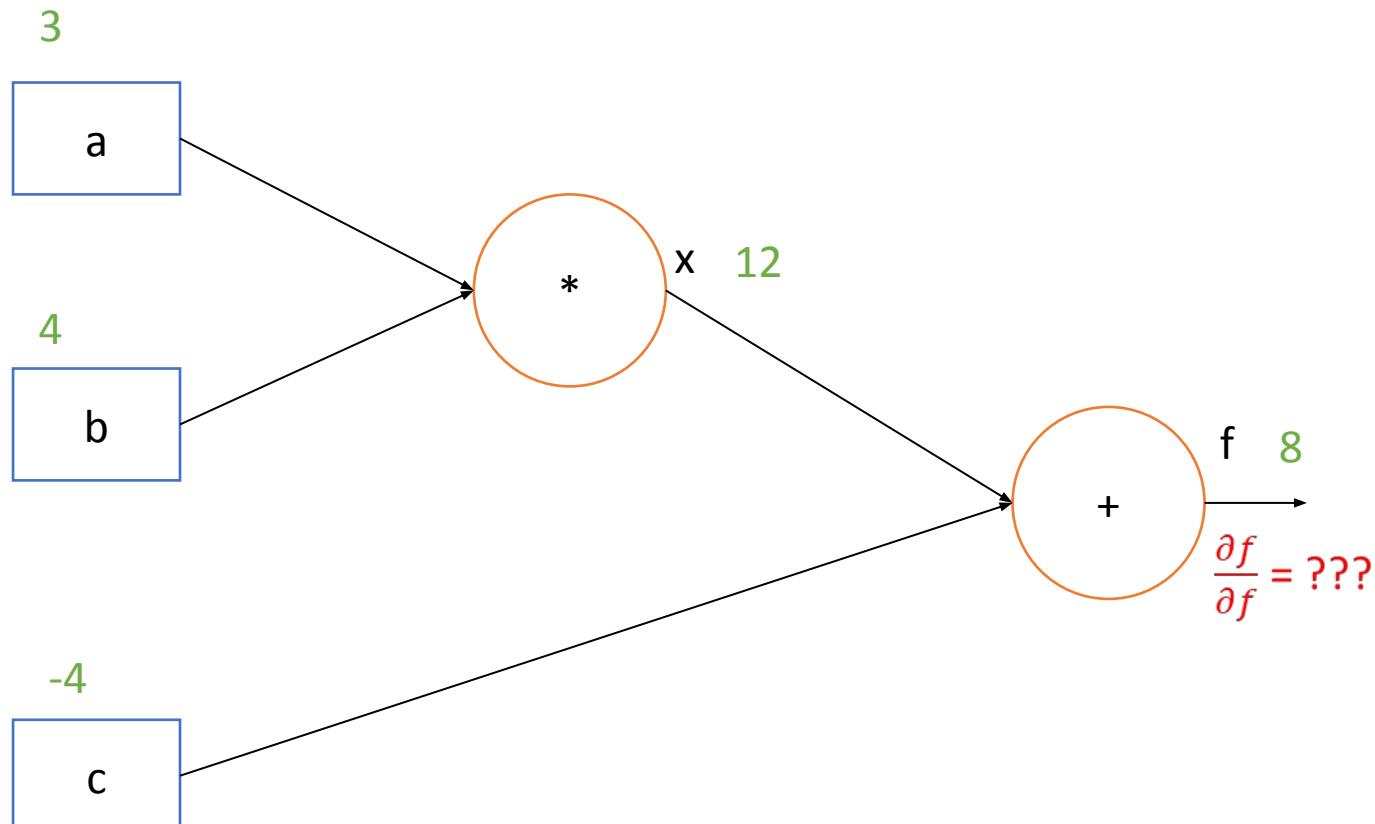
$$\frac{\partial f}{\partial a} = ???$$

$$\frac{\partial f}{\partial b} = ???$$

$$\frac{\partial f}{\partial c} = ???$$

# Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$$f(a, b, c) = a \cdot b + c$$

$$\begin{aligned} x &= a \cdot b & \frac{\partial x}{\partial a} &= b \\ & & \frac{\partial x}{\partial b} &= a \end{aligned}$$

$$\begin{aligned} f &= x + c & \frac{\partial f}{\partial c} &= 1 \\ & & \frac{\partial f}{\partial x} &= 1 \end{aligned}$$

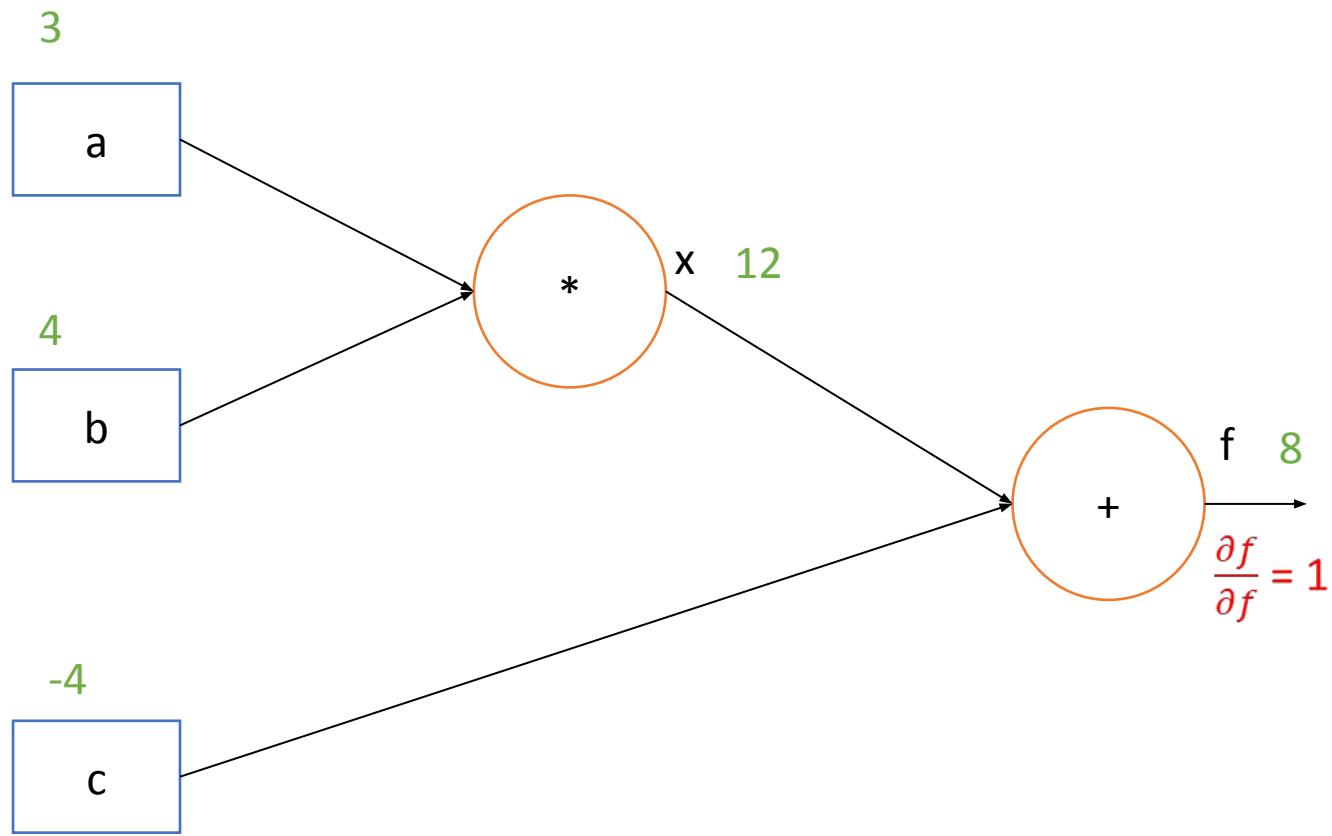
$$\frac{\partial f}{\partial a} = ???$$

$$\frac{\partial f}{\partial b} = ???$$

$$\frac{\partial f}{\partial c} = ???$$

# Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$$f(a, b, c) = a \cdot b + c$$

$$\begin{aligned}x &= a \cdot b \\ \frac{\partial x}{\partial a} &= b \\ \frac{\partial x}{\partial b} &= a\end{aligned}$$

$$\begin{aligned}f &= x + c \\ \frac{\partial f}{\partial c} &= 1 \\ \frac{\partial f}{\partial x} &= 1\end{aligned}$$

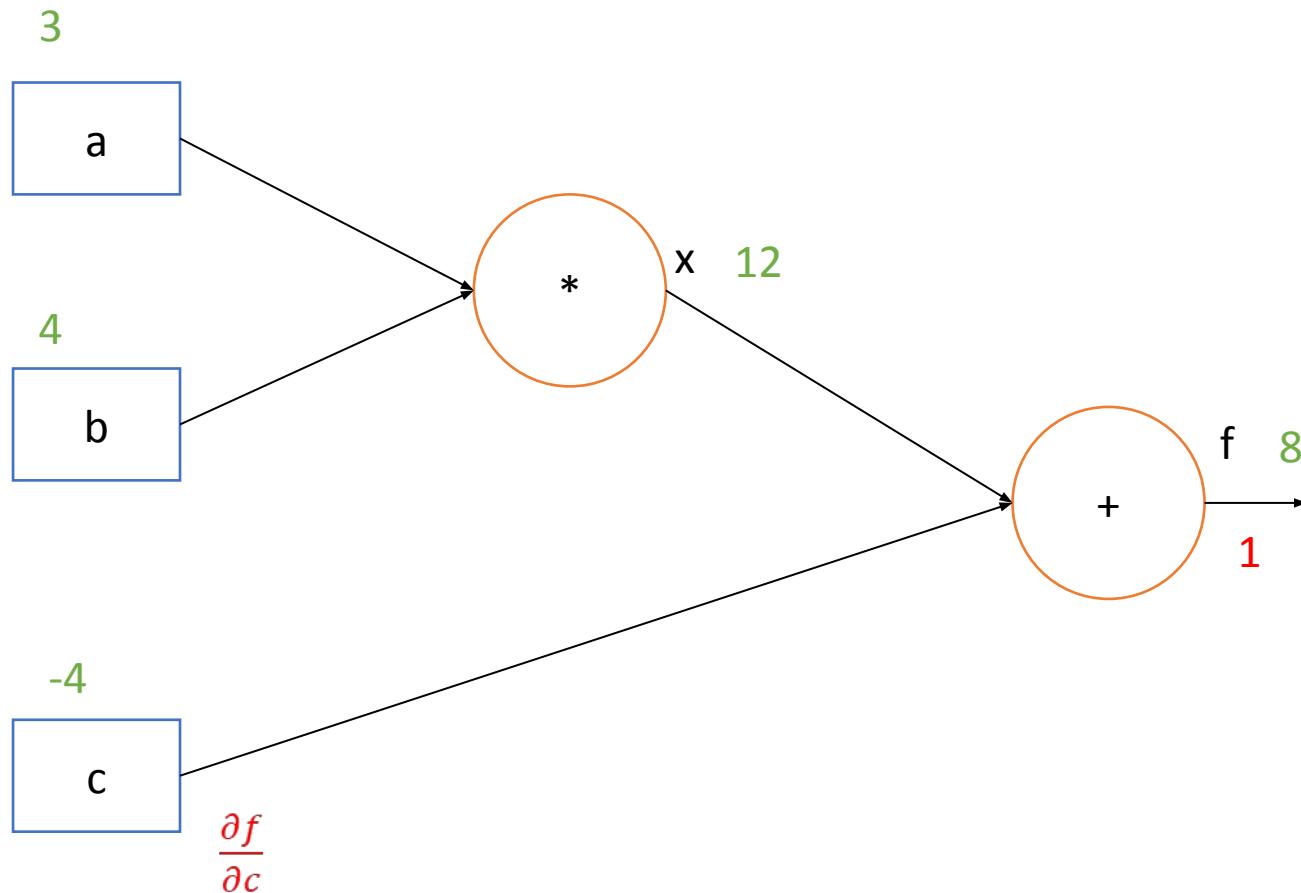
$$\frac{\partial f}{\partial a} = ???$$

$$\frac{\partial f}{\partial b} = ???$$

$$\frac{\partial f}{\partial c} = ???$$

# Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$$f(a, b, c) = a * b + c$$

$$\begin{aligned}x &= a * b \\ \frac{\partial x}{\partial a} &= b \\ \frac{\partial x}{\partial b} &= a\end{aligned}$$

$$\begin{aligned}f &= x + c \\ \frac{\partial f}{\partial c} &= 1 \\ \frac{\partial f}{\partial x} &= 1\end{aligned}$$

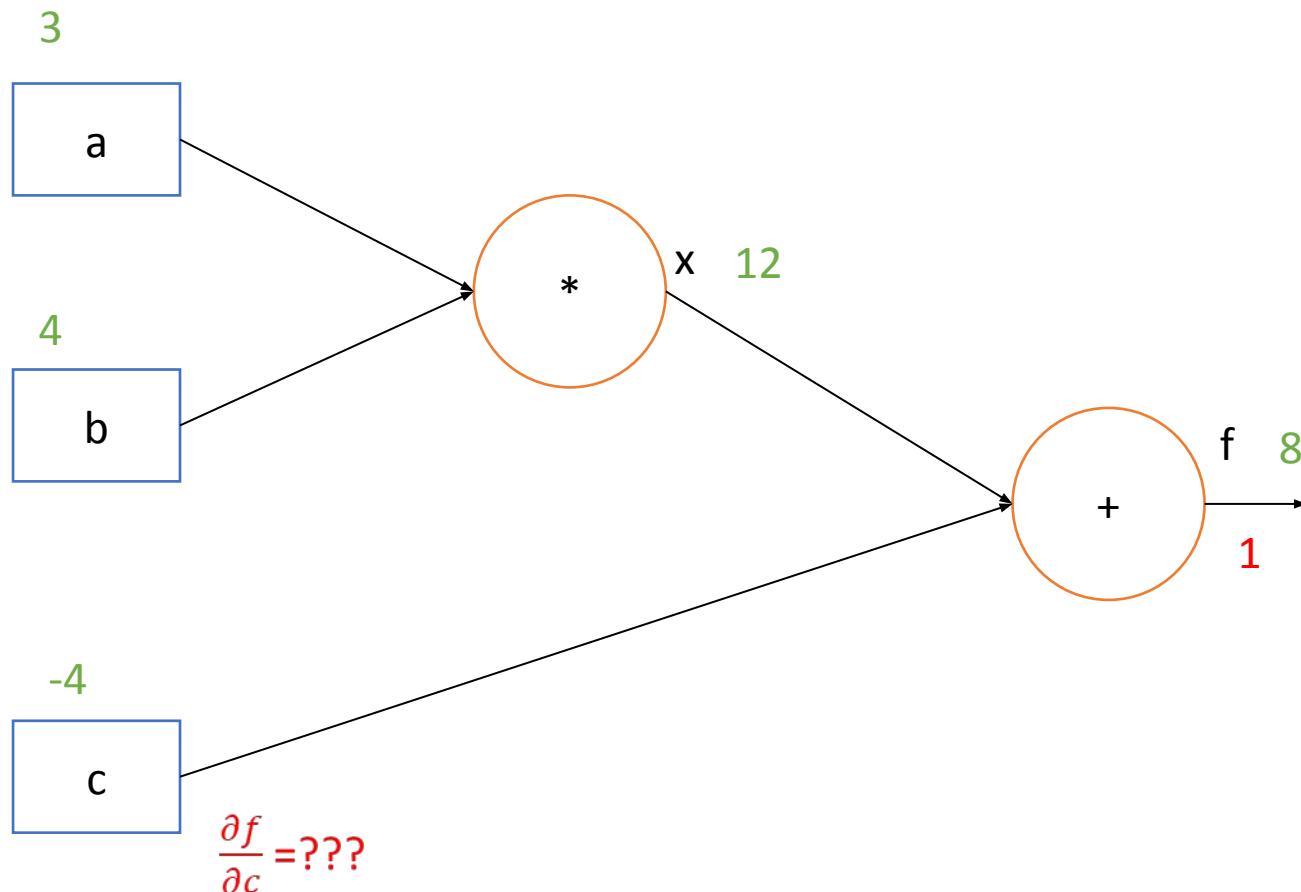
$$\frac{\partial f}{\partial a} = ???$$

$$\frac{\partial f}{\partial b} = ???$$

$$\frac{\partial f}{\partial c} = ???$$

# Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$$f(a, b, c) = a \cdot b + c$$

$$\begin{aligned} x &= a \cdot b & \frac{\partial x}{\partial a} &= b \\ && \frac{\partial x}{\partial b} &= a \end{aligned}$$

$$\begin{aligned} f &= x + c & \frac{\partial f}{\partial c} &= 1 \\ && \frac{\partial f}{\partial x} &= 1 \end{aligned}$$

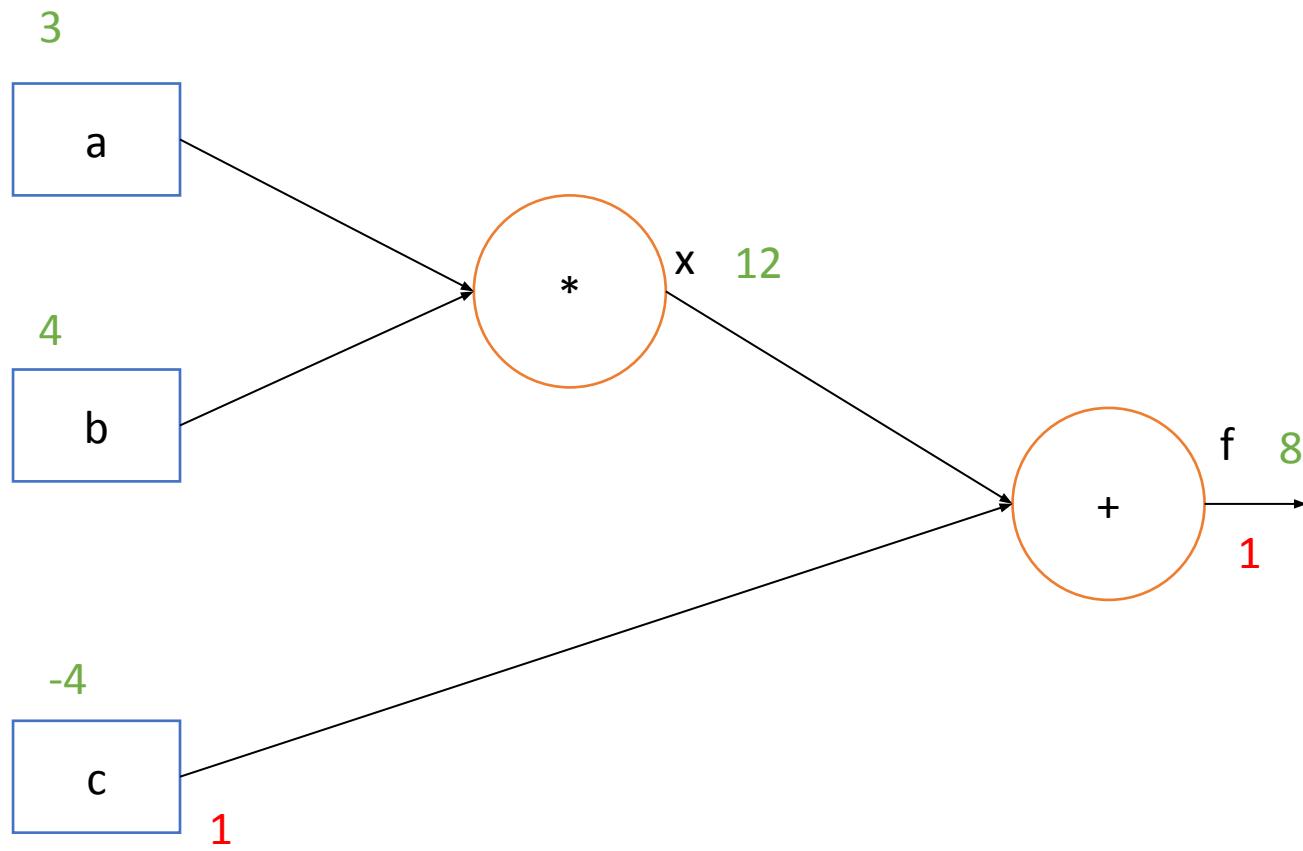
$$\frac{\partial f}{\partial a} = ???$$

$$\frac{\partial f}{\partial b} = ???$$

$$\frac{\partial f}{\partial v} = ???$$

# Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$$f(a, b, c) = a * b + c$$

$$\begin{aligned} x &= a * b & \frac{\partial x}{\partial a} &= b \\ && \frac{\partial x}{\partial b} &= a \end{aligned}$$

$$\begin{aligned} f &= x + c & \frac{\partial f}{\partial c} &= 1 \\ && \frac{\partial f}{\partial x} &= 1 \end{aligned}$$

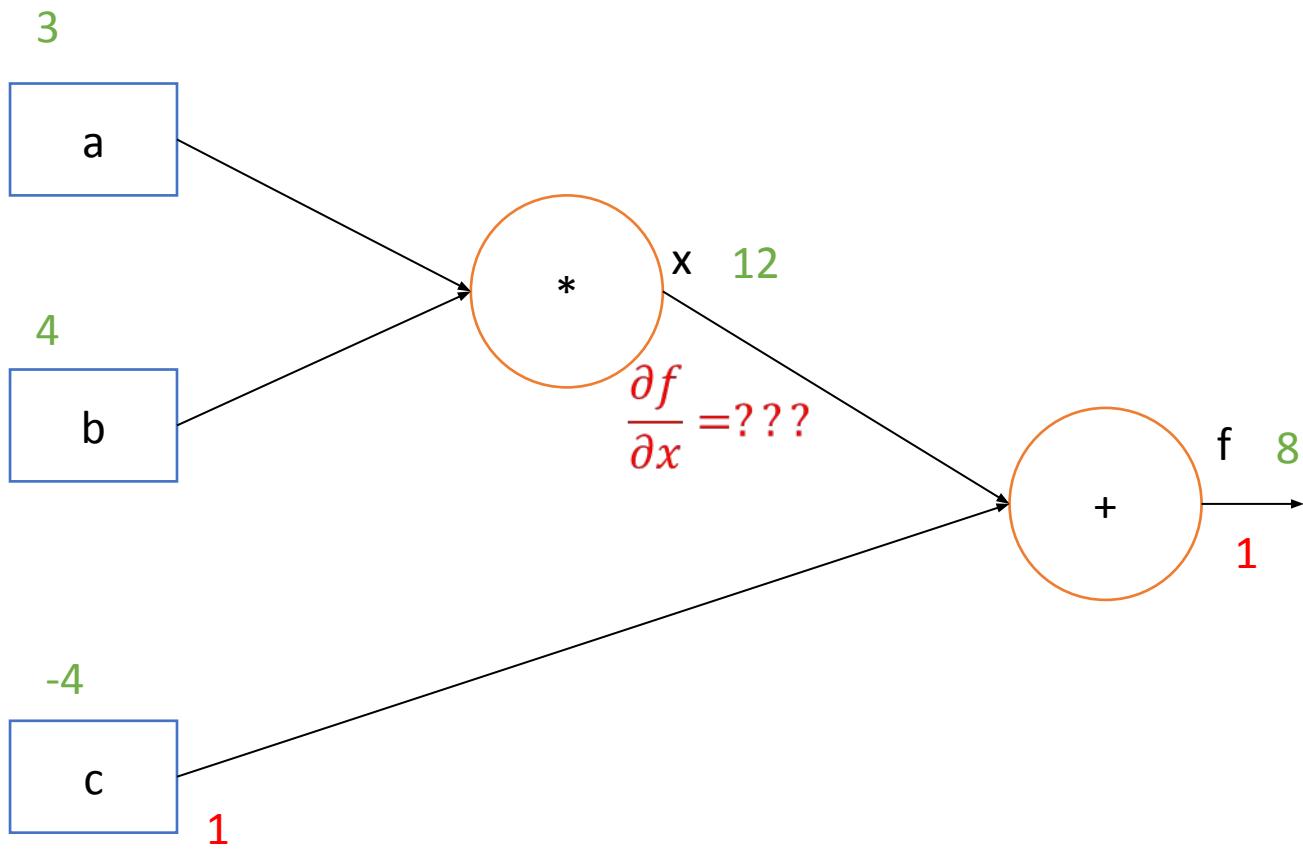
$$\frac{\partial f}{\partial a} = ???$$

$$\frac{\partial f}{\partial b} = ???$$

$$\frac{\partial f}{\partial c} = ???$$

# Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$$f(a, b, c) = a \cdot b + c$$

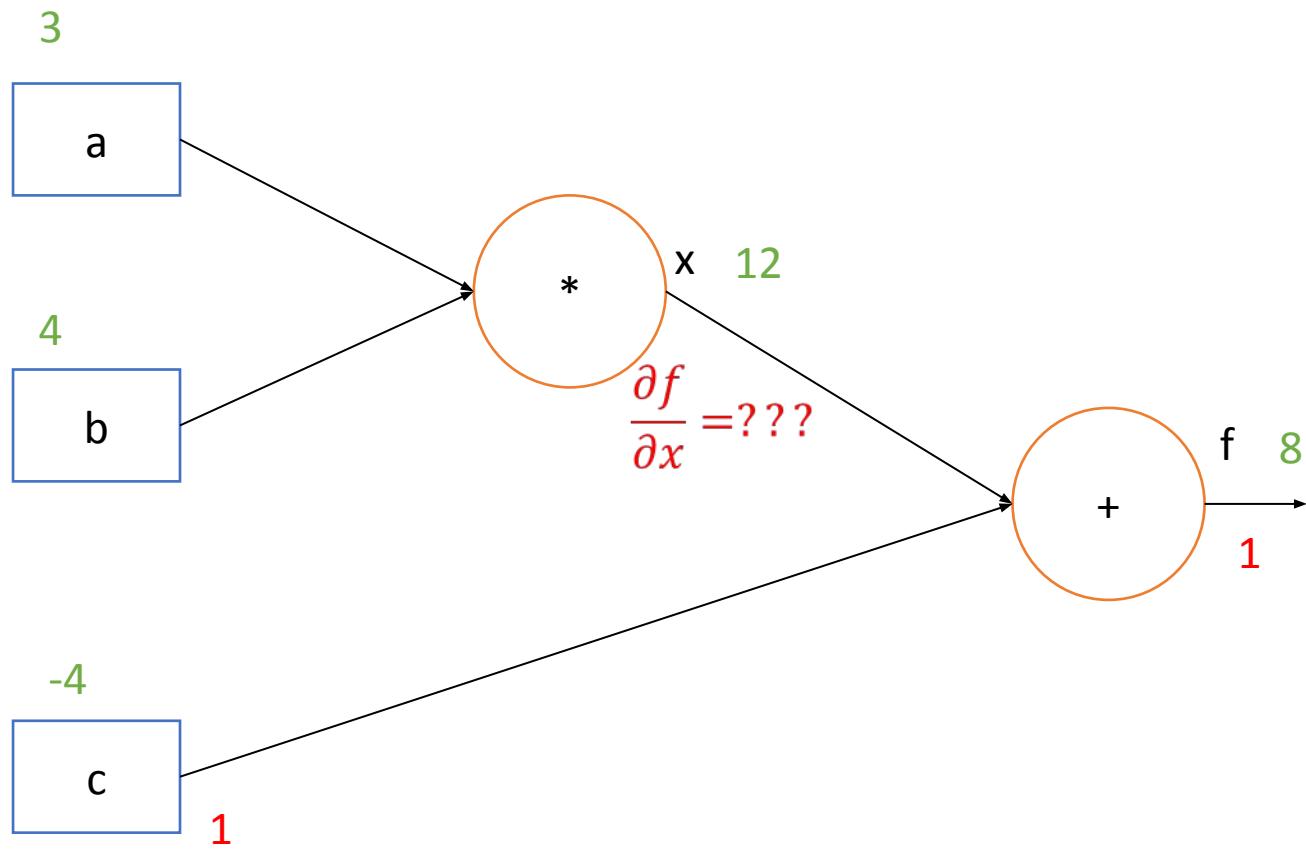
$$\begin{aligned}x &= a \cdot b \\ \frac{\partial x}{\partial a} &= b \\ \frac{\partial x}{\partial b} &= a\end{aligned}$$

$$\begin{aligned}f &= x + c \\ \frac{\partial f}{\partial c} &= 1 \\ \frac{\partial f}{\partial x} &= 1\end{aligned}$$

$$\begin{aligned}\frac{\partial f}{\partial a} &= ??? & \frac{\partial f}{\partial b} &= ??? & \frac{\partial f}{\partial c} &= ???\end{aligned}$$

# Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$$f(a, b, c) = a \cdot b + c$$

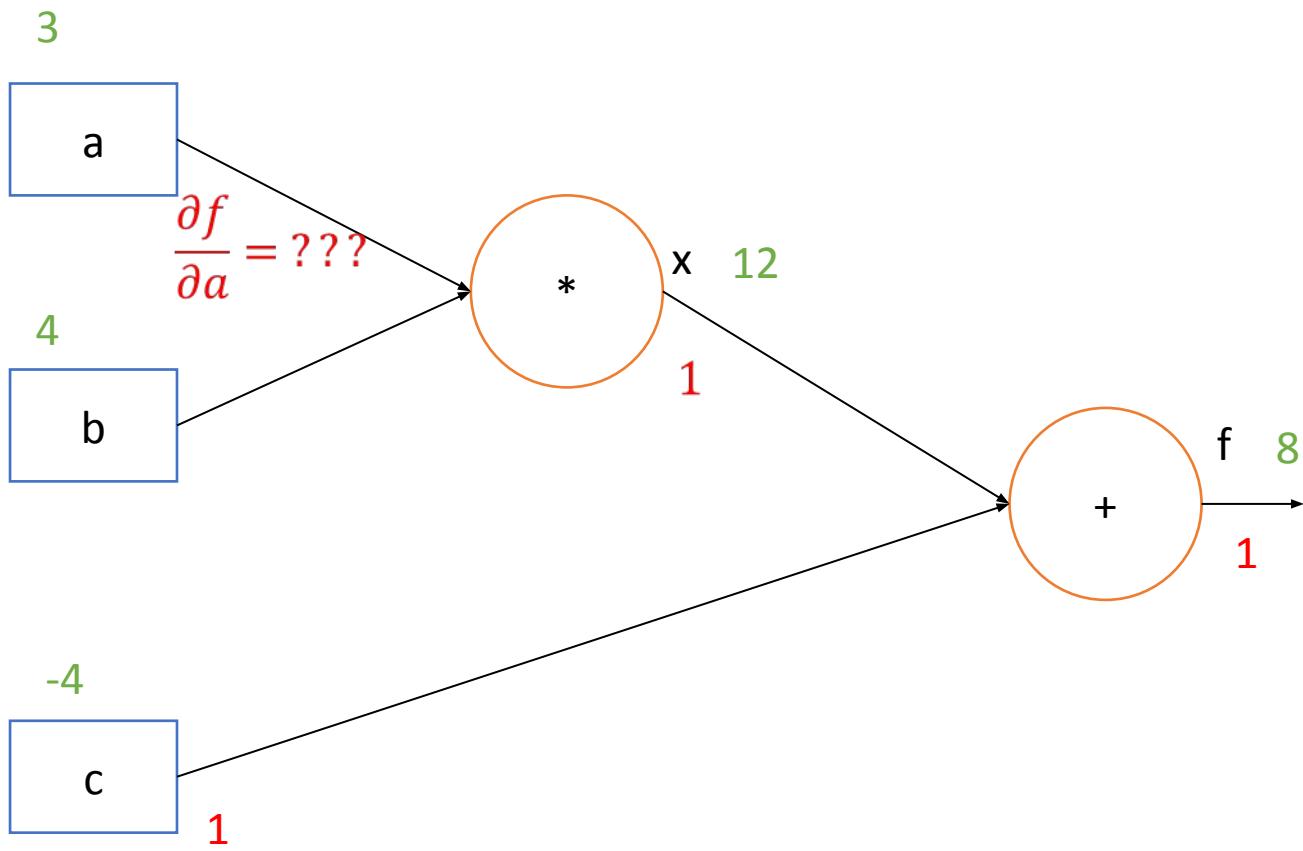
$$\begin{aligned}x &= a \cdot b \\ \frac{\partial x}{\partial a} &= b \\ \frac{\partial x}{\partial b} &= a\end{aligned}$$

$$\begin{aligned}f &= x + c \\ \frac{\partial f}{\partial c} &= 1 \\ \frac{\partial f}{\partial x} &= 1\end{aligned}$$

$$\begin{aligned}\frac{\partial f}{\partial a} &= ??? & \frac{\partial f}{\partial b} &= ??? & \frac{\partial f}{\partial c} &= ???\end{aligned}$$

# Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$$f(a, b, c) = a \cdot b + c$$

$$\begin{aligned} x &= a \cdot b & \frac{\partial x}{\partial a} &= b \\ & & \frac{\partial x}{\partial b} &= a \end{aligned}$$

$$\begin{aligned} f &= x + c & \frac{\partial f}{\partial c} &= 1 \\ & & \frac{\partial f}{\partial x} &= 1 \end{aligned}$$

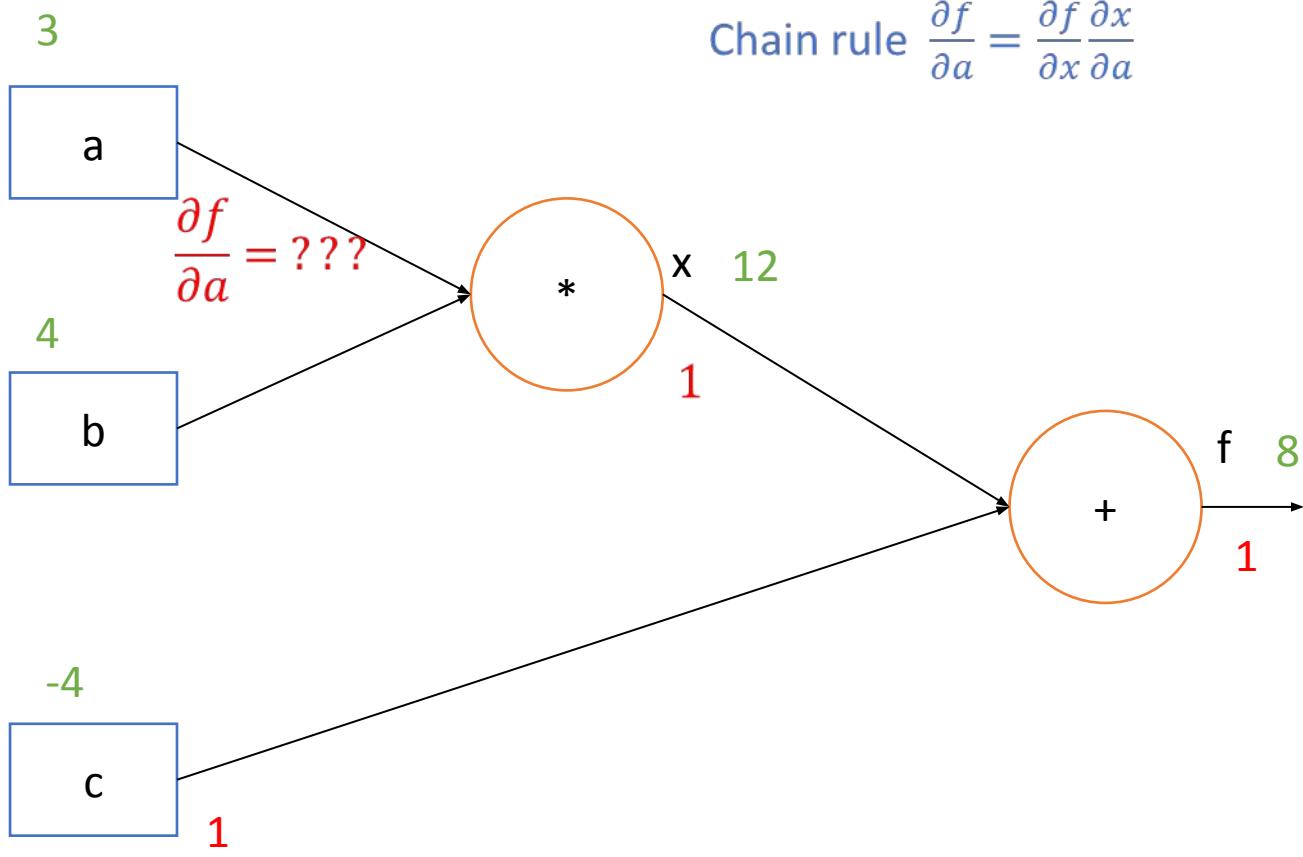
$$\frac{\partial f}{\partial a} = ???$$

$$\frac{\partial f}{\partial b} = ???$$

$$\frac{\partial f}{\partial c} = ???$$

# Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$$\text{Chain rule } \frac{\partial f}{\partial a} = \frac{\partial f}{\partial x} \frac{\partial x}{\partial a}$$

$$f(a, b, c) = a * b + c$$

$$x = a * b$$
$$\frac{\partial x}{\partial a} = b$$
$$\frac{\partial x}{\partial b} = a$$

$$f = x + c$$
$$\frac{\partial f}{\partial c} = 1$$
$$\frac{\partial f}{\partial x} = 1$$

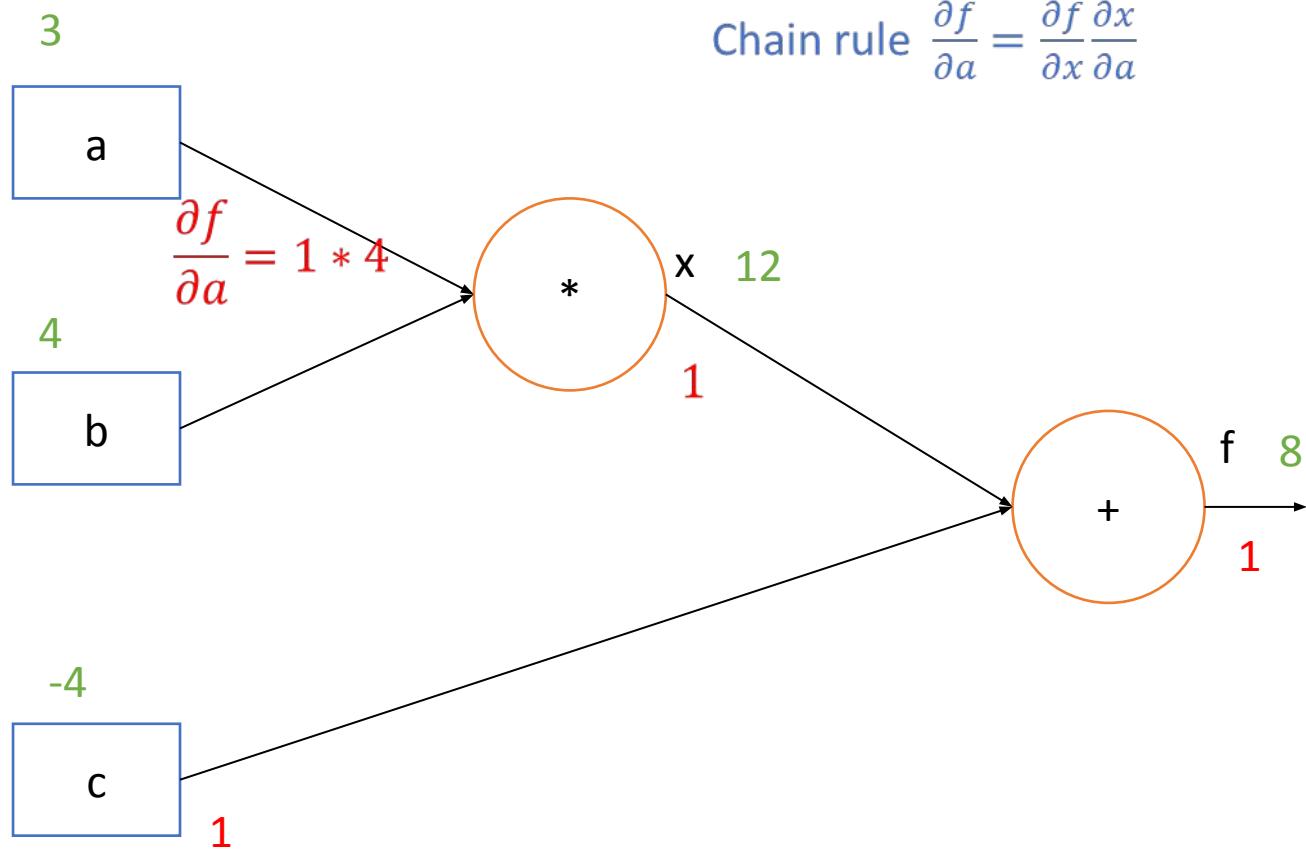
$$\frac{\partial f}{\partial a} = ???$$

$$\frac{\partial f}{\partial b} = ???$$

$$\frac{\partial f}{\partial c} = ???$$

# Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$$\text{Chain rule } \frac{\partial f}{\partial a} = \frac{\partial f}{\partial x} \frac{\partial x}{\partial a}$$

$$f(a, b, c) = a * b + c$$

$$x = a * b$$
$$\frac{\partial x}{\partial a} = b$$
$$\frac{\partial x}{\partial b} = a$$

$$f = x + c$$
$$\frac{\partial f}{\partial c} = 1$$
$$\frac{\partial f}{\partial x} = 1$$

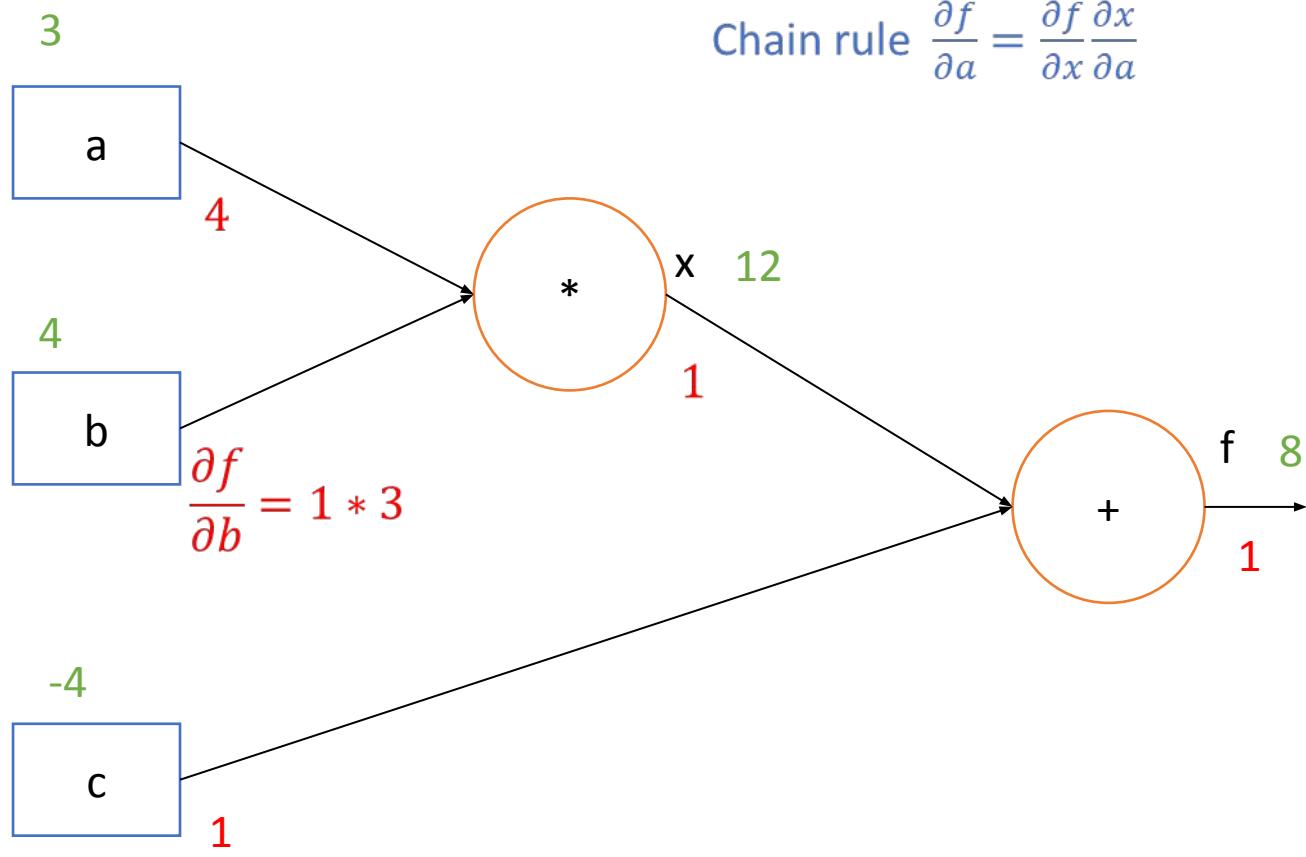
$$\frac{\partial f}{\partial a} = ???$$

$$\frac{\partial f}{\partial b} = ???$$

$$\frac{\partial f}{\partial c} = ???$$

# Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



Chain rule  $\frac{\partial f}{\partial a} = \frac{\partial f}{\partial x} \frac{\partial x}{\partial a}$

$$f(a, b, c) = a * b + c$$

$$\begin{aligned} x &= a * b & \frac{\partial x}{\partial a} &= b \\ & & \frac{\partial x}{\partial b} &= a \end{aligned}$$

$$\begin{aligned} f &= x + c & \frac{\partial f}{\partial c} &= 1 \\ & & \frac{\partial f}{\partial x} &= 1 \end{aligned}$$

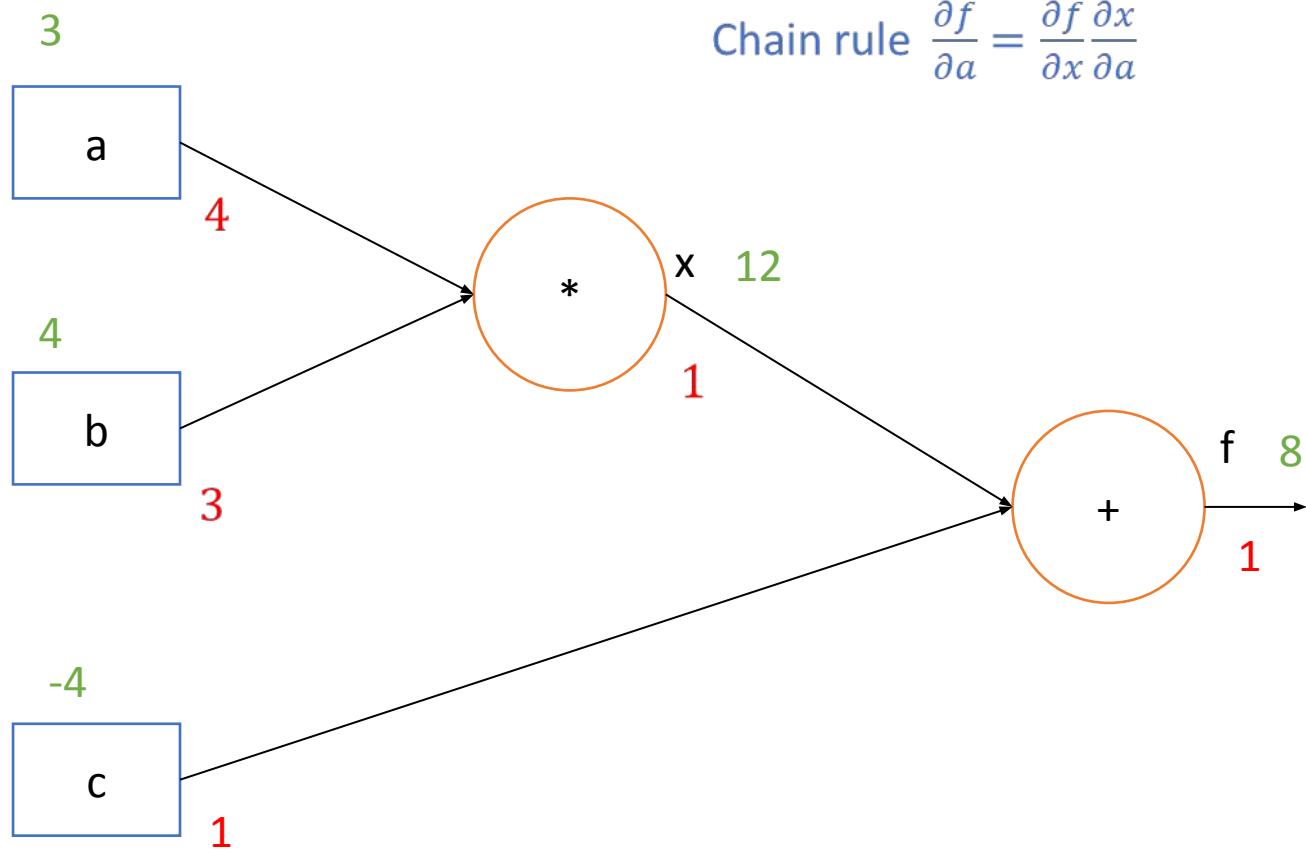
$$\frac{\partial f}{\partial a} = ???$$

$$\frac{\partial f}{\partial b} = ???$$

$$\frac{\partial f}{\partial c} = ???$$

# Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



$$f(a, b, c) = a * b + c$$

$$\begin{aligned} x &= a * b & \frac{\partial x}{\partial a} &= b \\ & & \frac{\partial x}{\partial b} &= a \end{aligned}$$

$$\begin{aligned} f &= x + c & \frac{\partial f}{\partial c} &= 1 \\ & & \frac{\partial f}{\partial x} &= 1 \end{aligned}$$

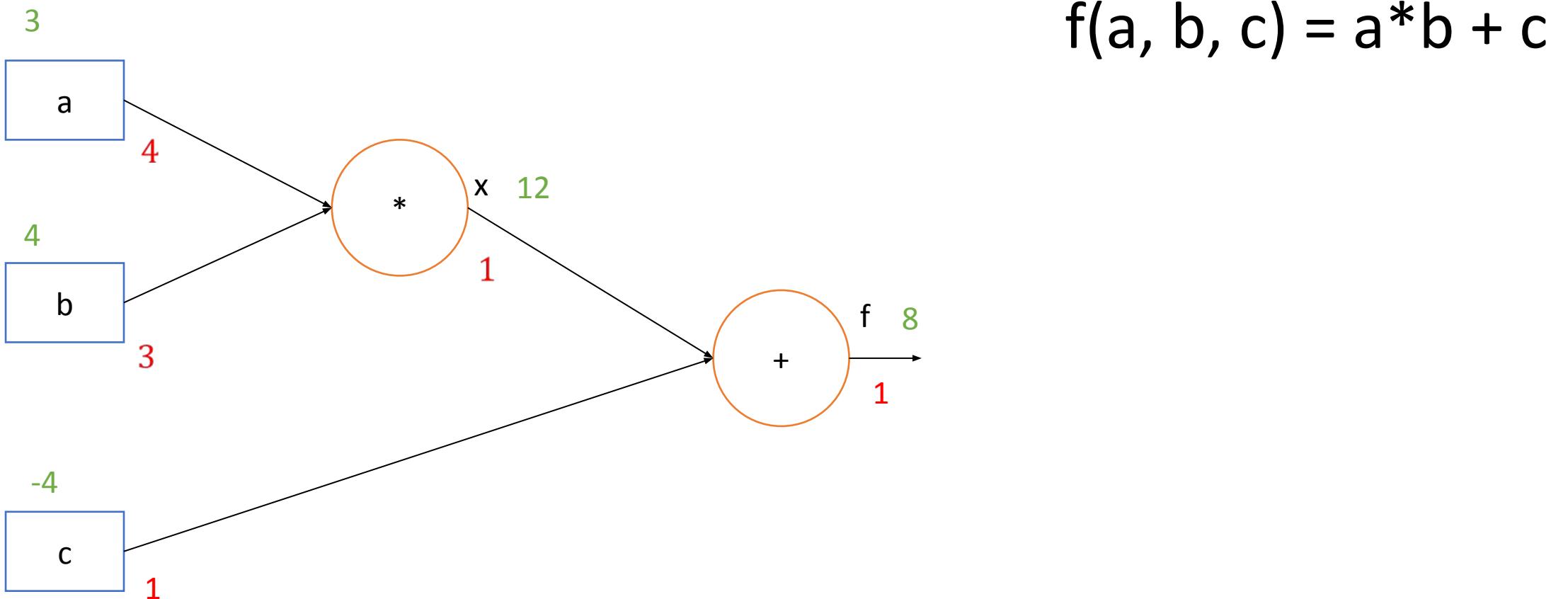
$$\frac{\partial f}{\partial a} = ???$$

$$\frac{\partial f}{\partial b} = ???$$

$$\frac{\partial f}{\partial c} = ???$$

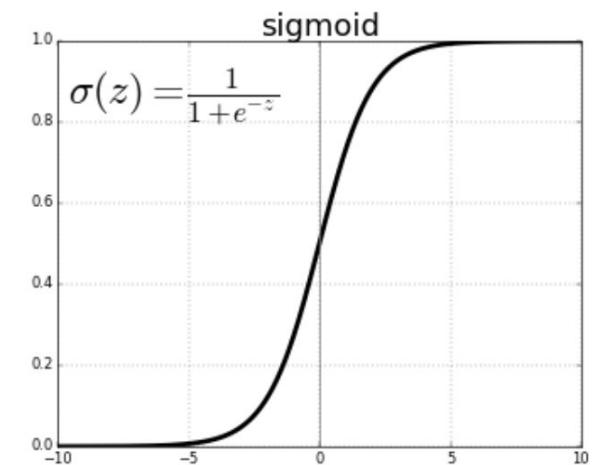
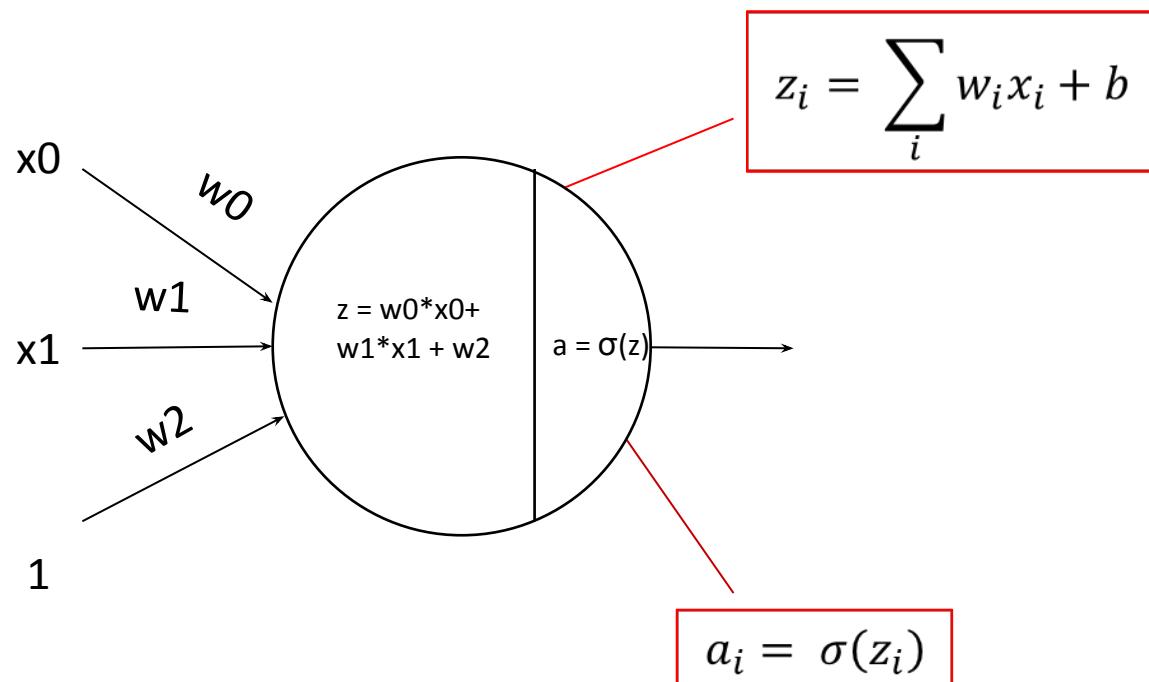
# Computational graphs – backward pass

Go backward, from the end of the computational graph, and compute the gradient for each node in the graph



# Backprop example with a neural network in mind

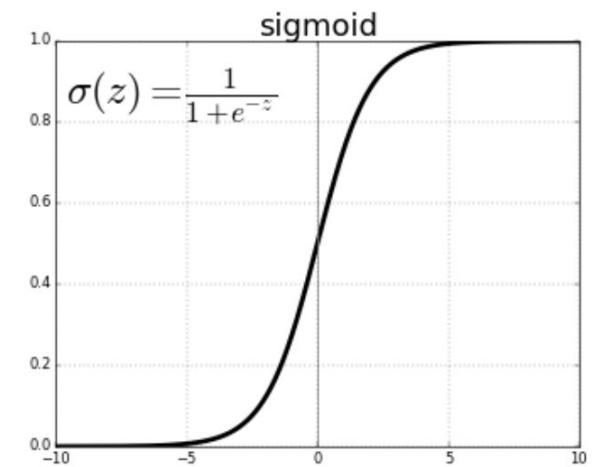
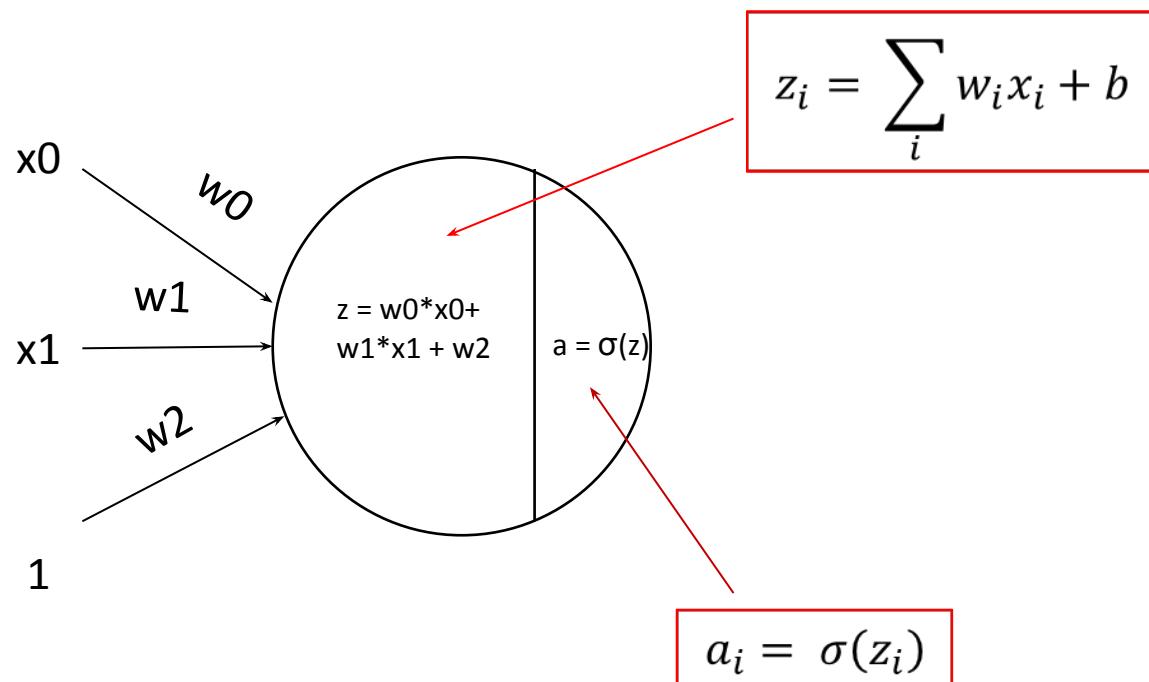
In a neural network a neuron computes a linear function, followed by a non-linearity (activation function).



# Backprop example

## With a neural network in mind

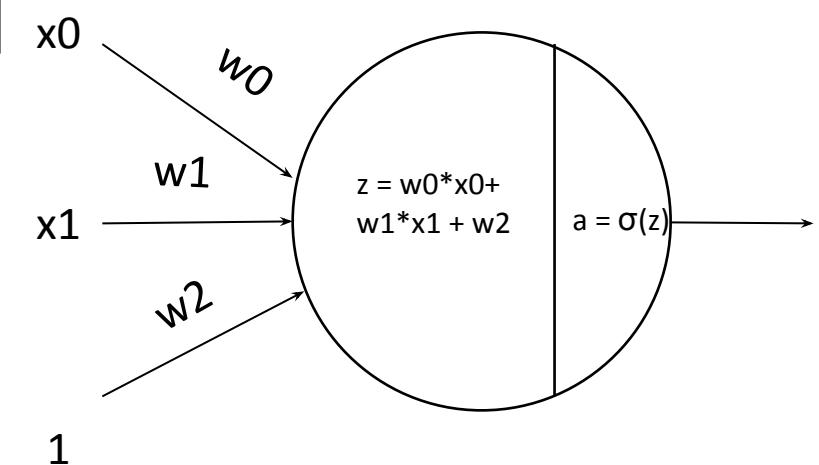
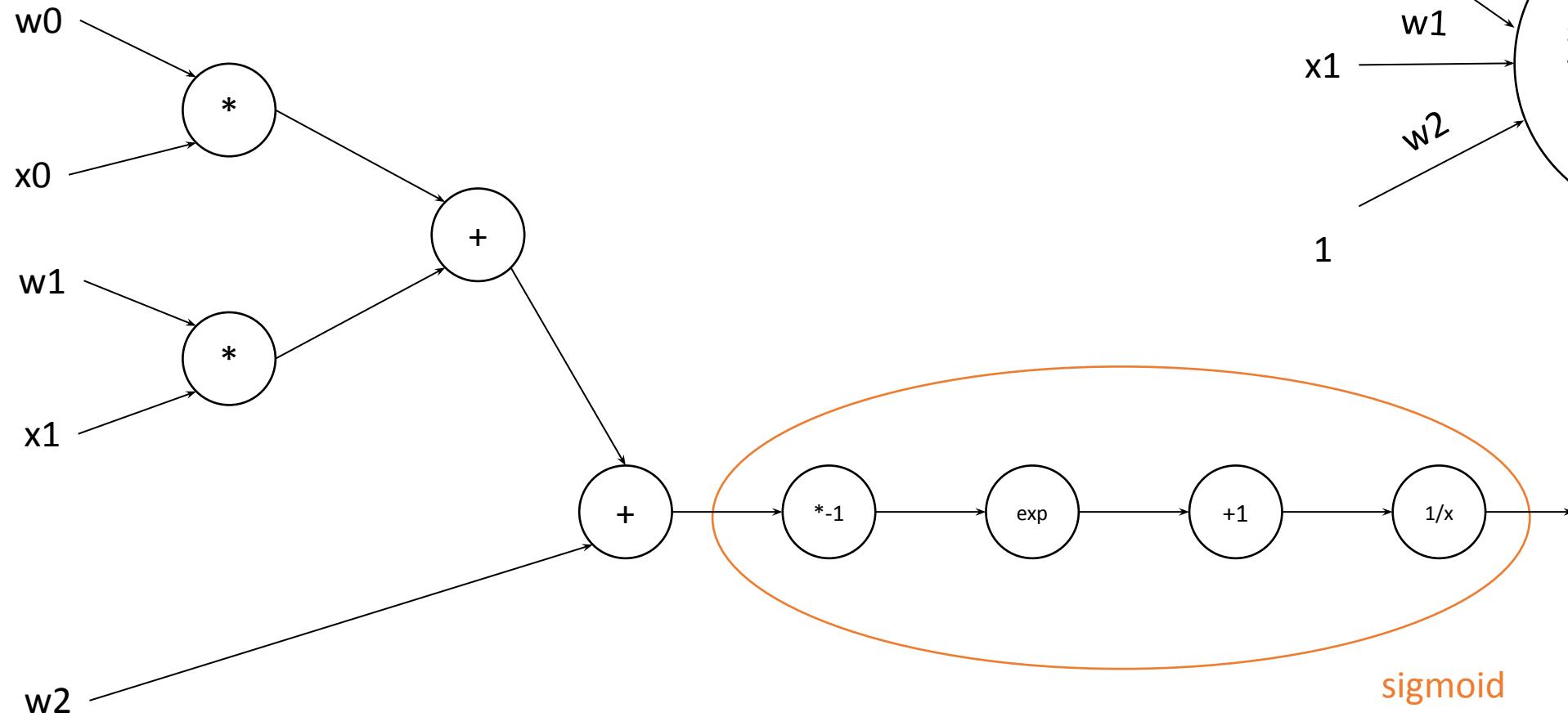
In a neural network a neuron computes a linear function, followed by a non-linearity (activation function).



Let's write this basic neuron as a computational graph!

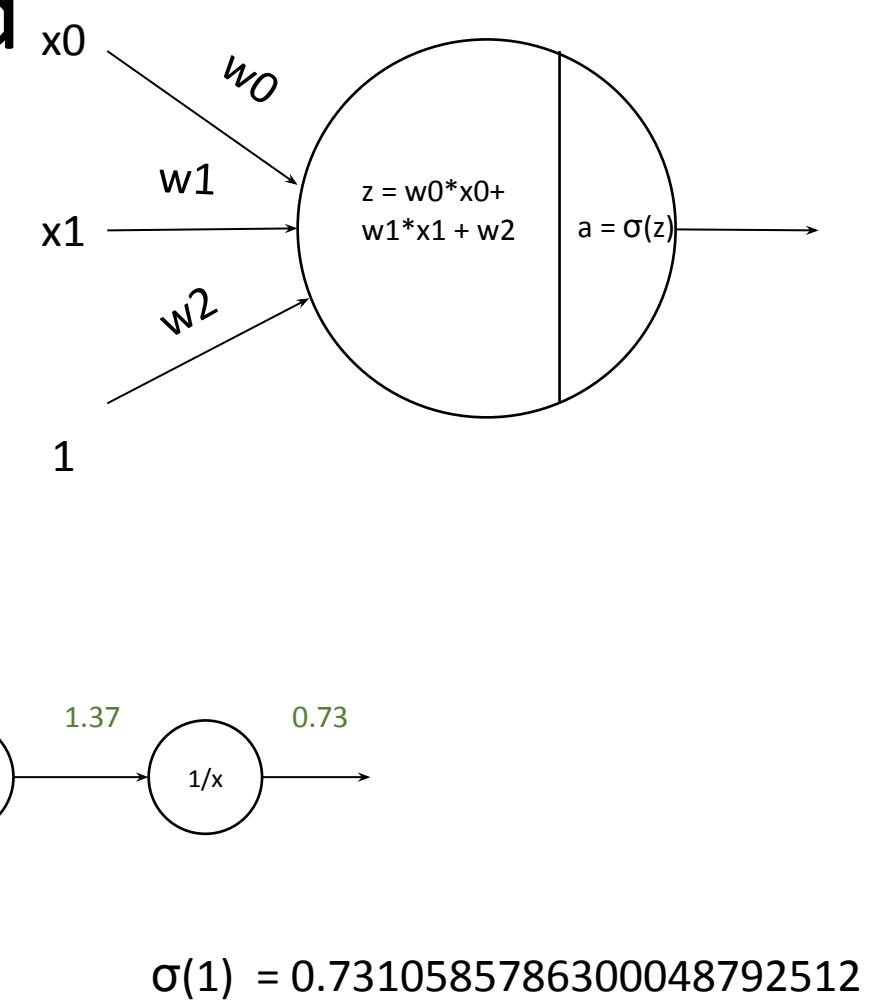
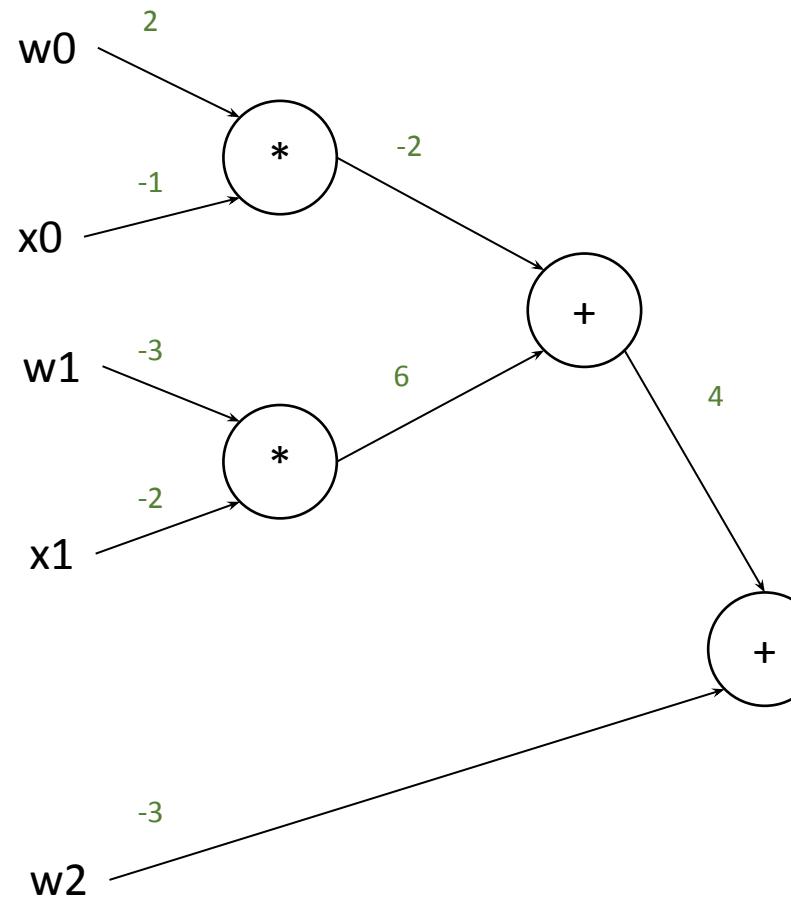
# Backprop example

## With a neural network in mind



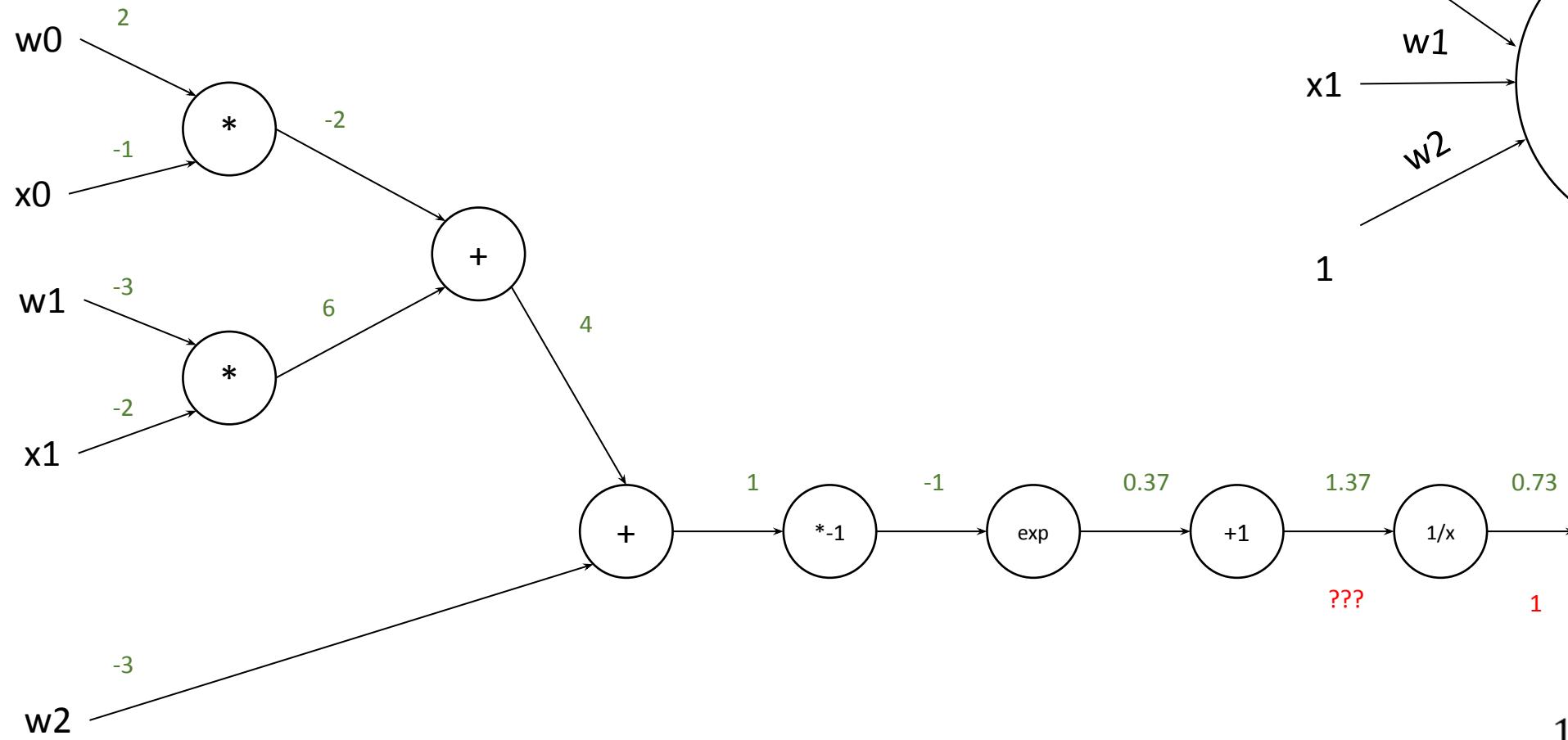
# Backprop example

## With a neural network in mind



# Backprop example

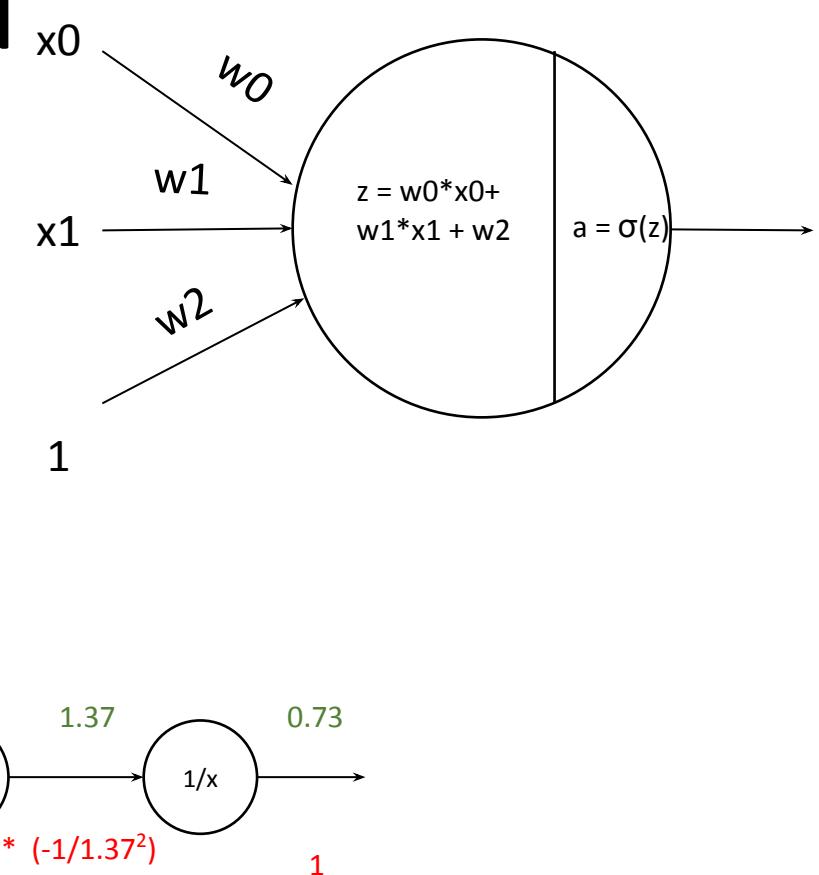
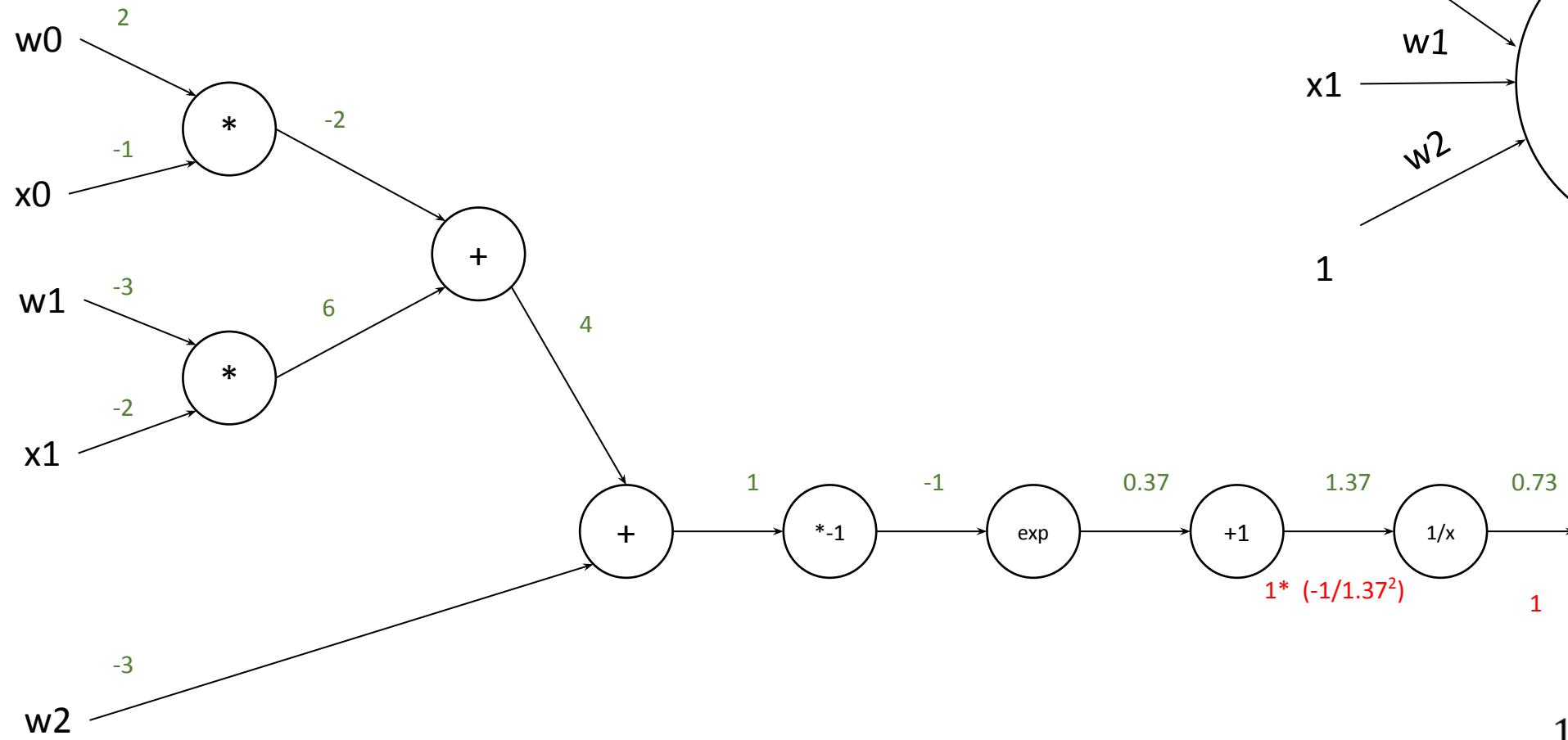
## With a neural network in mind



$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -\frac{1}{x^2}$$

# Backprop example

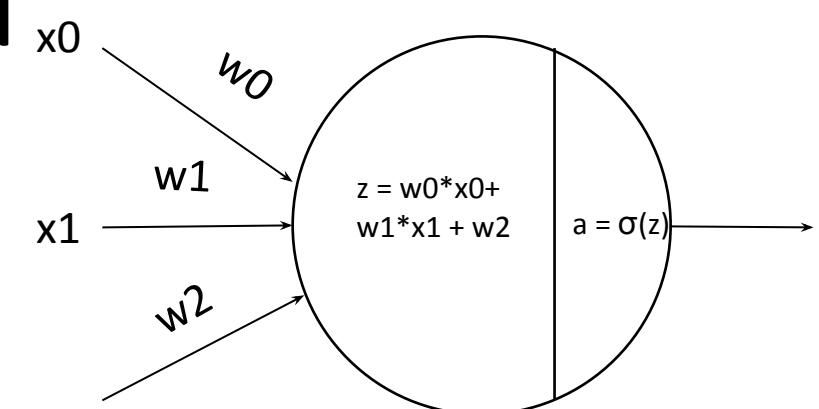
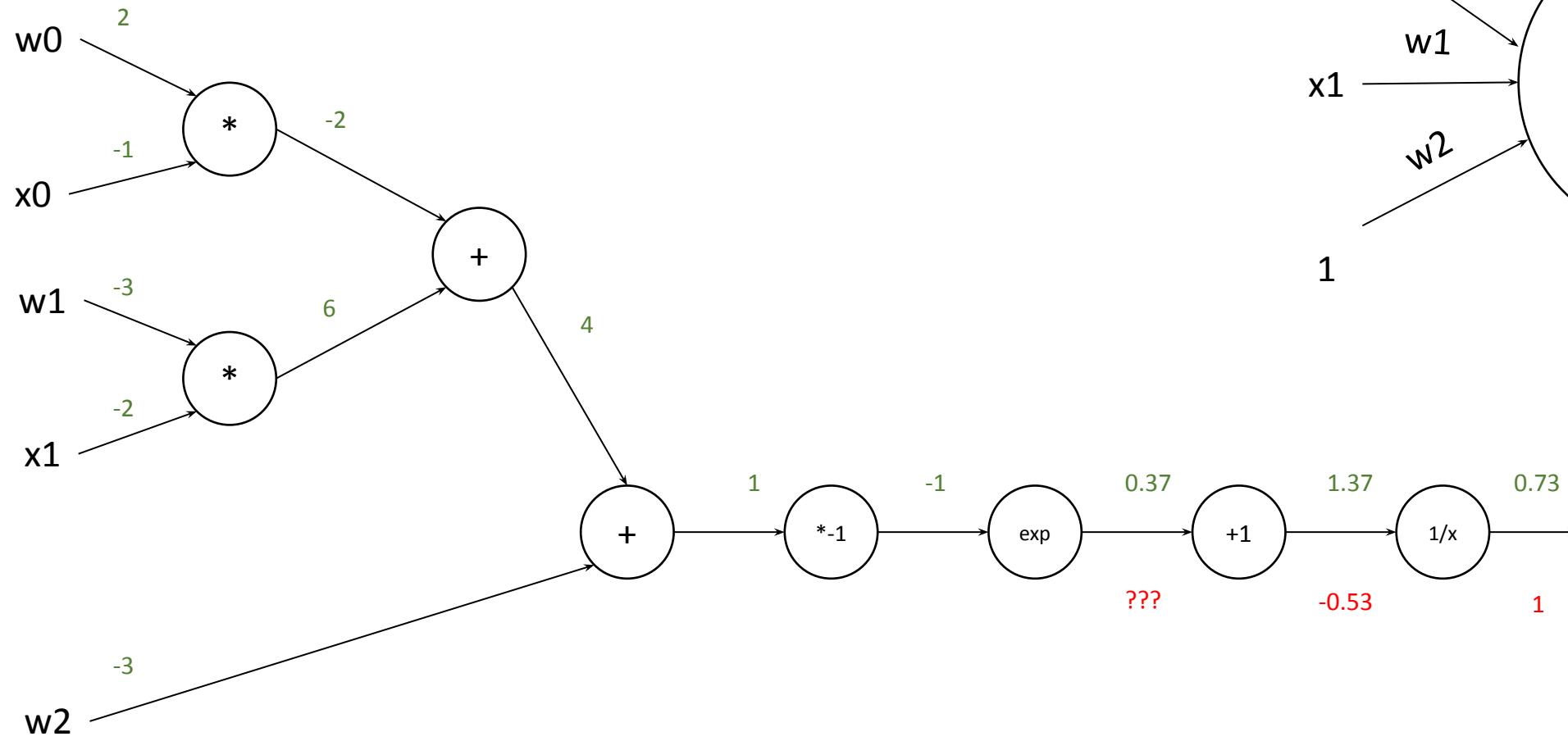
## With a neural network in mind



$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -\frac{1}{x^2}$$

# Backprop example

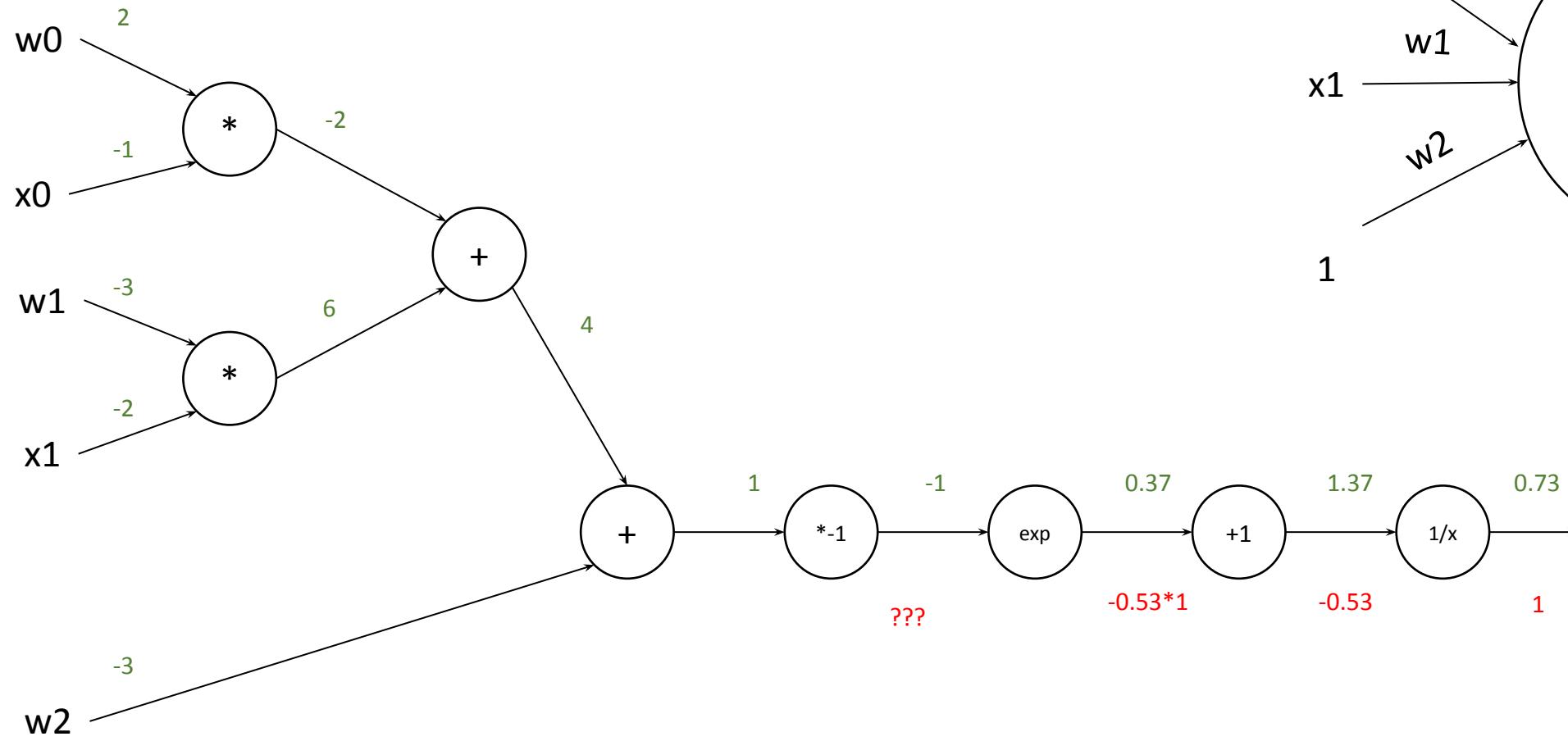
## With a neural network in mind



$$f(x) = x + c \rightarrow \frac{df}{dx} = 1$$

# Backprop example

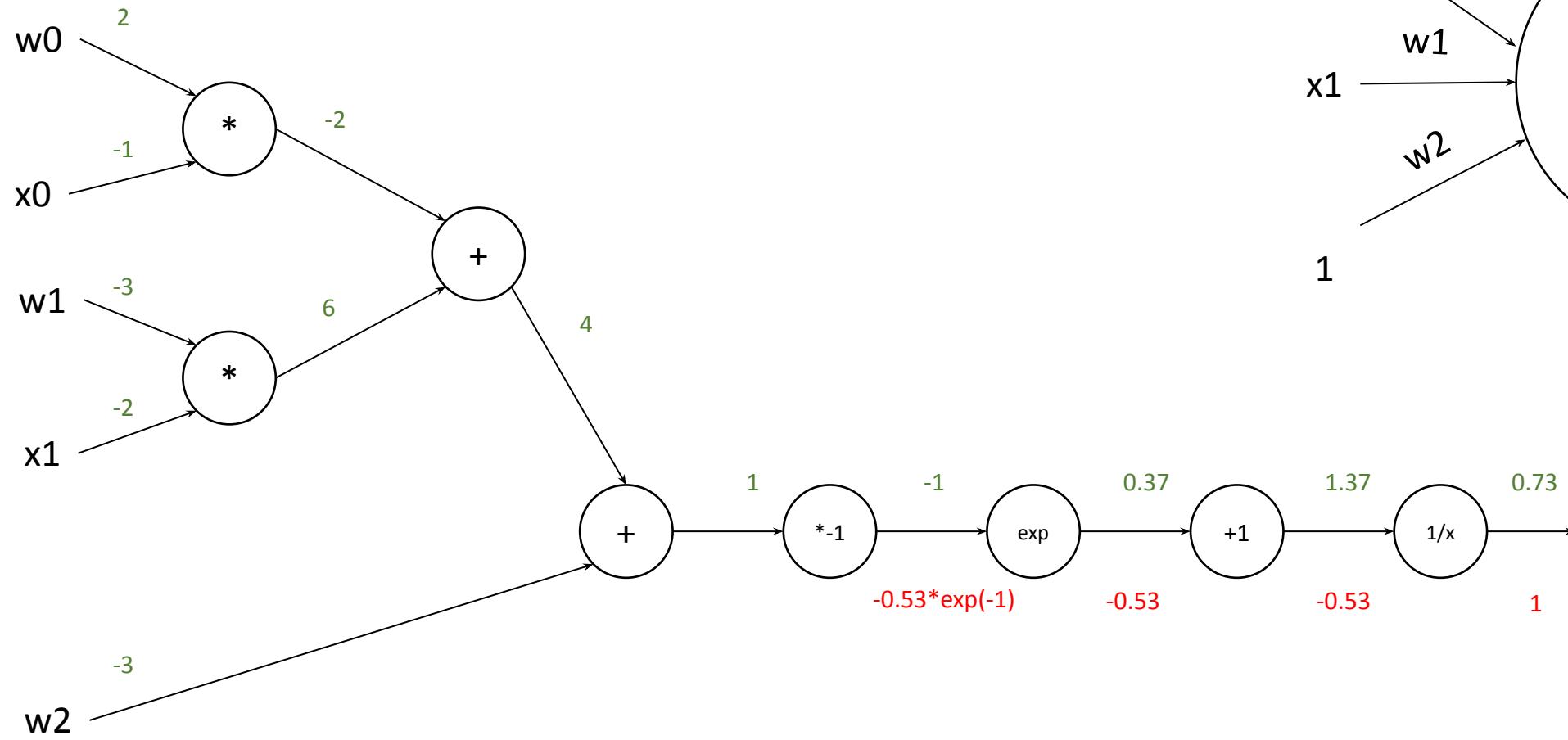
## With a neural network in mind



$$f(x) = x + c \rightarrow \frac{df}{dx} = 1$$

# Backprop example

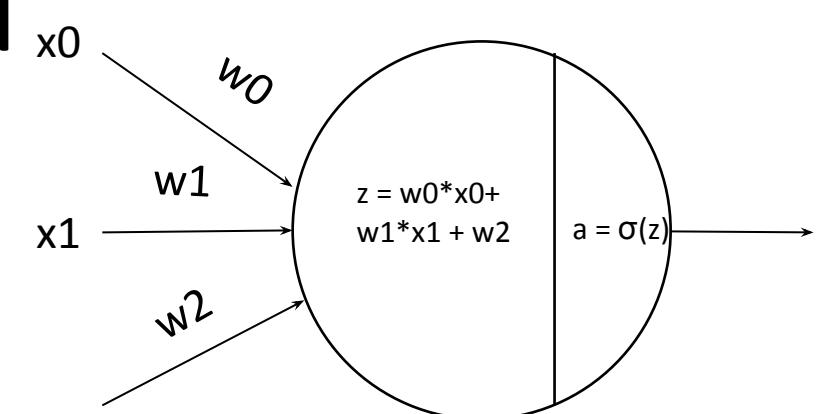
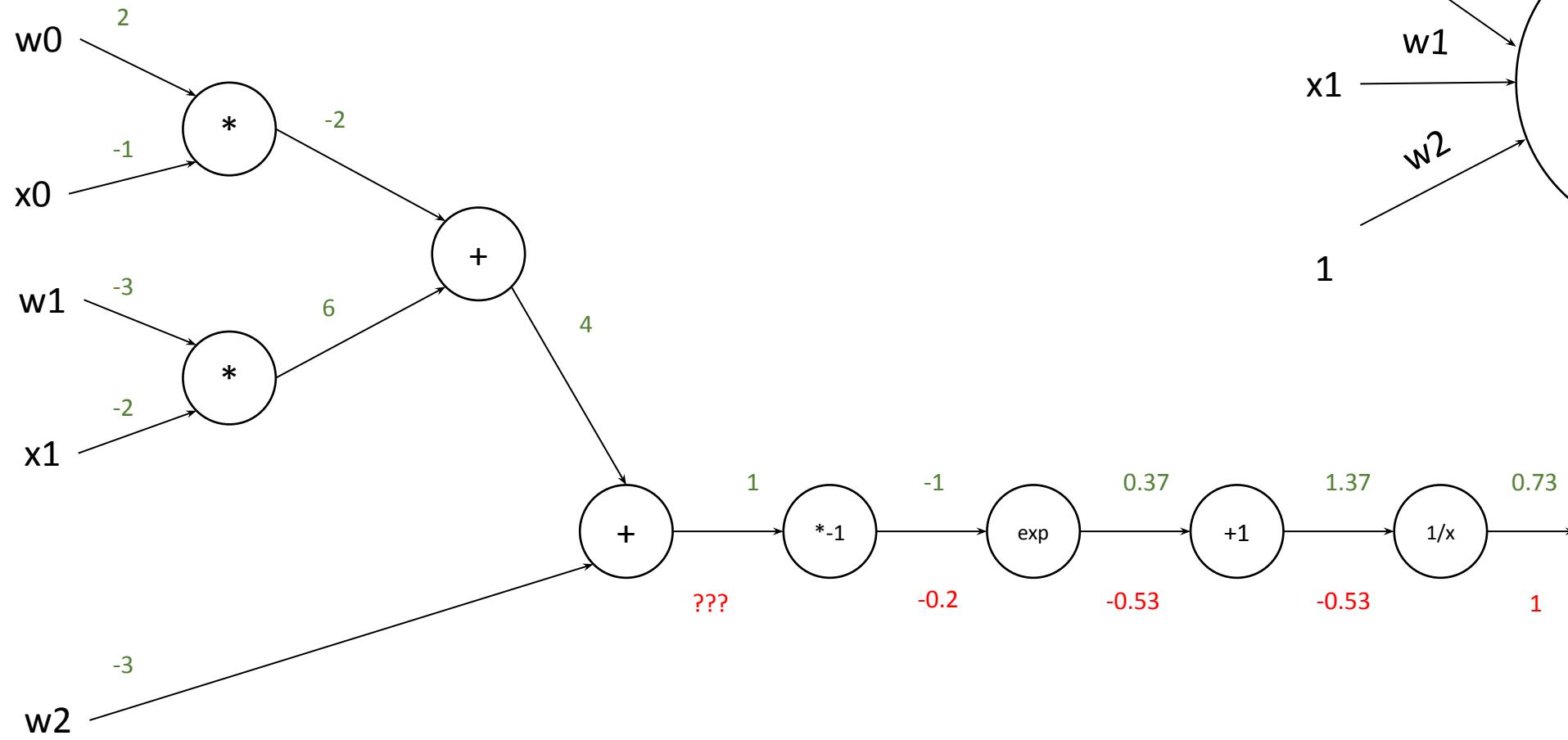
## With a neural network in mind



$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

# Backprop example

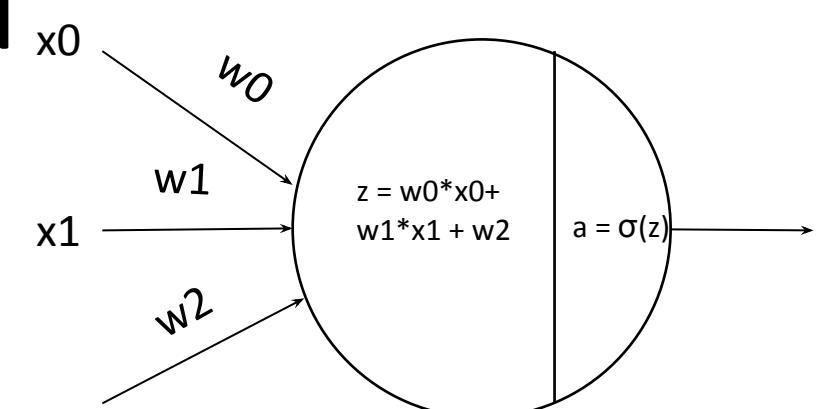
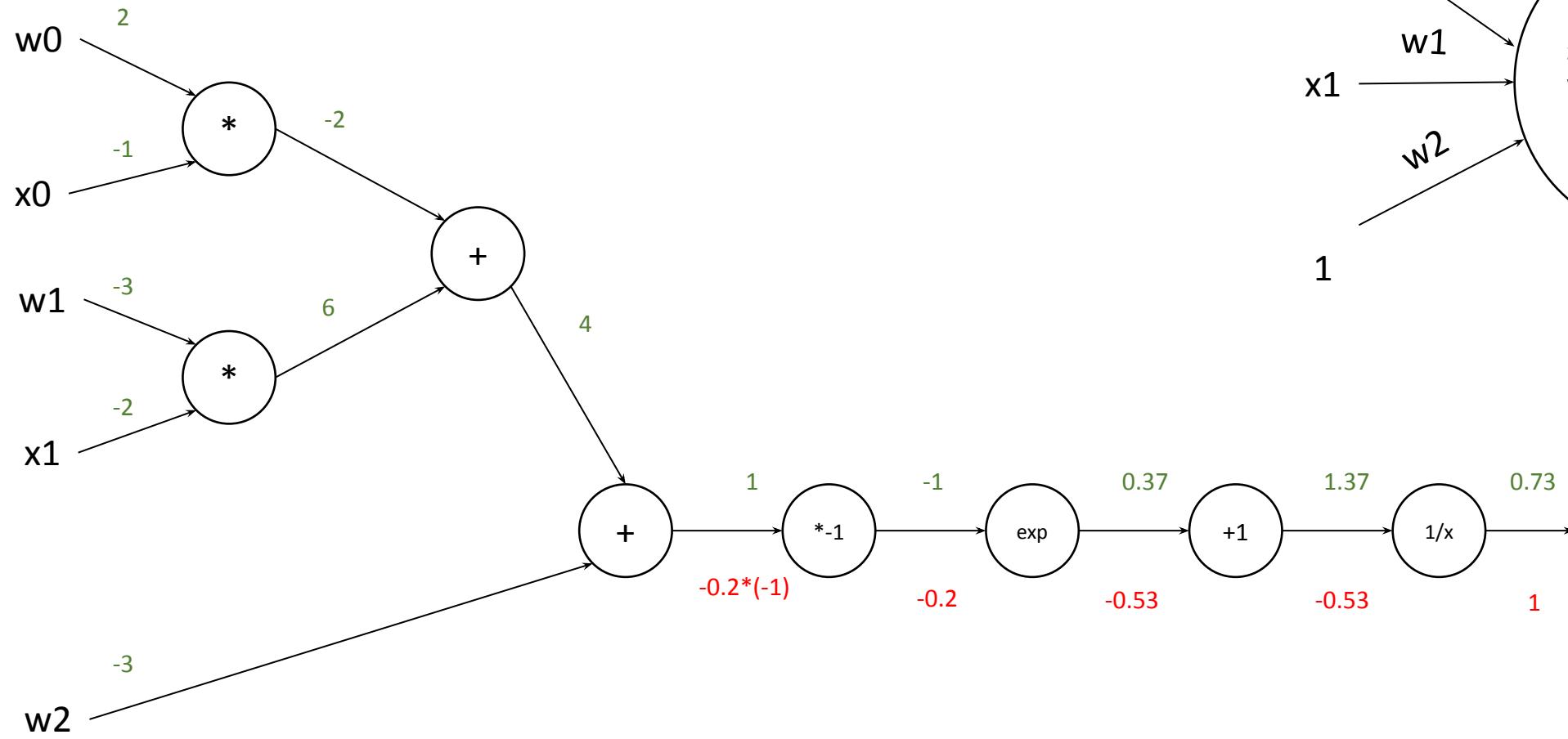
## With a neural network in mind



$$f(x) = ax \rightarrow \frac{df}{dx} = a$$

# Backprop example

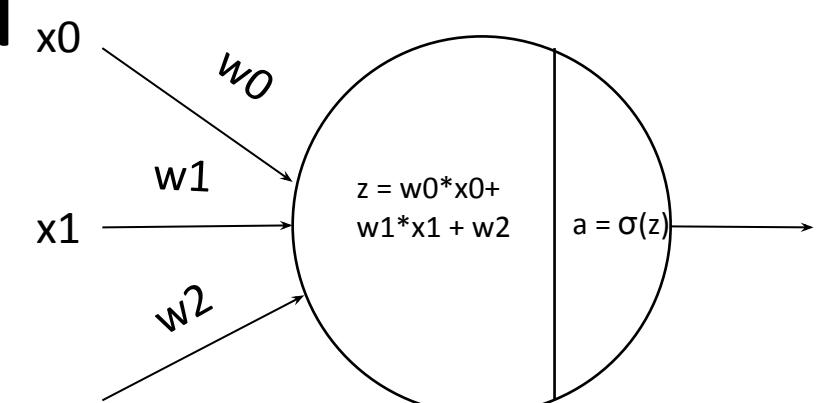
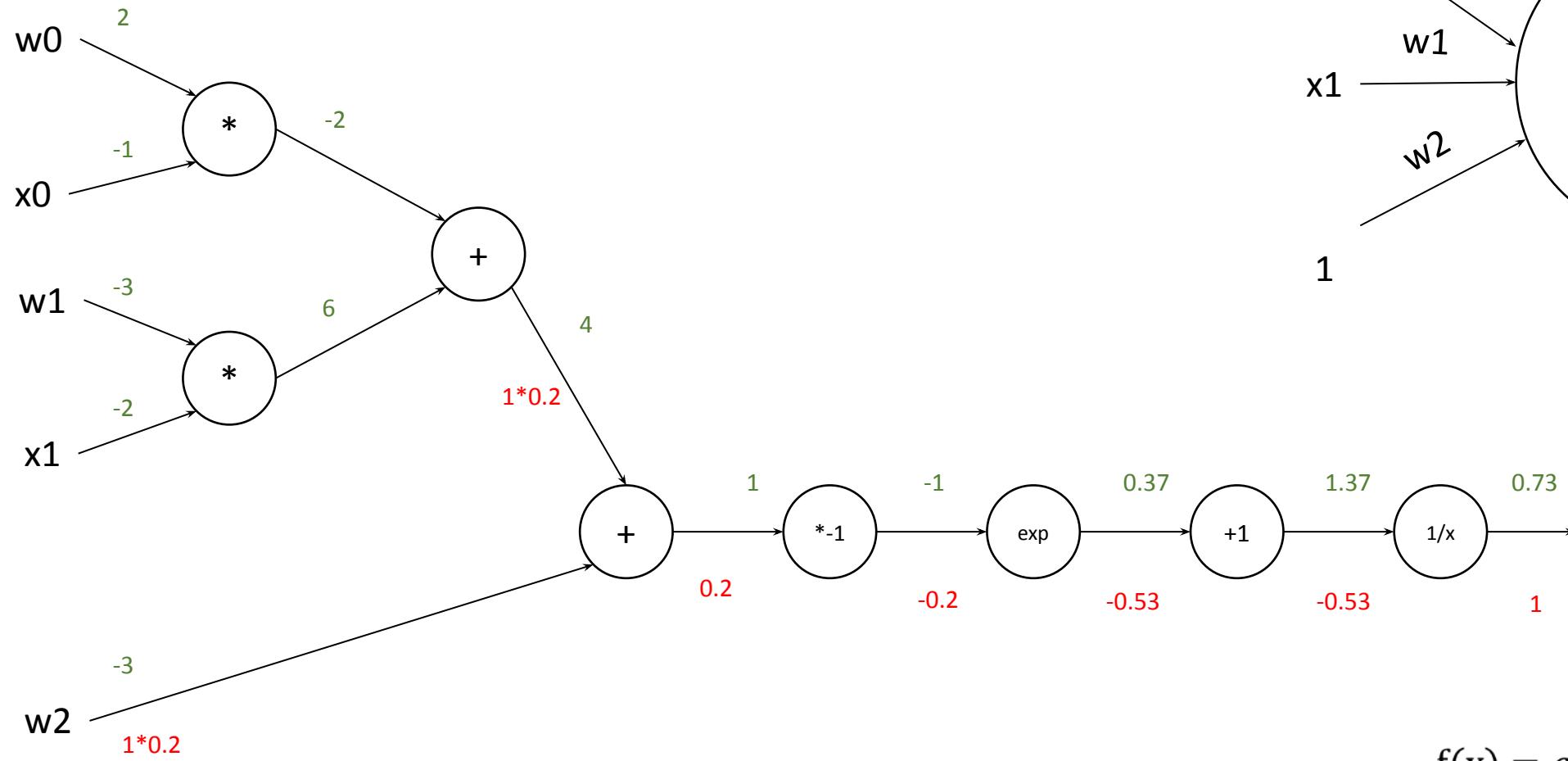
## With a neural network in mind



$$f(x) = ax \rightarrow \frac{df}{dx} = a$$

# Backprop example

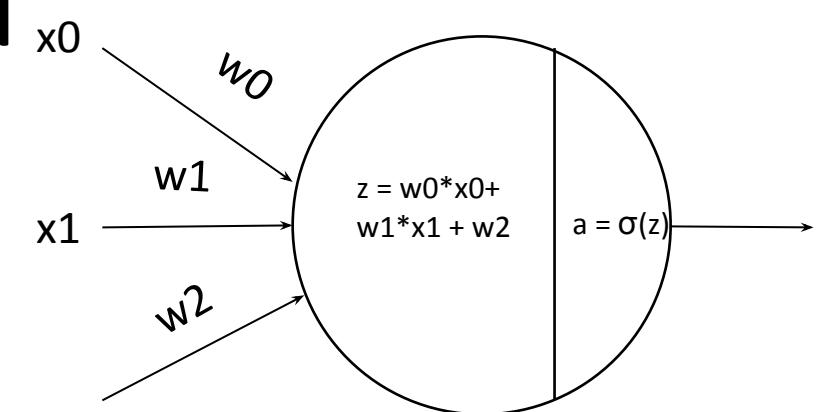
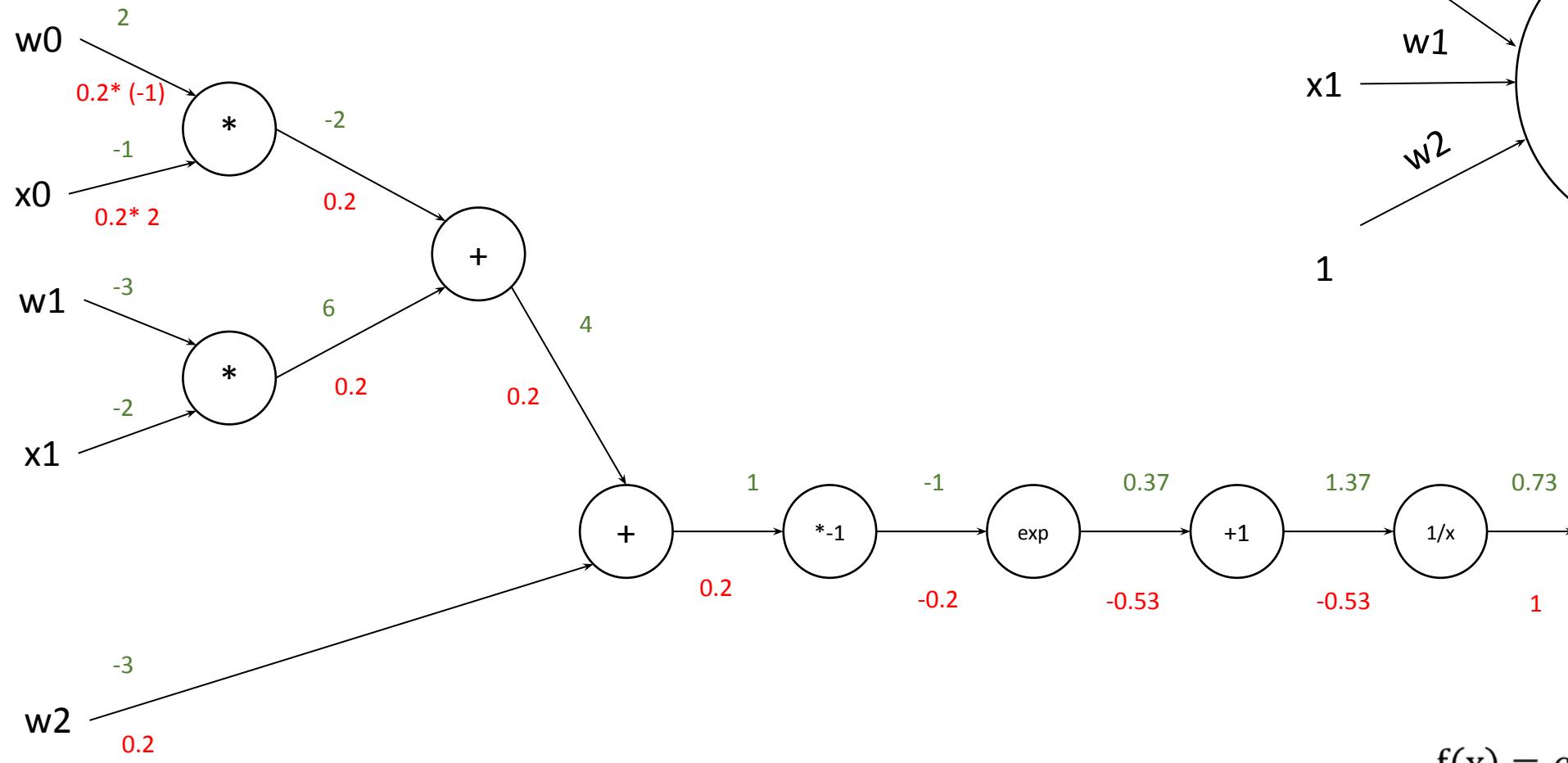
## With a neural network in mind



$$f(x) = c + x \rightarrow \frac{df}{dx} = 1$$

# Backprop example

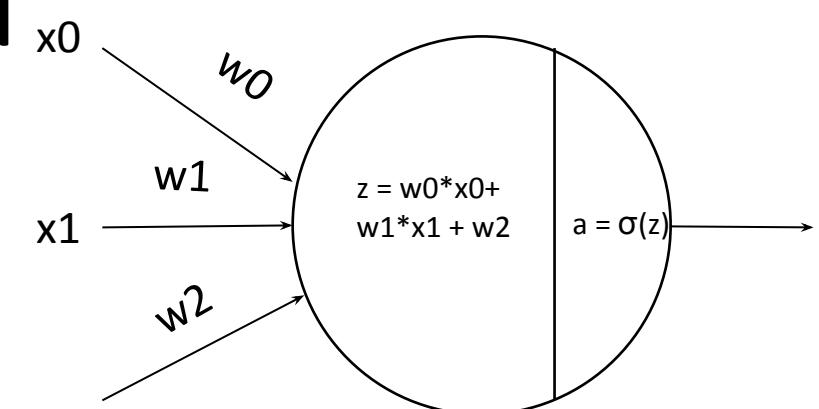
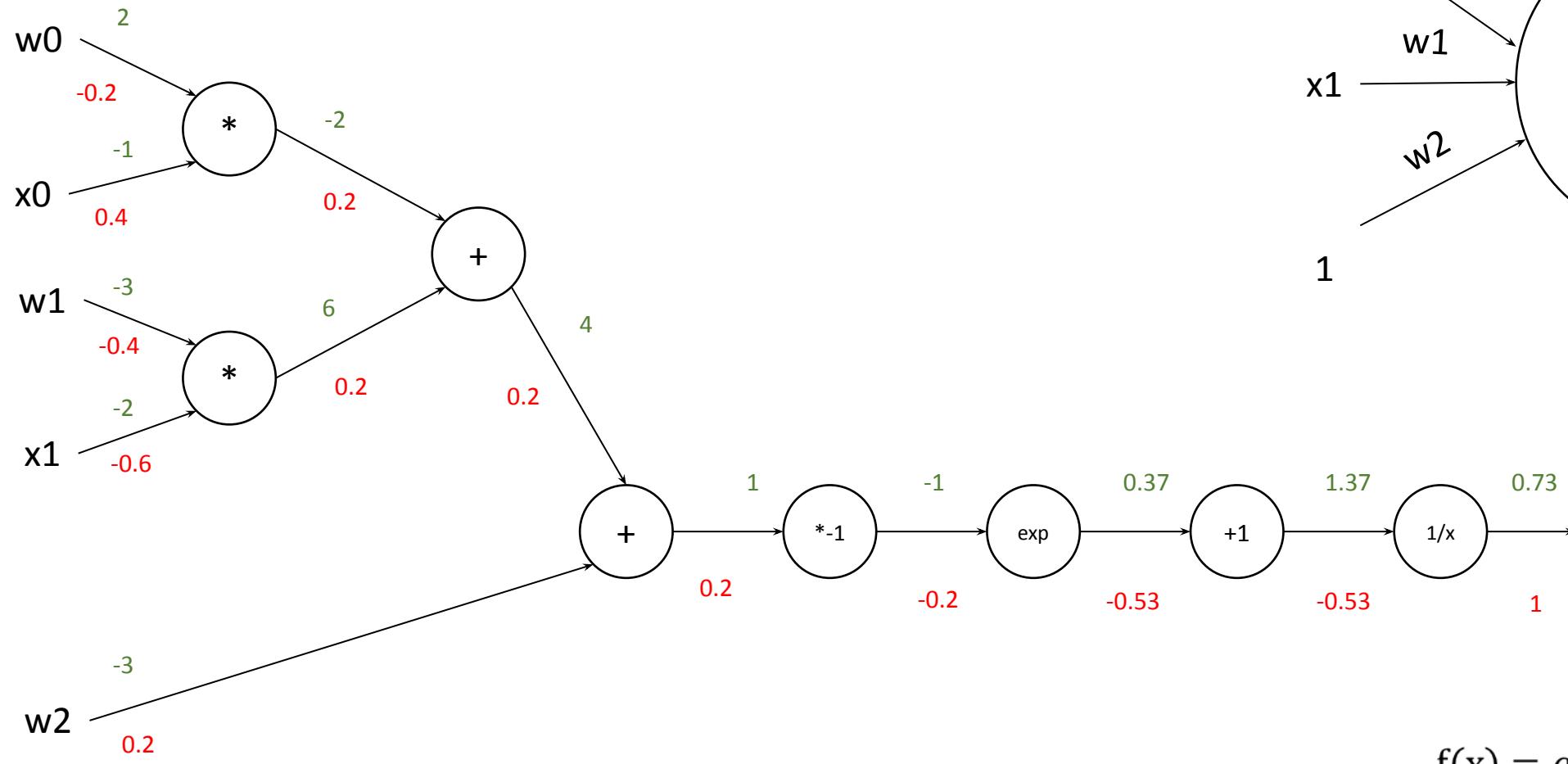
## With a neural network in mind



$$f(x) = ax \rightarrow \frac{df}{dx} = a$$

# Backprop example

## With a neural network in mind

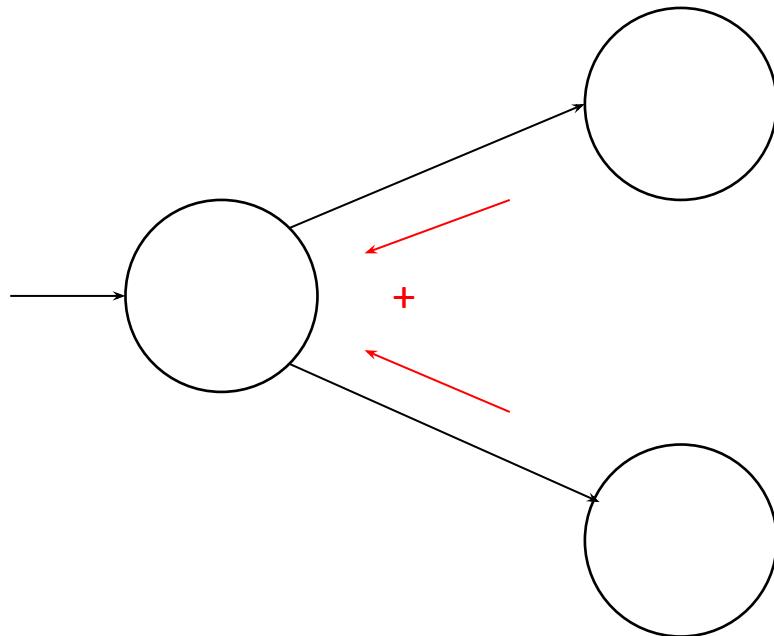


$$f(x) = ax \rightarrow \frac{df}{dx} = a$$

# Multivariate chain rule

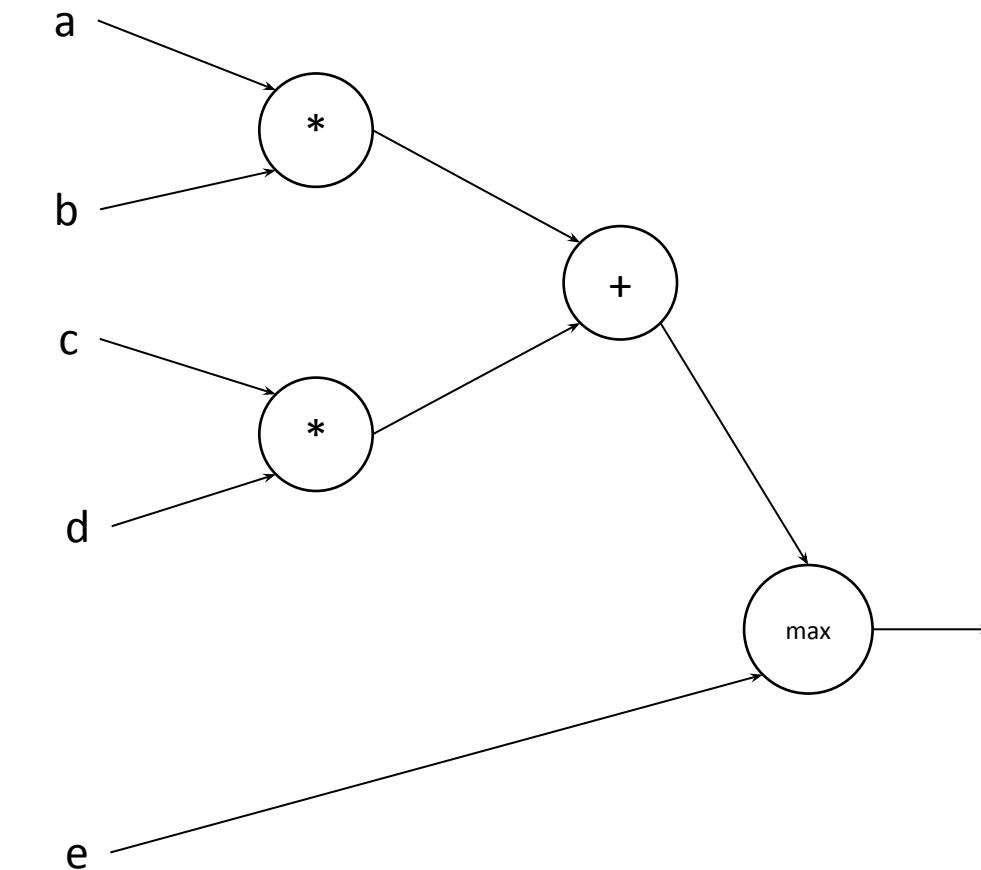
- 

$$\frac{\partial f}{\partial x} = \sum_i \frac{\partial f}{\partial q_i} \frac{\partial q_i}{\partial x}$$



# Patterns in backpropagation

- Addition gate: gradient distributor
- Multiplication gate: gradient switcher
- Max gate: gradient router



# Back-propagation for vectors

## Scalar to Scalar

$$x \in \mathbb{R}, y \in \mathbb{R}$$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If  $x$  changes by a small amount, how much will  $y$  change?

## Vector to Scalar

$$x \in \mathbb{R}^N, y \in \mathbb{R}$$

Derivative is **Gradient**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N \quad \left( \frac{\partial y}{\partial x} \right)_n = \frac{\partial y}{\partial x_n}$$

For each element of  $x$ , if it changes by a small amount then how much will  $y$  change?

## Vector to Vector

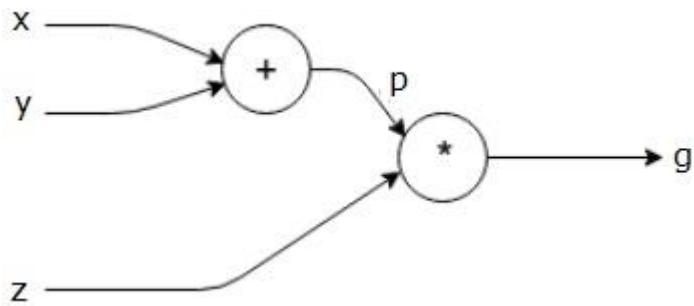
$$x \in \mathbb{R}^N, y \in \mathbb{R}^M$$

Derivative is **Jacobian**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} \quad \left( \frac{\partial y}{\partial x} \right)_{n,m} = \frac{\partial y_m}{\partial x_n}$$

For each element of  $x$ , if it changes by a small amount then how much will each element of  $y$  change?

# Forward/backward API



```
class ComputationalGraph():
    ...
    def forward(inputs):
        for node in topological_sort(self.nodes):
            # forward inputs through each node
            # (also cache these inputs)
            node.forward()
        return predictions
    ...
    def backward(loss):
        for node in reversed(topological_sort(self.nodes)):
            # backpropagation (apply chain rule)
            node.backward()
        return input_gradients
```

# Mark I Perceptron machine: first implementation of the perceptron algorithm (~1957)

- It was connected to a camera with  $20 \times 20$  cadmium sulfide photocells to make a 400-pixel image
- Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors

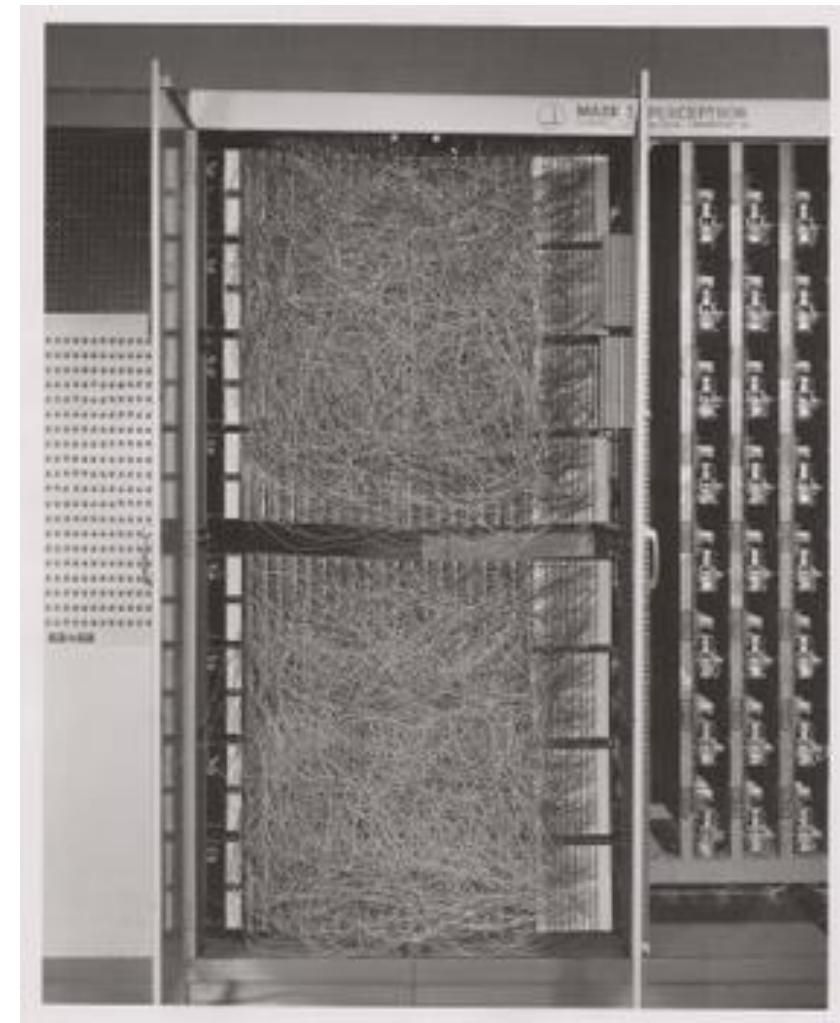
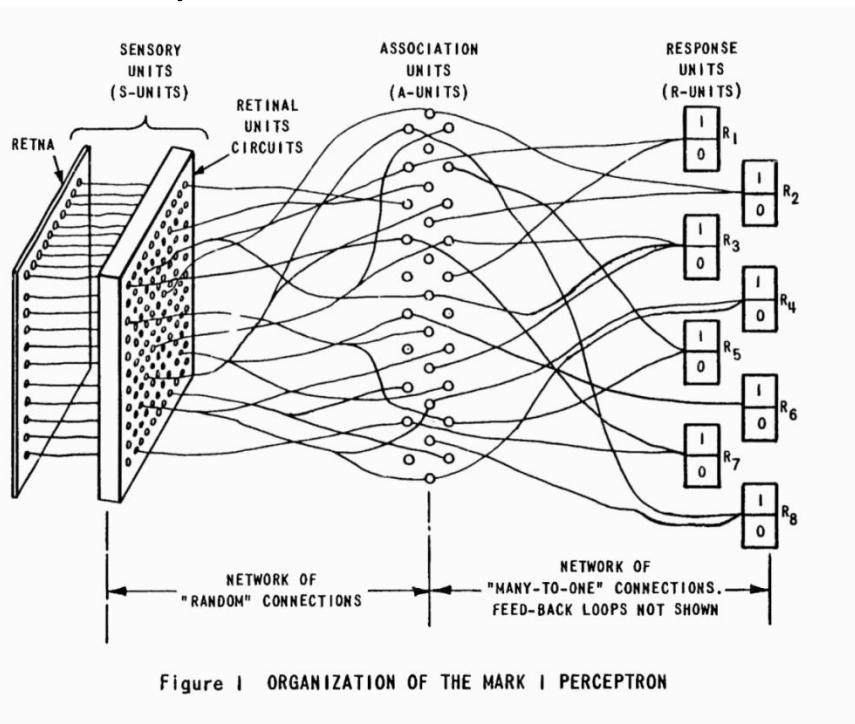
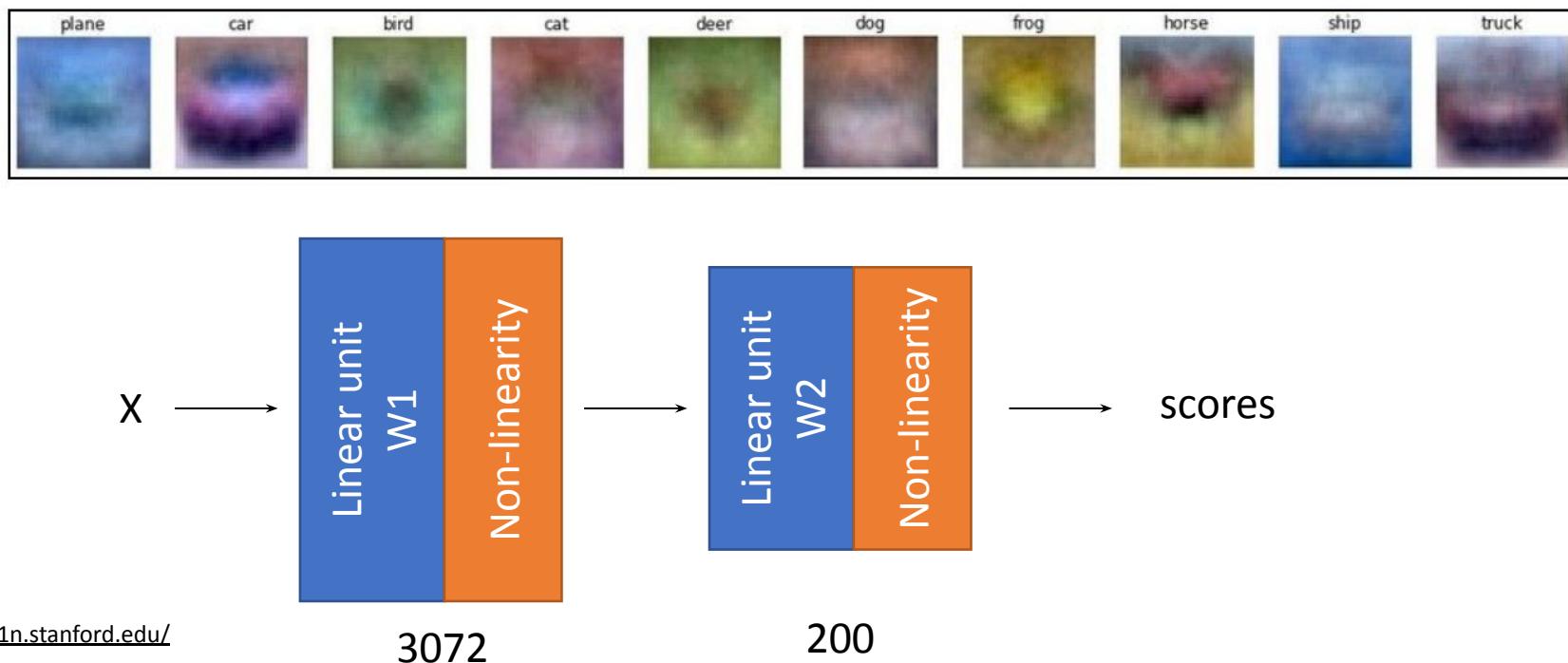


Image source: <https://en.wikipedia.org/wiki/Perceptron>

[https://www.youtube.com/watch?v=cNxadbrN\\_al](https://www.youtube.com/watch?v=cNxadbrN_al)

# Neural networks

- Neural network (multi layer perceptron, fully connected networks) :
    - Stack two (or more) linear classifiers on top of each other with a non-linear activation function in between



# Why do we need activation functions?

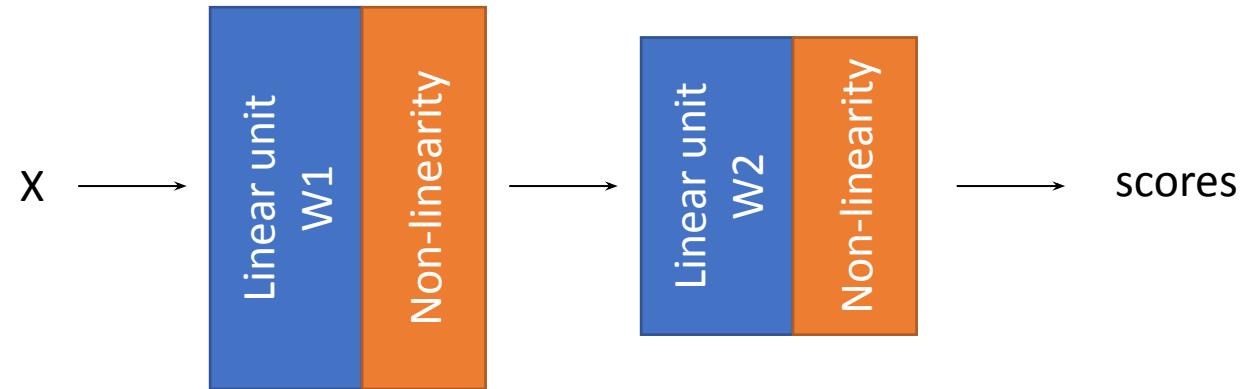
Layer 1:

$$Z_1 = W_1 \cdot X$$

Layer 2:

$$Z_2 = W_2 \cdot g(W_1 \cdot X), \text{ where } g \text{ is the activation function}$$

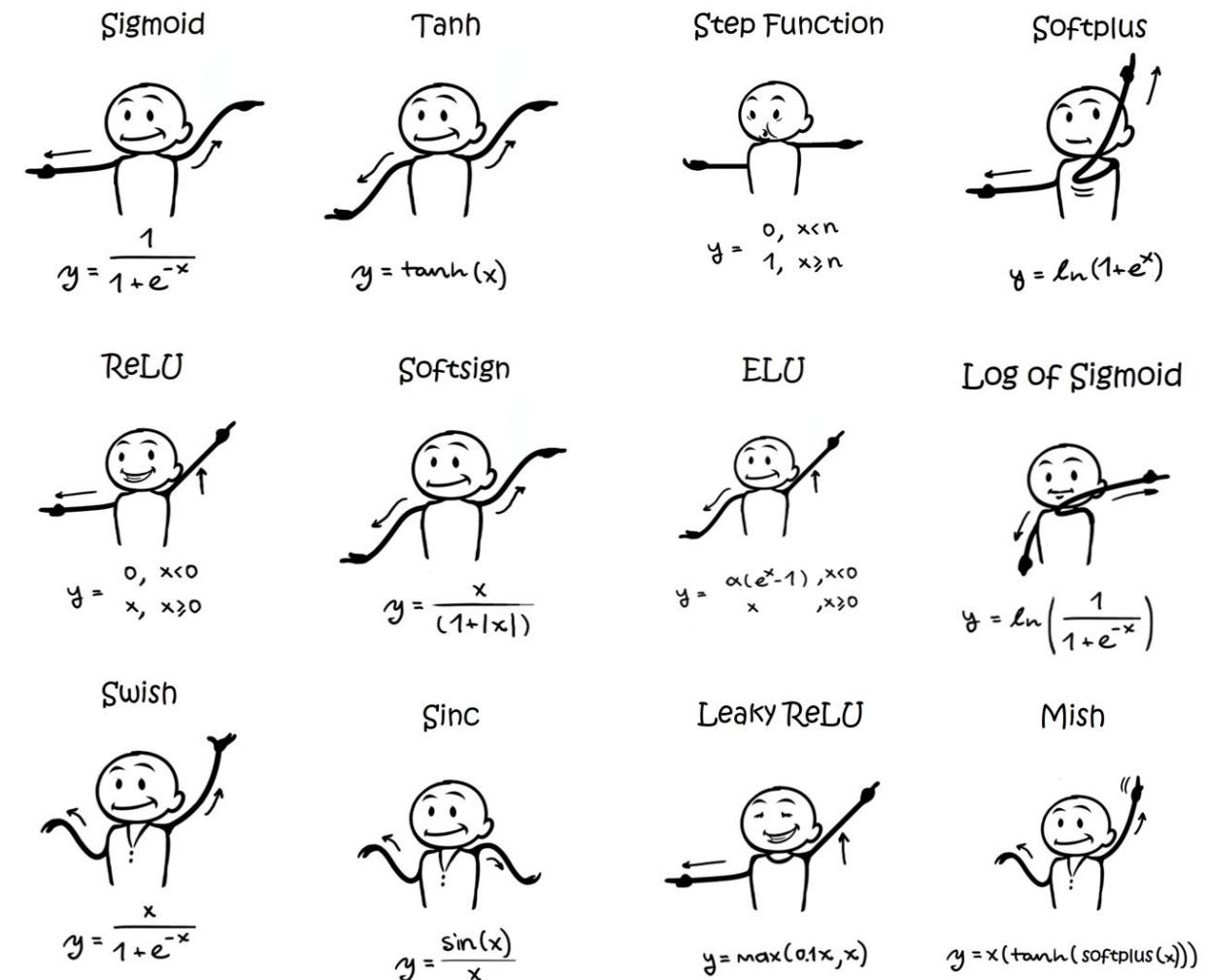
What happens if we didn't use a non-linearity?



# Activation functions

Two properties:

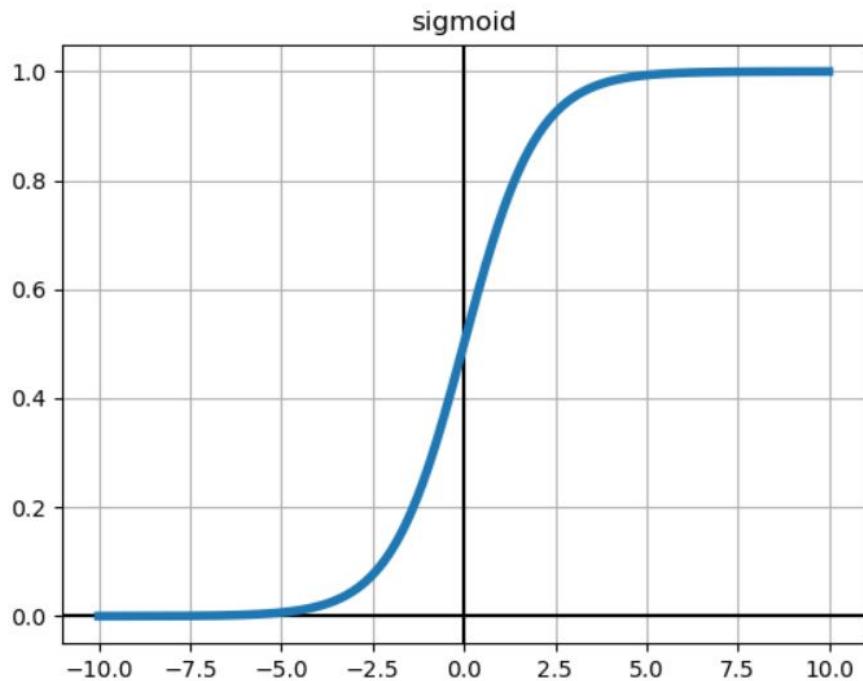
- Differentiable
- Non linear



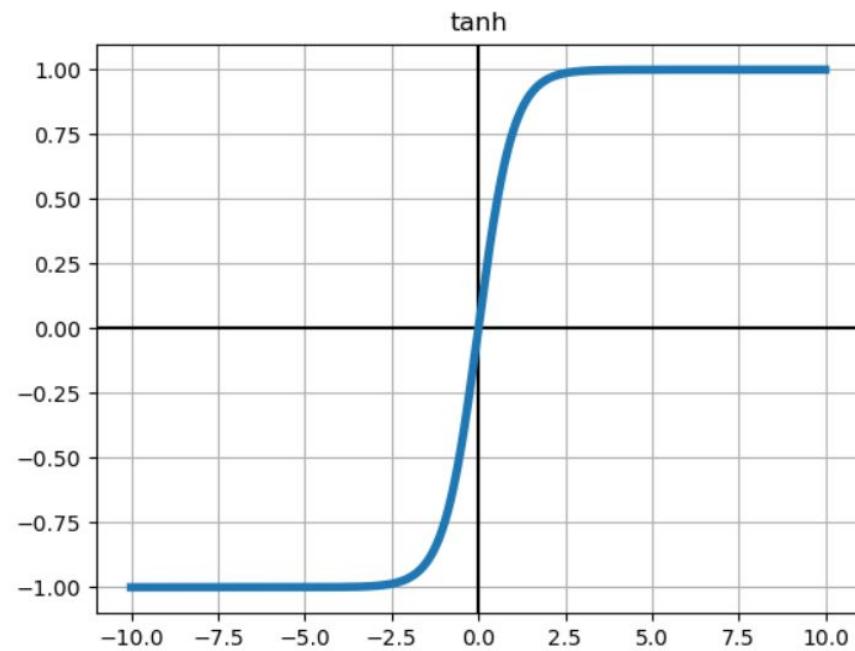
Commonly used activation functions and their derivative

[https://ml-cheatsheet.readthedocs.io/en/latest/activation\\_functions.html](https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html)

# Activation functions



$$\sigma(x) = \frac{1}{1+e^{-x}}$$



$$\tanh(x)$$

# Activation functions

- GELU

$$0.5x(1 + \tanh[\sqrt{2/\pi}(x + 0.044715x^3)])$$

- ELU

$$R(z) = \begin{cases} z & z > 0 \\ \alpha \cdot (e^z - 1) & z \leq 0 \end{cases}$$

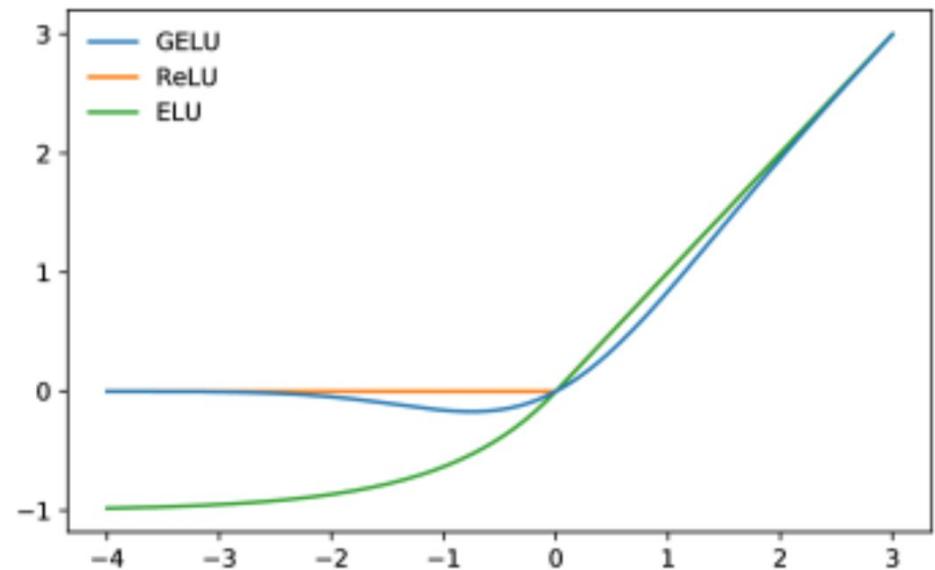
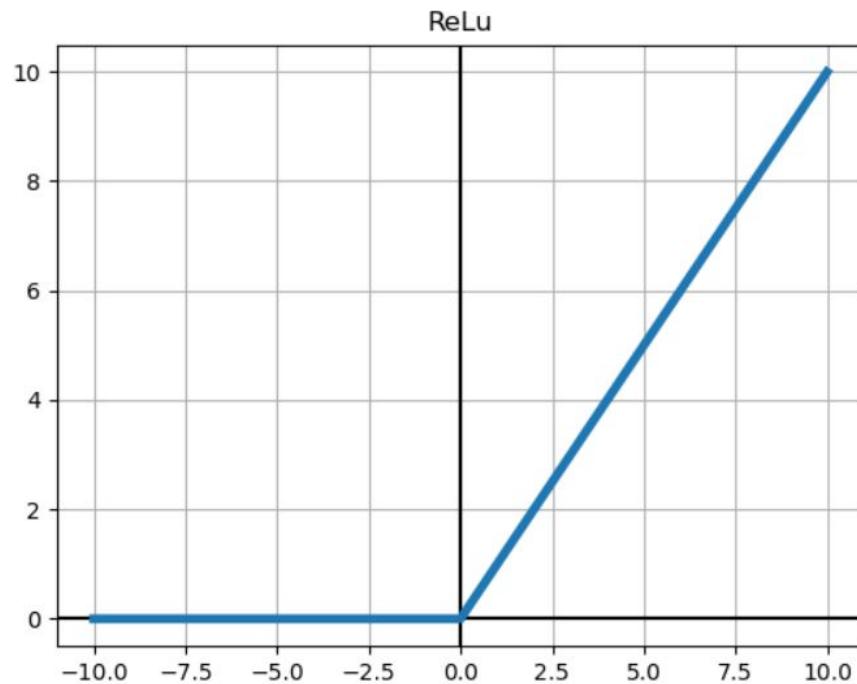
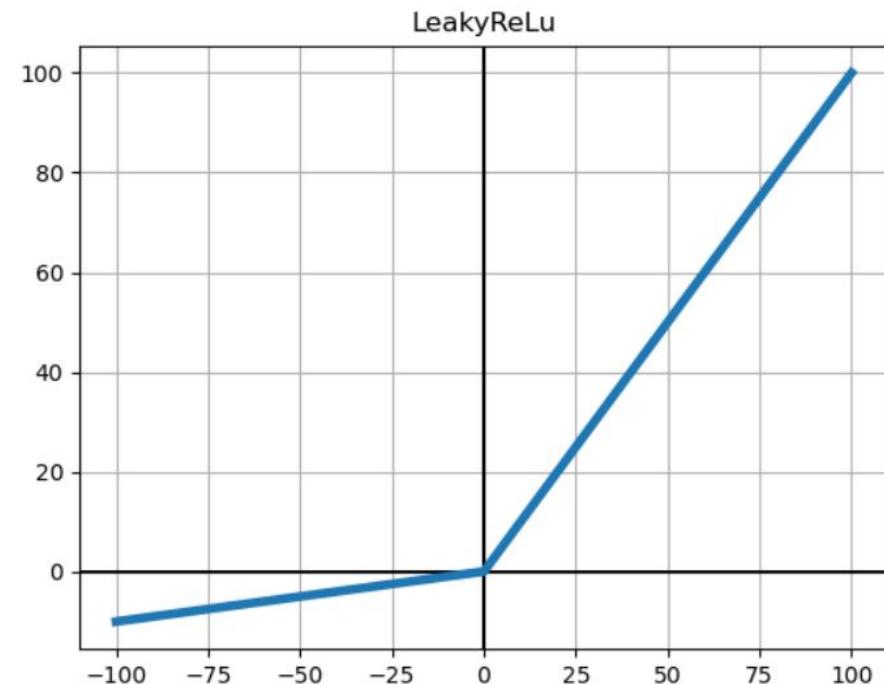


Figure 1: The Gaussian Error Linear Unit ( $\mu = 0, \sigma = 1$ ), the Rectified Linear Unit, and the Exponential Linear Unit ( $\alpha = 1$ ).

# Activation functions

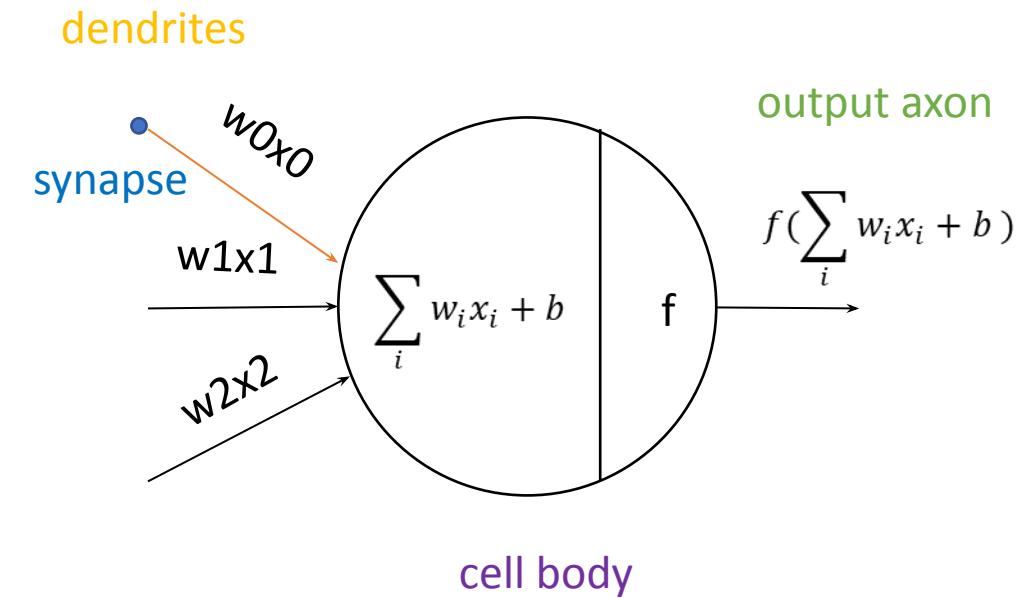
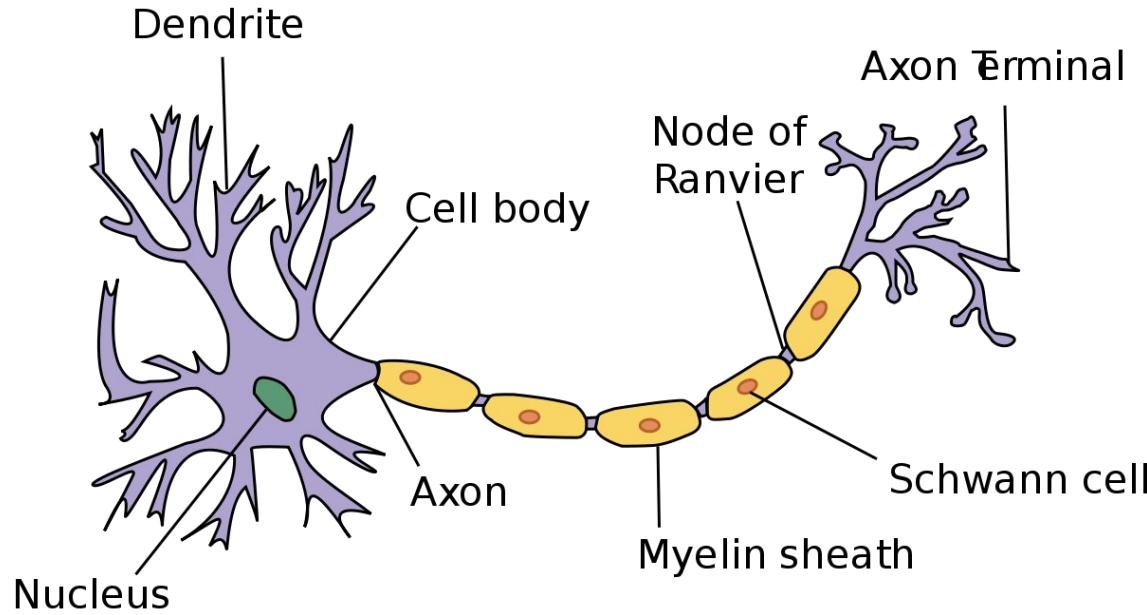


$$\max(0, x)$$



$$\max(0.1x, x)$$

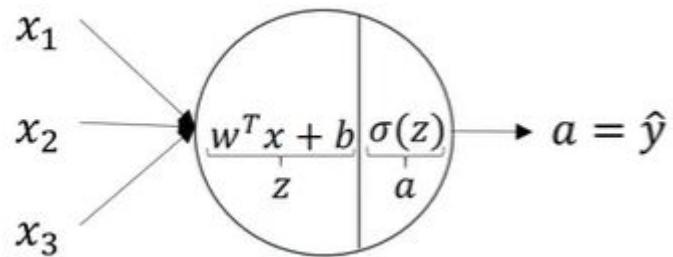
# Neural networks



# Neural network and the brain

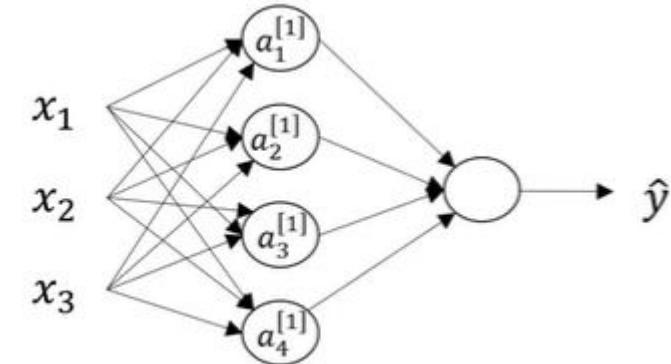
- deeplearning.ai Course 1, Week 4: *What does this have to do with the brain?*
  - <https://www.coursera.org/lecture/neural-networks-deep-learning/what-does-this-have-to-do-with-the-brain-objnR>
- We should be more reserved about the brain analogies
  - There are actually many different types of biological neurons
  - Dendrites can perform complex non-linear operations
  - Complex connectivity patterns

# Neural networks - notation



$a^{[l]}$  - the activations of the  $l^{\text{th}}$  layer

$a_i^{[l]}$  - the activations of  $i^{\text{th}}$  neuron in the  $l^{\text{th}}$  layer



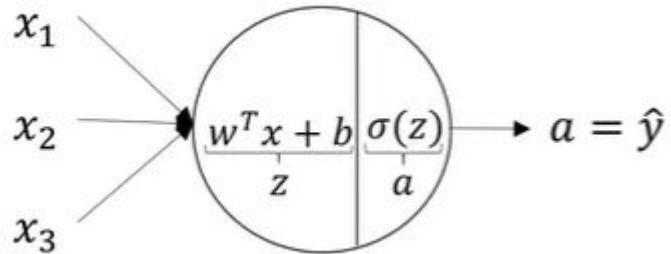
2 layer neural network

We don't count the input layer!

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$

# Neural networks - notation



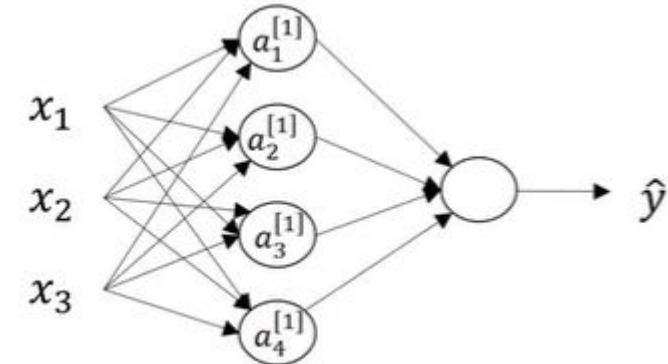
$a^{[l]}$  - the activations of the  $l^{\text{th}}$  layer

$a_i^{[l]}$  - the activations of  $i^{\text{th}}$  neuron in the  $l^{\text{th}}$  layer

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$

$$Z^{[l]} = W^{[l]} \cdot X + b^{[l]}$$



2 layer neural network

We don't count the input layer!

$$X: (3, 1)$$

$$W^{[1]}: (4, 3)$$

$$b^{[1]}: (4, 1)$$

$$z^{[1]}: (4, 1)$$

$$a^{[1]}: (4, 1)$$

$$a^{[2]}: (4, 1)$$

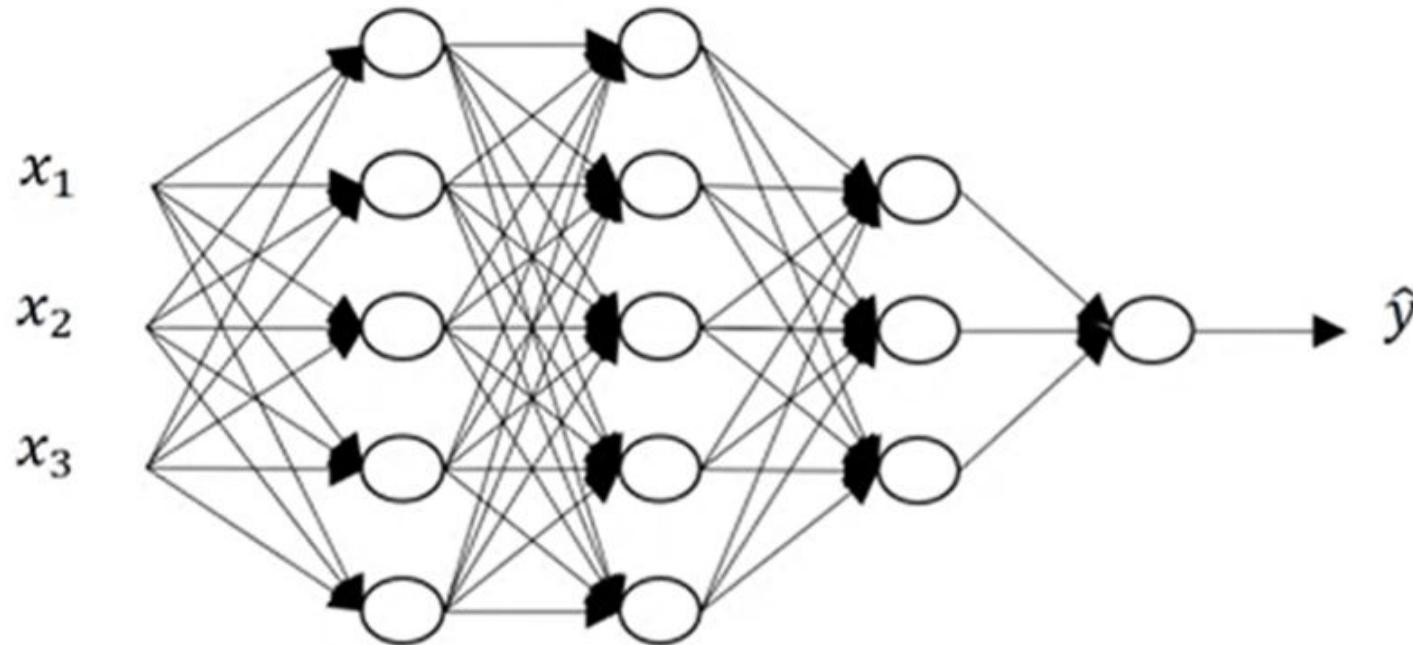
$$W^{[2]}: (1, 4)$$

$$b^{[2]}: (1, 1)$$

$$z^{[2]}: (1, 1)$$

$$a^{[2]}: (1, 1)$$

# Multiple layer hidden network



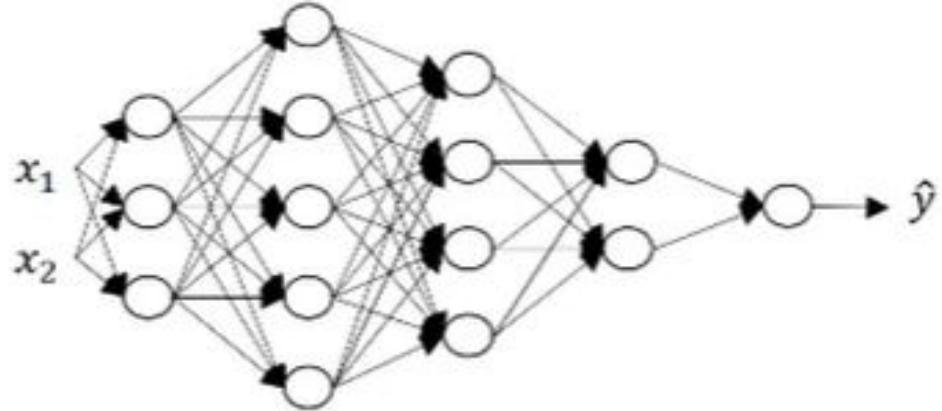
L – number of layers  
n<sup>[l]</sup> – number of hidden layers in each unit  
n<sup>[0]</sup> = n<sub>x</sub>  
W<sup>[l]</sup> – weight matrix of the l<sup>th</sup> layer  
b<sup>[l]</sup> – bias vector for the l<sup>th</sup> layer  
a<sup>[l]</sup> – activations computed by the l<sup>th</sup> layer

$$Z^{[1]} = W^{[1]} \cdot X + b^{[1]}$$

$$Z^{[2]} = W^{[2]} \cdot a^{[1]} + b^{[2]}$$

# Multiple layer hidden network

## Notation and matrix dimensions



$$z^{[1]} = W^{[1]} \cdot x + b^{[1]}$$

$$z^{[2]} = W^{[2]} \cdot a^{[1]} + b^{[2]}$$

**Matrix dimensions for layer L:**

$$z^{[l]} : (n^{[l]}, 1)$$

$$a^{[l]} : (n^{[l]}, 1)$$

$$W^{[l]} : (n^{[l]}, n^{[l-1]})$$

$$b^{[l]} : (n^{[l]}, 1)$$

$$dW^{[l]} : (n^{[l]}, n^{[l-1]})$$

$$db^{[l]} : (n^{[l]}, 1)$$

L – number of layers

$n^{[l]}$  – number of hidden layers in each unit

$$n^{[0]} = n_x$$

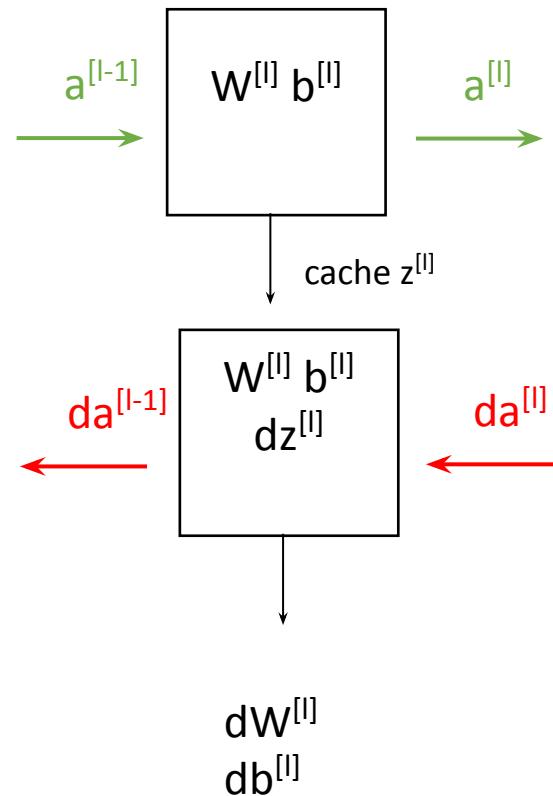
$W^{[l]}$  – weight matrix of the  $l^{\text{th}}$  layer

$b^{[l]}$  – bias vector for the  $l^{\text{th}}$  layer

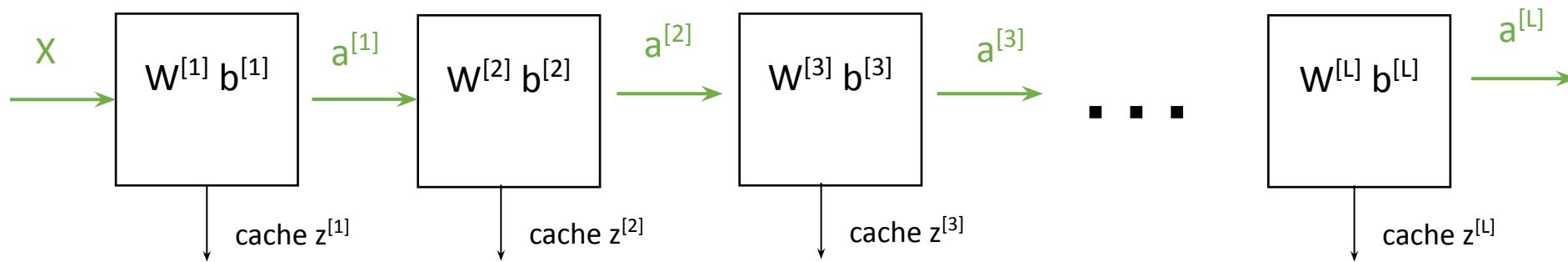
$a^{[l]}$  – activations computed by the  $l^{\text{th}}$  layer

Putting it all together

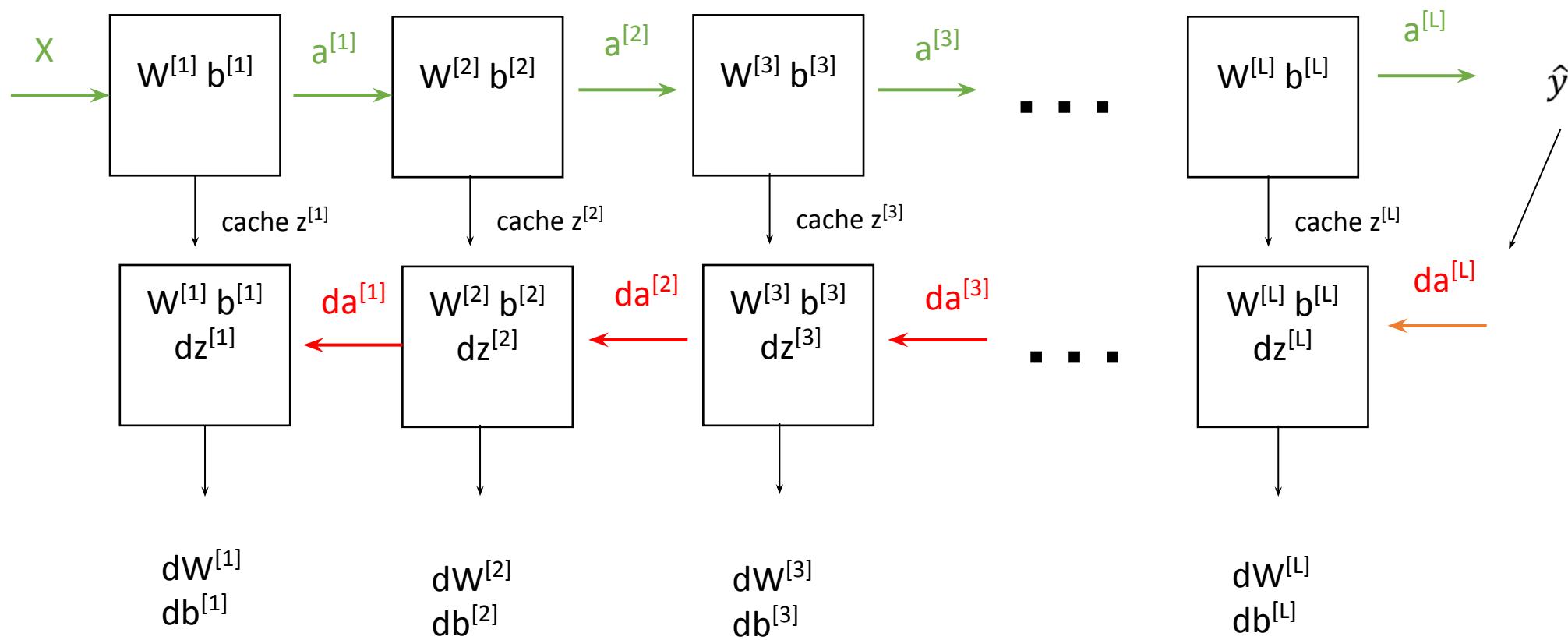
# Forward and backward computations at a given layer $l$



# Forward and backward functions in NN



# Forward and backward functions in NN



# Forward propagation for a layer $l$

- Input
  - $a^{[l-1]}$
- Output:
  - $a^{[l]}$
  - Cache:  $z^{[l]}, W^{[l]}, b^{[l]}$

$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$$

$$a^{[l]} = g^{[l]}(z^{[l]})$$

# Backward propagation for a layer $l$

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}}$$

$$C_0 = (a^{(L)} - y)^2$$

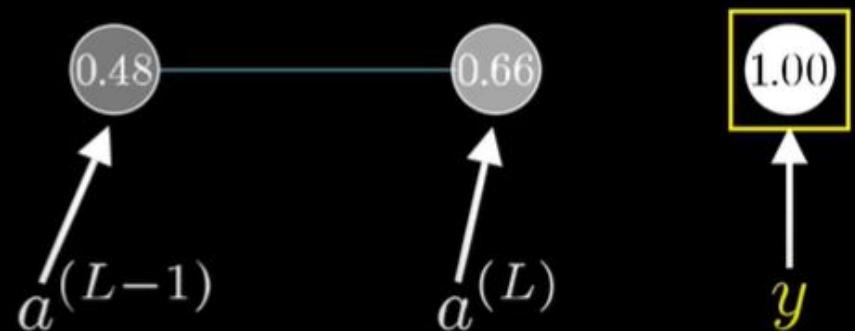
$$z^{(L)} = w^{(L)}a^{(L-1)} + b^{(L)}$$

$$\frac{\partial C_0}{\partial a^{(L)}} = 2(a^{(L)} - y)$$

$$a^{(L)} = \sigma(z^{(L)})$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \sigma'(z^{(L)})$$

$$\frac{\partial z^{(L)}}{\partial w^{(L)}} = a^{(L-1)}$$



# Backward propagation for a layer $l$

- Input
  - $da^{[l]}$
- Output
  - $da^{[l-1]}, dW^{[l]}, db^{[l]}$

$$dz^{[l]} = da^{[l]} * g^{[l]'}(z^{[l]})$$

$$dW^{[l]} = dz^{[l]} \cdot a^{[l-1]T}$$

$$db^{[l]} = dz^{[l]}$$

$$da^{[l-1]} = W^{[l]T} \cdot dz^{[l]}$$

# *torch.autograd*

- *torch.autograd* is PyTorch's automatic differentiation engine that powers neural network training.
  - Specify ***requires\_grad=True*** for tensors that you want autograd to track operations on
  - When calling ***.backward()***, *autograd* calculates these gradients and stores them in the respective tensors' ***.grad*** attribute.

<https://colab.research.google.com/drive/1HlbodLDZiZYcG43BfMardWQfgKXRUStI?usp=sharing>

# Why do we need deep representation?

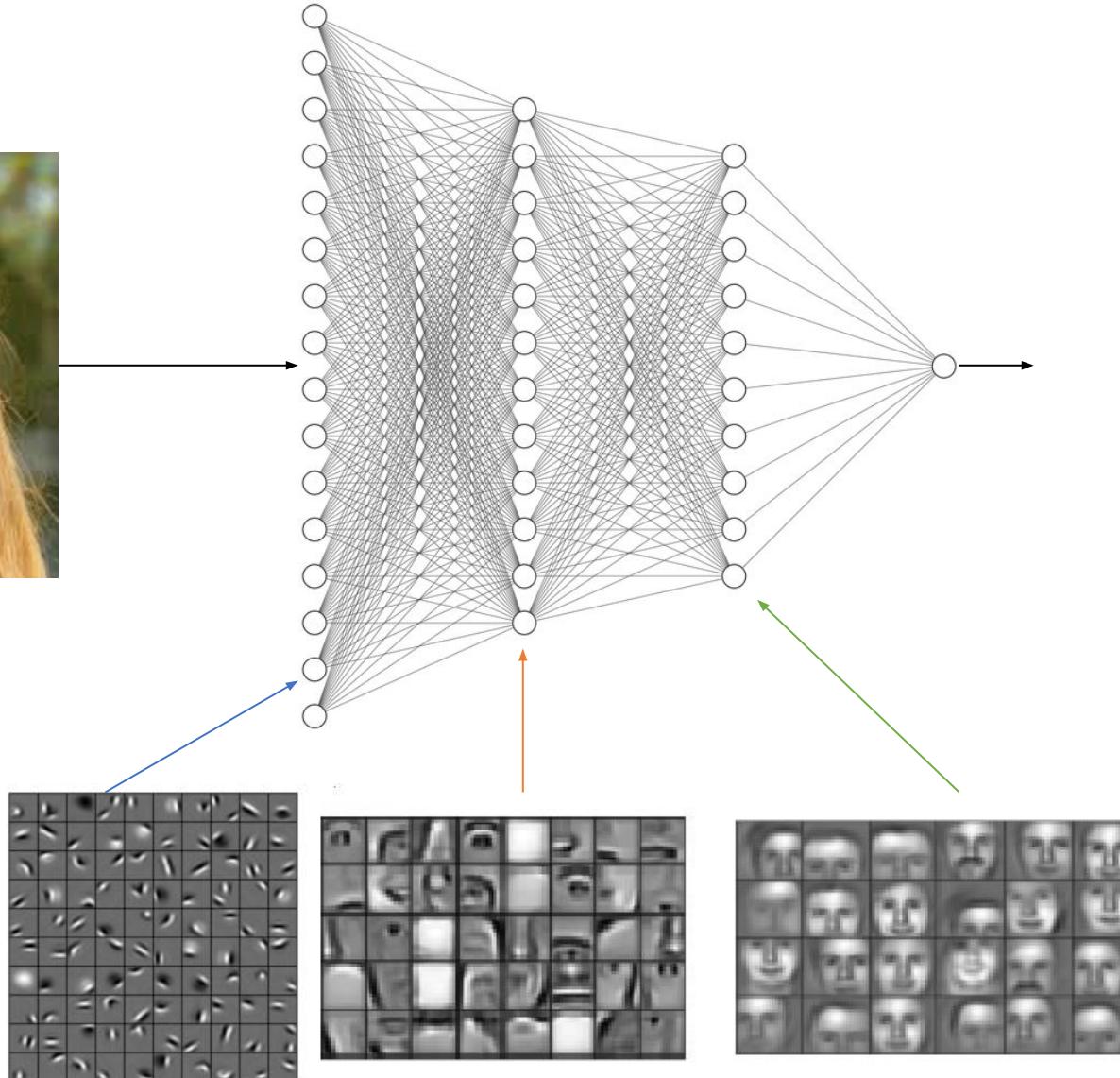


Image source: <https://developer.nvidia.com/blog/deep-learning-nutshell-core-concepts/#activation-function>

# Weights initialization

- Option 1
  - Initialize all weights to 0

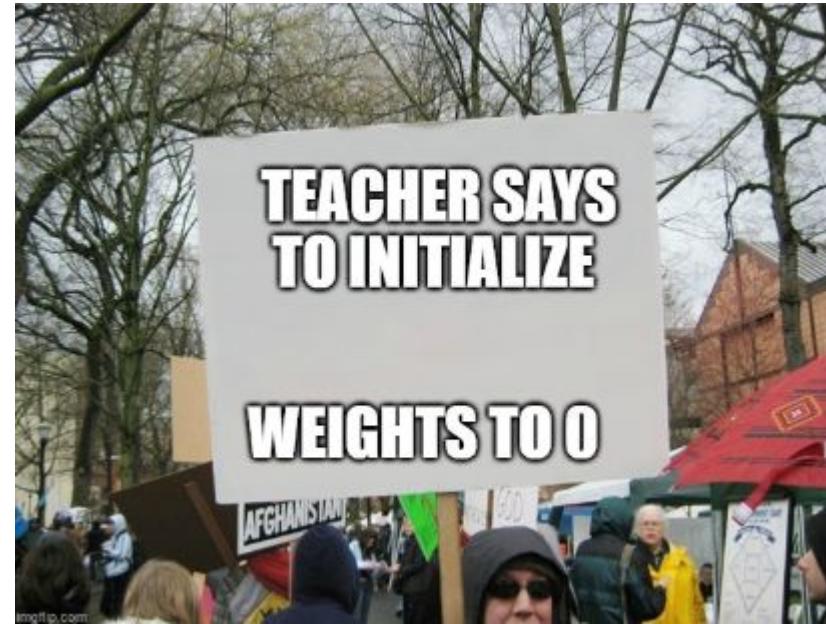


Image credit: A.S., UBB student 2020

# Weights initialization

- Option 1
  - ~~Initialize all weights to 0~~
  - All neurons will compute the same output (and implicitly they will have the same gradients during backprop → same parameter output)
  - No source of asymmetry
  - **Bad idea!**

# Weights initialization

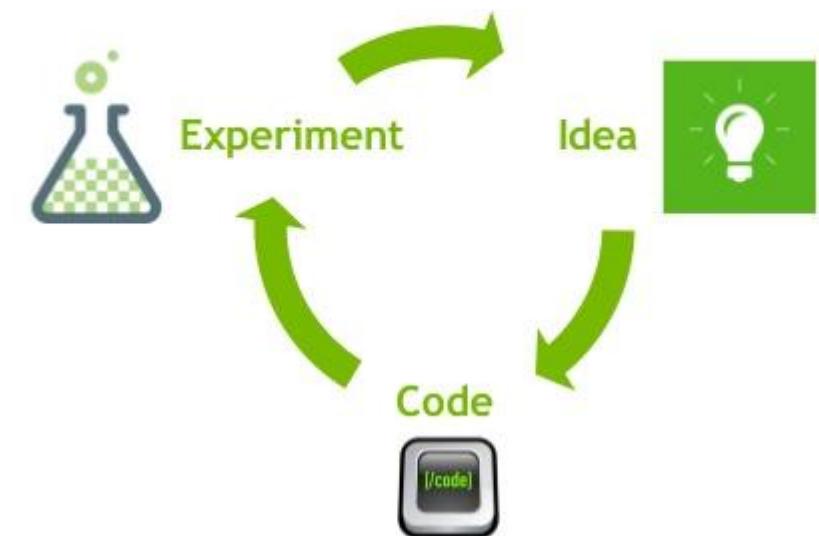
- Option 1: Initialize all weights to 0
- Option 2: Small random numbers
  - *Symmetry breaking*
  - Sample from a uniform distribution: `0.01*np.random.rand(D, H)`
  - Sample from a normal distribution: `0.01*np.random.randn(D, H)`

# Weights initialization

- Option 1: Initialize all weights to 0
- Option 2: Small random numbers
  - *Symmetry breaking*
  - Sample from a uniform distribution: `0.01*np.random.rand(D, H)`
  - Sample from a normal distribution: `0.01*np.random.randn(D, H)`
- Option 3: Sparse initialization
  - initialize weights to 0, but break symmetry: every neuron is randomly connected to a fixed number of neurons below it (with weights randomly sampled)

# Parameters vs Hyperparameters

- Parameters:  $W, b$
  - Hyper-parameters: control/ determine the values of the parameters we compute
    - Learning rate (for gradient descent)
    - Number of iterations for gradient descent
    - Number of hidden layers  $L$
    - Number of hidden units:  $n^{[1]}, n^{[2]}$
    - Activation functions: ReLu, Leaky ReLu, tanh, sigmoid
  - Deep learning is an empirical and iterative process
    - Idea
    - Code
    - Experiment
- REPEAT



# Data Preprocessing

- Mean subtraction
  - subtracting the mean across every individual *feature* in the data, and has the geometric interpretation of centering the cloud of data around the origin along every dimension
- Normalization
  - normalizing the data so that they are of approximately the same scale
  - divide each dimension by its standard deviation, once it has been zero-centered
  - normalize each dimension so that the min and max along the dimension is -1 and 1 respectively

**!!Any pre-processing statistics must only be computed on the training data, and then applied to the validation / test data !!**

# Playground

<https://playground.tensorflow.org/>

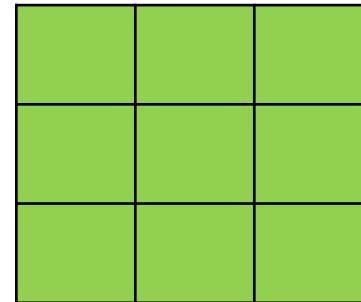
# Convolutions

# Image convolutions

$$g(i, j) = \sum_{m=1}^M \sum_{n=1}^N f(m, n)h(i-m, j-n)$$

10	20	30	15	12
11	200	34	23	45
32	35	255	255	10
10	33	6	7	59
13	43	13	45	3

-1	-1	-1
0	0	0
1	1	1



# Image convolutions

$$g(i, j) = \sum_{m=1}^M \sum_{n=1}^N f(m, n)h(i-m, j-n)$$

10	20	30	15	12
11	200	34	23	45
32	35	255	255	10
10	33	6	7	59
13	43	13	45	3

-1	-1	-1
0	0	0
1	1	1



262		

$$-10 - 20 - 30 + 32 + 35 + 255 = 262$$

# Image convolutions

$$g(i, j) = \sum_{m=1}^M \sum_{n=1}^N f(m, n)h(i-m, j-n)$$

10	20	30	15	12
11	200	34	23	45
32	35	255	255	10
10	33	6	7	59
13	43	13	45	3

-1	-1	-1
0	0	0
1	1	1



262	480	

$$-20 - 30 - 15 + 35 + 255 + 255 = 480$$

# Convolutional filters

What do you think is the effect of the following filters?

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

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

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$\frac{1}{273}$$

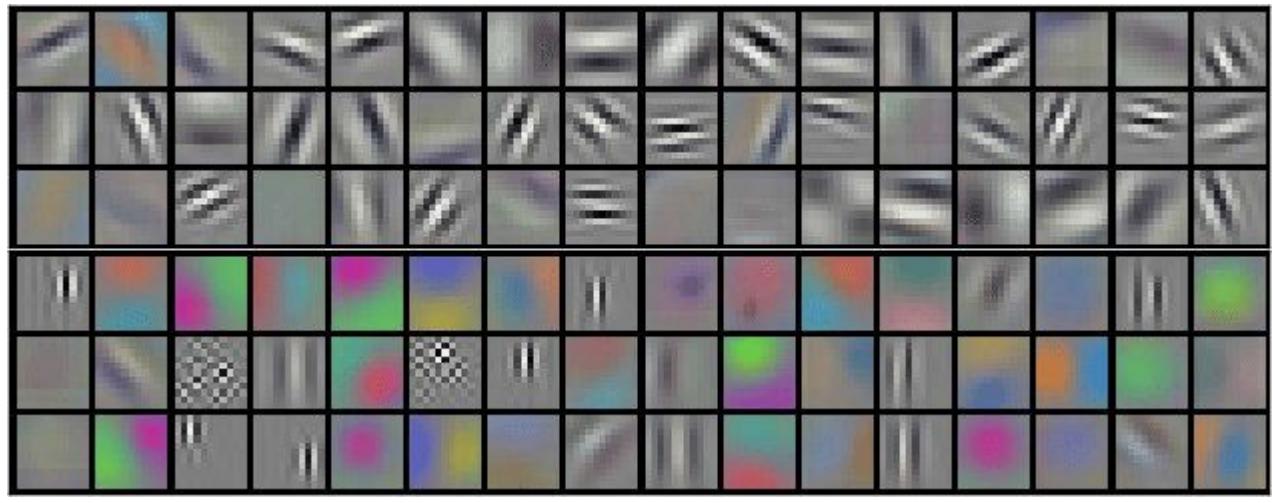
1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

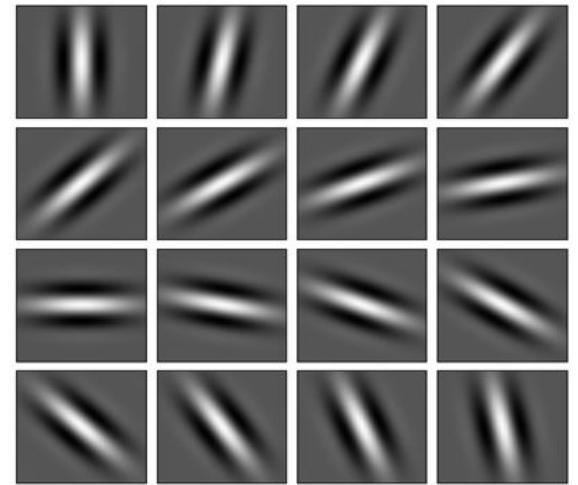
$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



Kernels learned from AlexNet first convolutional layers



Gabor filters

# What did we learn today?

- Backpropagation
  - Forward pass, backward pass
- Artificial neural networks
  - Input, output, hidden layers
  - Activation functions
- Image convolution (final step towards CNN)

# Recommended resources

- Course 1: deeplearning.ai Neural networks fundamentals
  - week 1, 2, 3, 4
- <http://cs231n.stanford.edu/>
- <https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b>
- <https://developer.nvidia.com/blog/deep-learning-nutshell-core-concepts/#activation-function>
- Back-propagation calculus explained:
  - [https://www.youtube.com/watch?v=IHZwWFHWa-w&t=347s&ab\\_channel=3Blue1Brown](https://www.youtube.com/watch?v=IHZwWFHWa-w&t=347s&ab_channel=3Blue1Brown)
  - [https://www.youtube.com/watch?v=llg3gGewQ5U&ab\\_channel=3Blue1Brown](https://www.youtube.com/watch?v=llg3gGewQ5U&ab_channel=3Blue1Brown)
  - [https://www.youtube.com/watch?v=tIeHLnjs5U8&ab\\_channel=3Blue1Brown](https://www.youtube.com/watch?v=tIeHLnjs5U8&ab_channel=3Blue1Brown)

# Computer Vision and Deep Learning

Lecture 4

# Recap

- Backpropagation
  - Forward pass, backward pass
- Artificial neural networks
  - Input, output, hidden layers
  - Activation functions

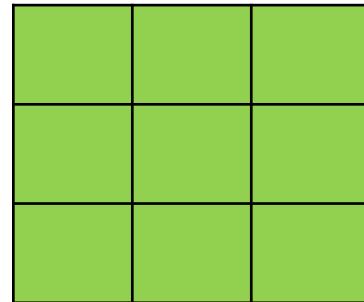
# Convolutions

# Image convolutions

$$g(i, j) = \sum_{m=1}^M \sum_{n=1}^N f(m, n)h(i-m, j-n)$$

10	20	30	15	12
11	200	34	23	45
32	35	255	255	10
10	33	6	7	59
13	43	13	45	3

-1	-1	-1
0	0	0
1	1	1



# Image convolutions

$$g(i, j) = \sum_{m=1}^M \sum_{n=1}^N f(m, n)h(i-m, j-n)$$

10	20	30	15	12
11	200	34	23	45
32	35	255	255	10
10	33	6	7	59
13	43	13	45	3

-1	-1	-1
0	0	0
1	1	1



262		

$$-10 - 20 - 30 + 32 + 35 + 255 = 262$$

# Image convolutions

$$g(i, j) = \sum_{m=1}^M \sum_{n=1}^N f(m, n)h(i-m, j-n)$$

10	20	30	15	12
11	200	34	23	45
32	35	255	255	10
10	33	6	7	59
13	43	13	45	3

-1	-1	-1
0	0	0
1	1	1



262	480	

$$-20 - 30 - 15 + 35 + 255 + 255 = 480$$

# Convolutional filters

What do you think is the effect of the following filters?

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

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

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$\frac{1}{273}$$

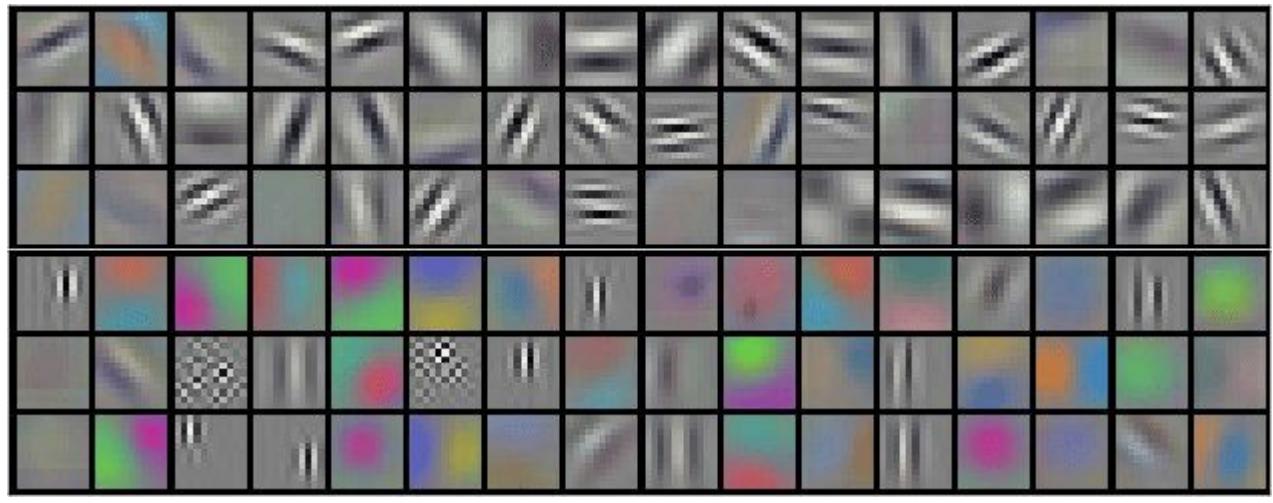
1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

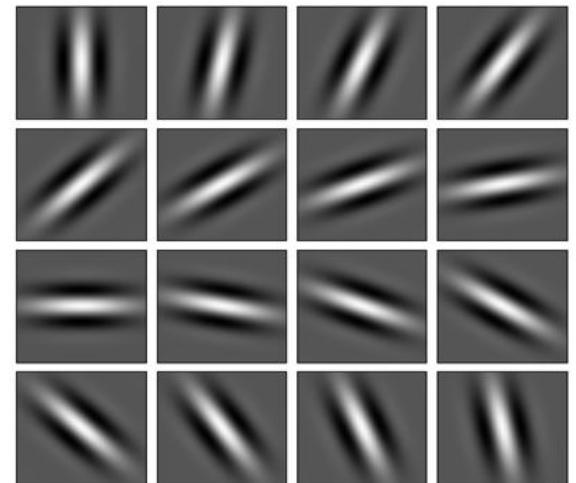
$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



Kernels learned from AlexNet first convolutional layers



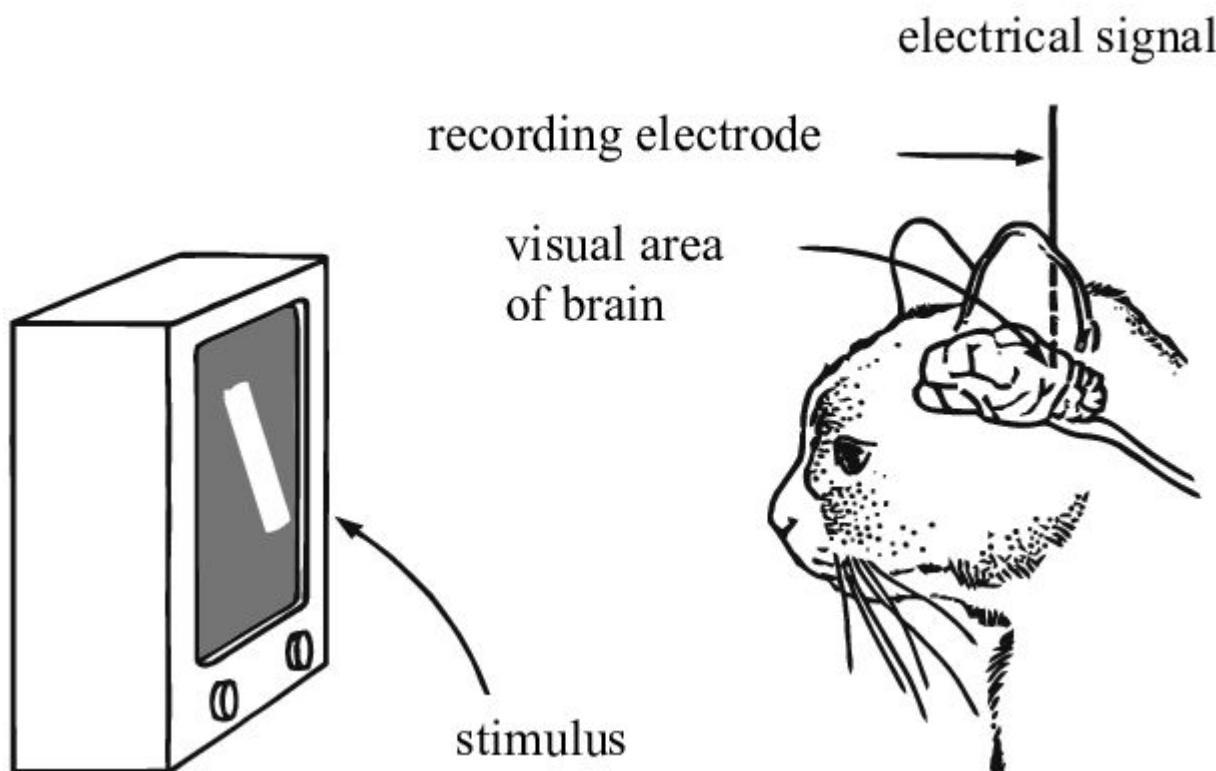
Gabor filters

# Today's agenda

- Short history of neural networks
- Convolutional neural networks
- How to evaluate a classifier

# Convolutional neural networks

# Understanding the visual cortex



Hubel and Wiesel, 1959

[https://www.youtube.com/watch?v=IOHayh06LJ4&ab\\_channel=PaulLester](https://www.youtube.com/watch?v=IOHayh06LJ4&ab_channel=PaulLester)



Nobel Prize for Physiology or Medicine in 1981:

David Hubel and Torsten Wiesel

**Simple cells:**

orientation, position

**Complex cells:**

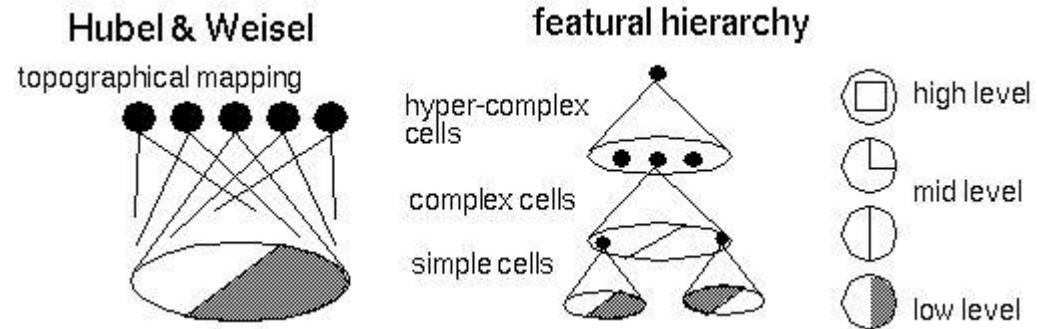
orientation, motion, direction

**"Hypercomplex" cells:**

orientation, motion, direction, length

# Understanding the visual cortex

- Nearby cells in the cortex represented and processed nearby regions in the visual field
- Hubel and Wiesel hypothesized that the visual cortex can be described by a hierarchical organization of simple cells that fed into complex cells which have more complicated activations and can form higher level representations



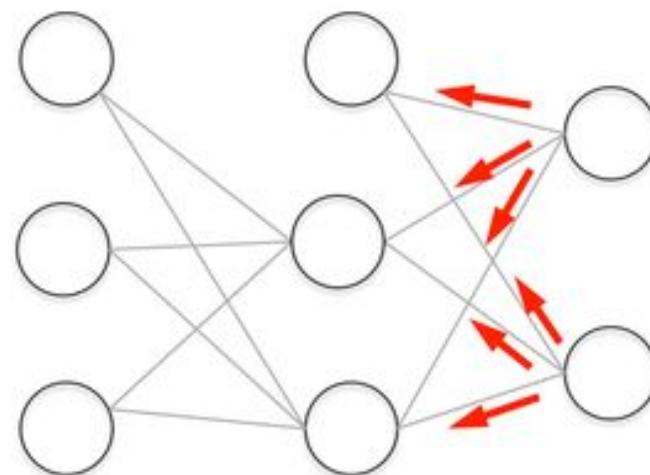
# Backpropagation, 1986

## Learning representations by back-propagating errors

David E. Rumelhart\*, Geoffrey E. Hinton†  
& Ronald J. Williams\*

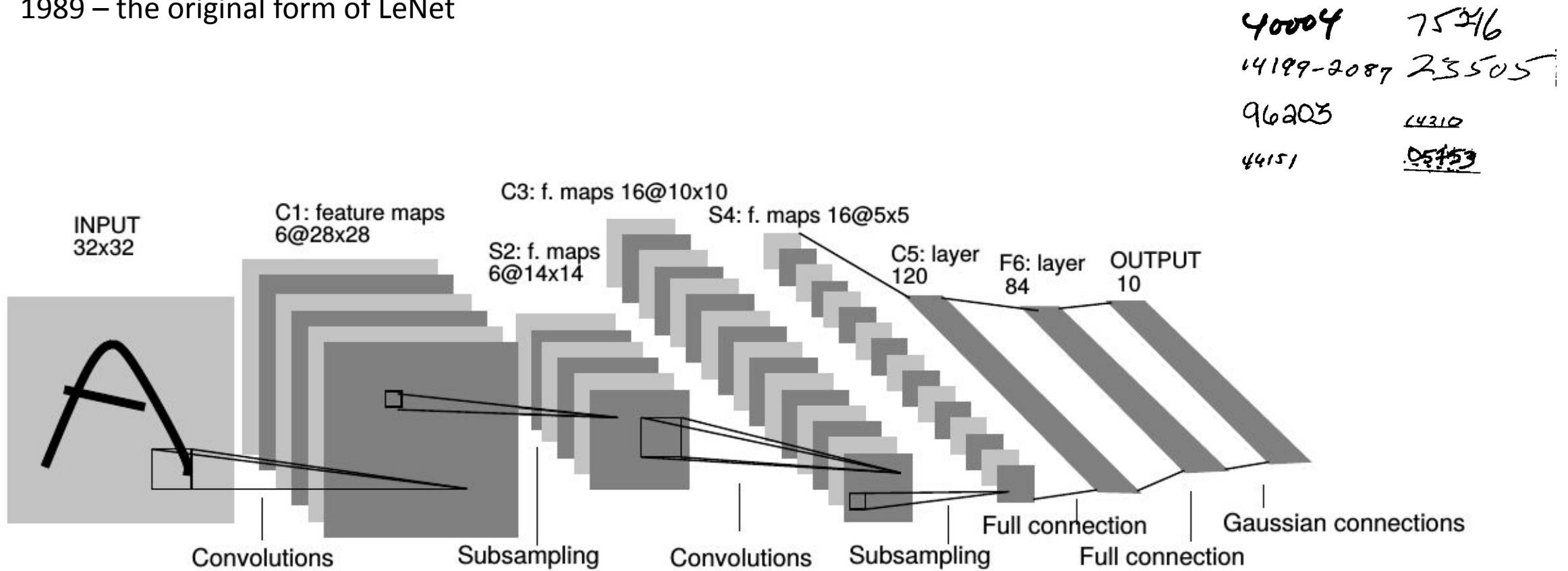
\* Institute for Cognitive Science, C-015, University of California,  
San Diego, La Jolla, California 92093, USA

† Department of Computer Science, Carnegie-Mellon University,  
Pittsburgh, Philadelphia 15213, USA



# Gradient-Based Learning Applied to Document Recognition, Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner, 1998

1989 – the original form of LeNet



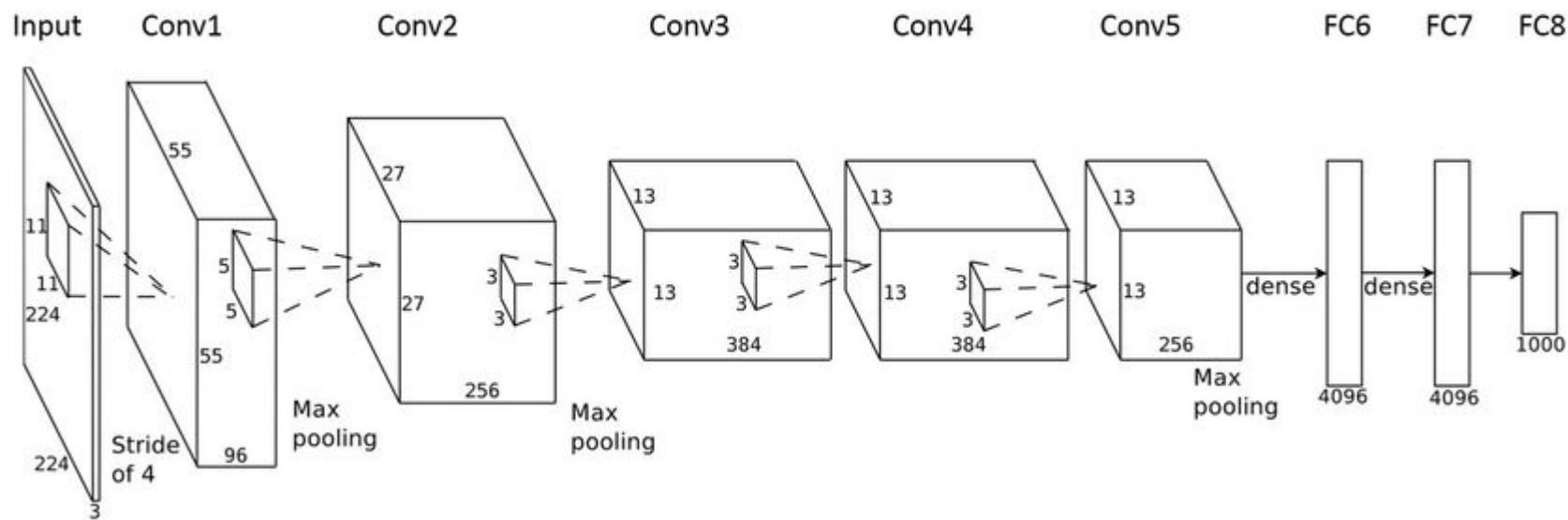
# Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition

*Dan Ciresan, 2010*

- One of the very first implementations of GPU Neural nets
  - forward and backward pass implemented of an artificial neural network (up to 9 layers) implemented on an NVIDIA GTX 280 graphic processor

2012 ImageNet Classification with Deep Convolutional

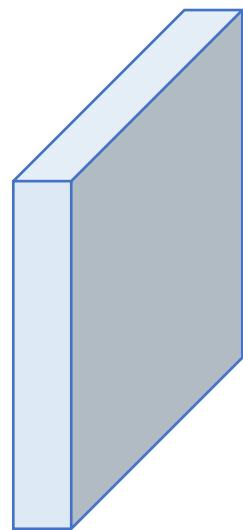
Neural Networks, *Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton*



# Convolutional neural networks

# Convolutional layers

- Preserve the spatial information
- Convolve the filter over the entire input volume
  - The filter has the **same depth** as the input



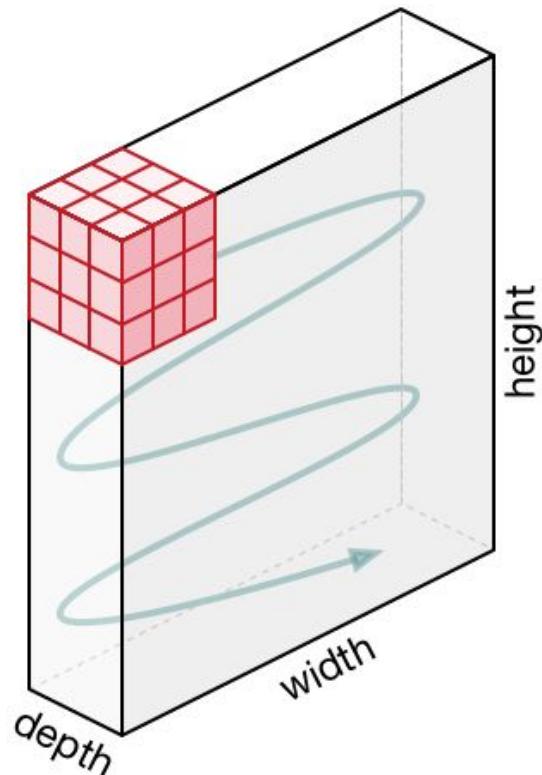
Input volume  
 $32 \times 32 \times 3$



filter  
 $5 \times 5 \times 3$

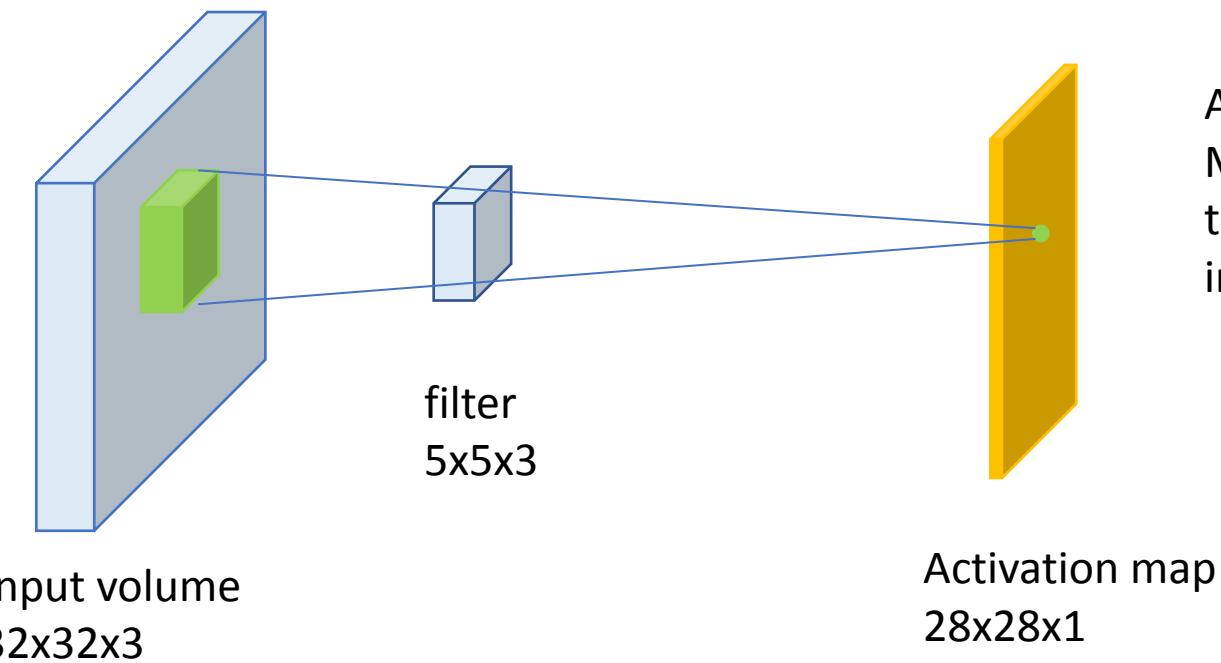


Activation map  
 $28 \times 28 \times 1$



# Convolutional layers

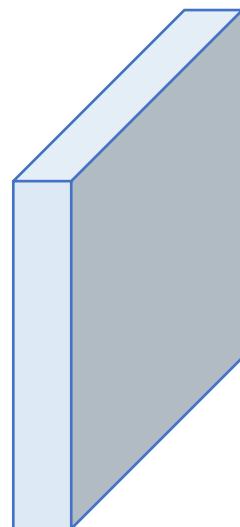
- Preserve the spatial information
- Convolve the filter over the entire input volume
  - The filter has the **same depth** as the input



At each position in the input volume:  
Multiply (element-wise, across all channels)  
the filter and a small patch ( $5 \times 5 \times 3$ ) in this  
input volume and add a bias term

# Convolutional layers

- Preserve the spatial information
- Convolve the filter over the entire input volume
  - The filter has the **same depth** as the input



Input volume  
32x32x3



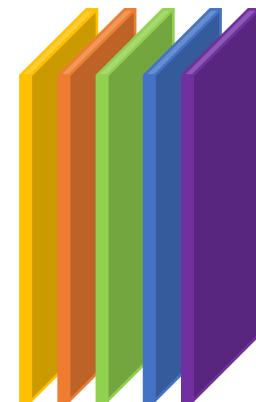
$k$  filters



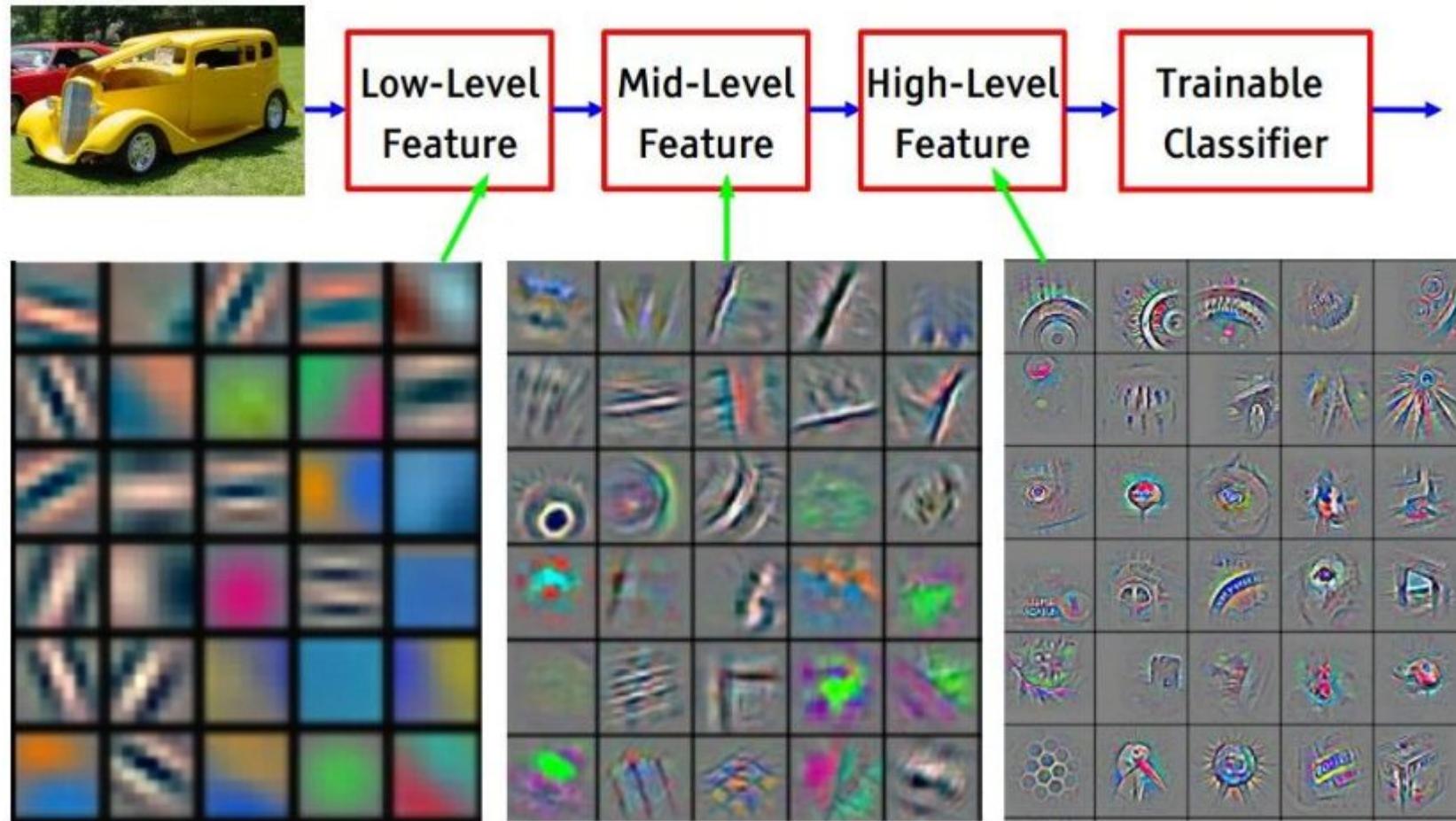
$k$  activation maps  
28x28x1

# Convolutional layers

- Neurons in an activation map:
  - Each neuron is connected to a small region in the input
  - All of neurons in the activation map share parameters
- Receptive field of a neuron
- $k$  filters →  $k$  different neurons all looking at the same region in the input volume



$k$  activation  
maps



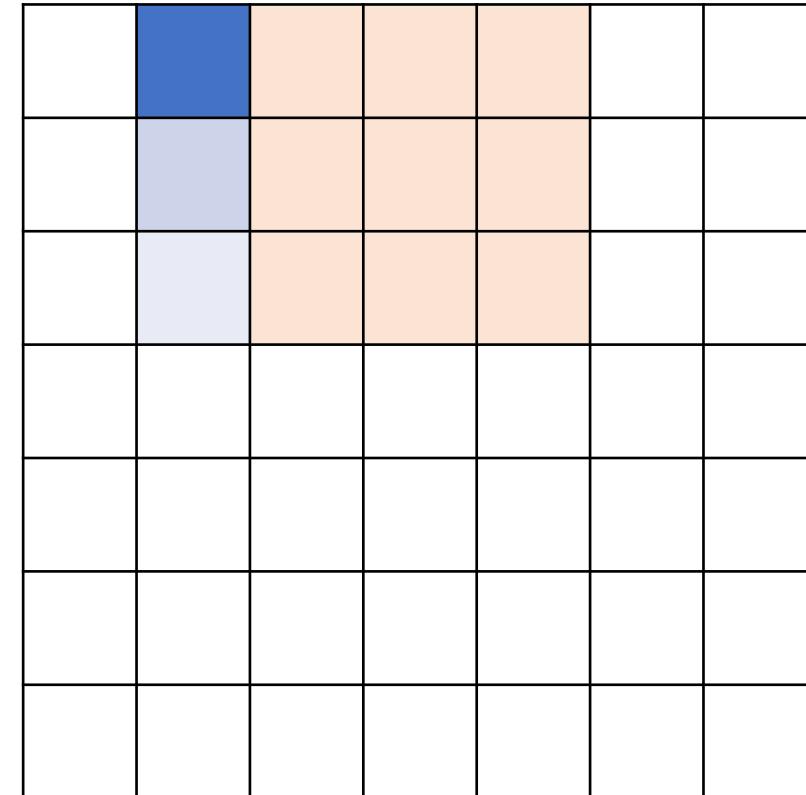
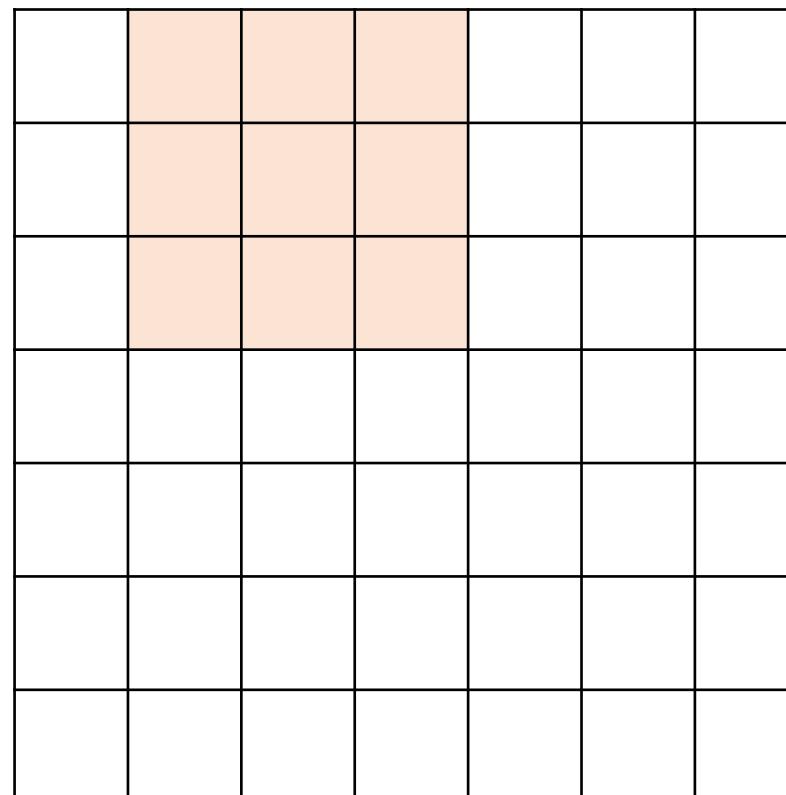
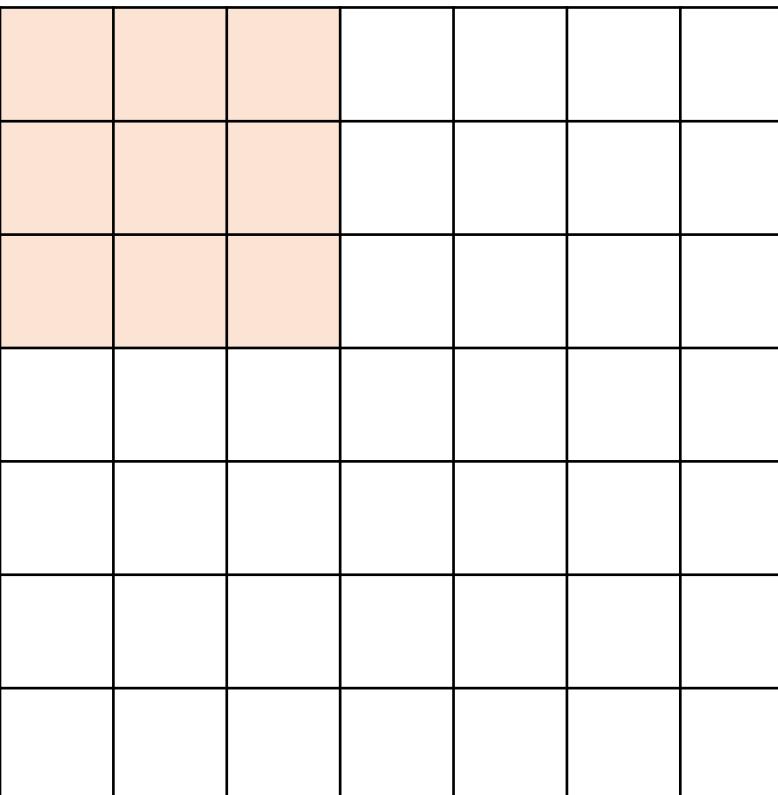
[https://medium.com/@chriskevin\\_80184/feature-maps-ee8e11a71f9e](https://medium.com/@chriskevin_80184/feature-maps-ee8e11a71f9e)

<https://www.youtube.com/watch?v=AgkfIQ4IGaM>

# Convolutional layers

## Stride

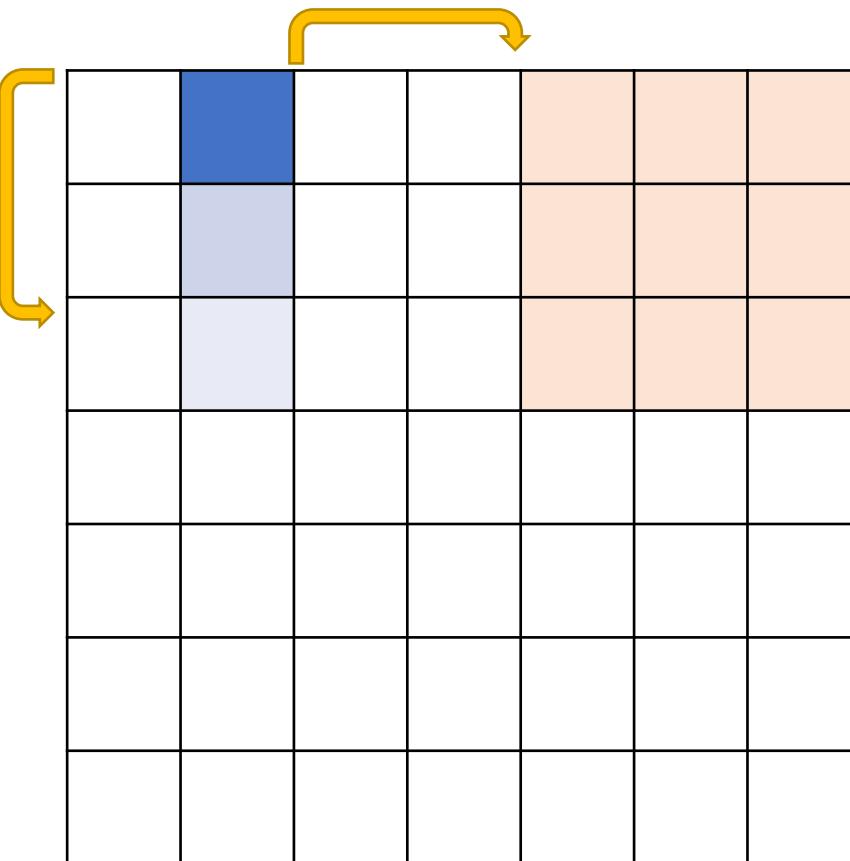
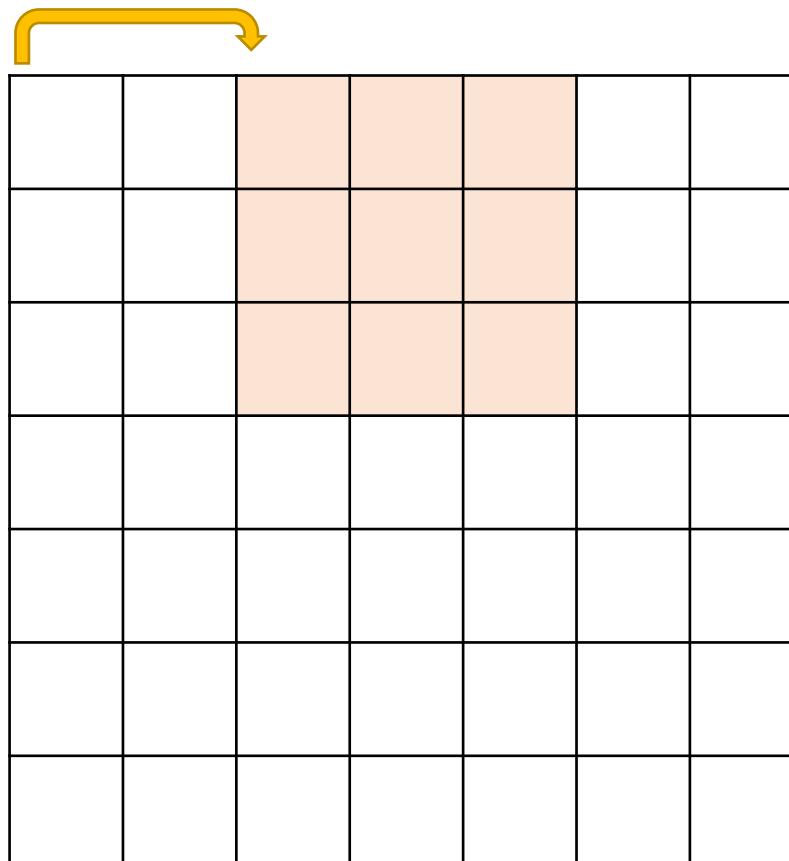
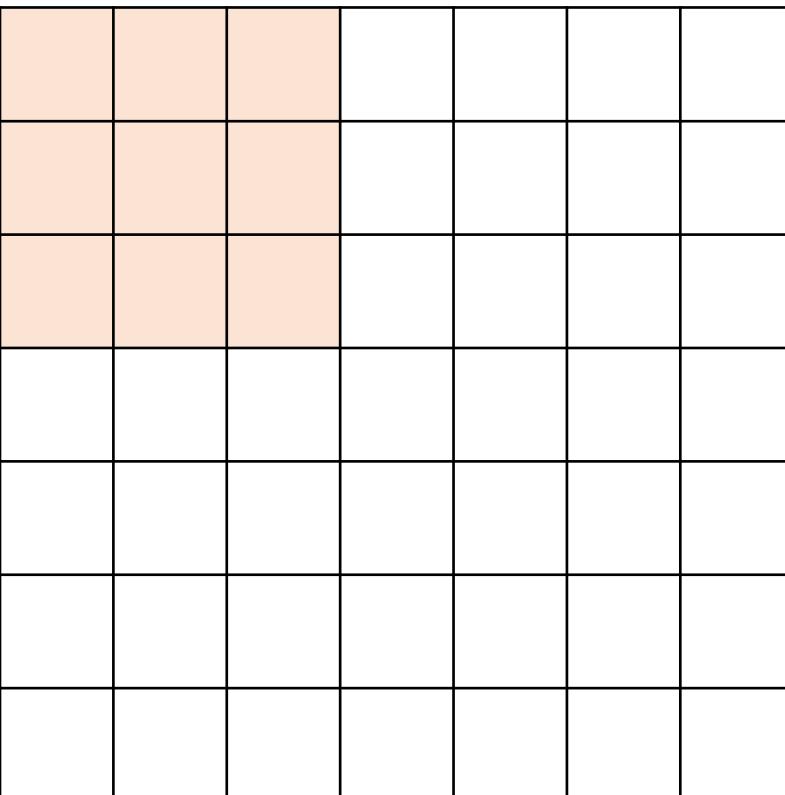
- Stride – the amount by which the filter shifts
- Stride 1



# Convolutional layers

## Stride

- Stride – the amount by which the filter shifts
- Stride 2



# Convolutional layers

## Padding

0	0	0	0	0	0	0
0						0
0						0
0						0
0						0
0						0
0	0	0	0	0	0	0

padding: 1

Convolve an input of spatial size 7x7 with a 3x3 filter : output spatial size?

Convolve an input of spatial size 7x7 with a 3x3 filter, and applying 0 padding to the input: output spatial size?

To preserve size spatially:  
CONV layers with stride 1, filters of size FxF,  
and zero-padding with  $(F-1)/2$

# Why convolutions

- Parameter sharing
  - A feature detector that's useful in one part of an image is probably useful in other parts of the image
  - Translation invariance
- Sparsity of connections
  - The output volume depends only on a small (filter size) subset of the input volume

# Convolutional layers

( $W_I, H_I, D_I$ ) ( $W_O, H_O, D_O$ )

$D_I$  – filter depth

$W_I$  – input width,  $W_O$  – output width

$H_I$  – input height,  $H_O$  – input height

F – filter size

P – padding

S - stride

$$W_O = \frac{W_I - F + 2P}{S} + 1$$

$$H_O = \frac{H_I - F + 2P}{S} + 1$$

## CONV2D ⚡

---

```
CLASS torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1,
groups=1, bias=True, padding_mode='zeros', device=None, dtype=None) [SOURCE]
```

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size  $(N, C_{\text{in}}, H, W)$  and output  $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$  can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where  $\star$  is the valid 2D [cross-correlation](#) operator,  $N$  is a batch size,  $C$  denotes a number of channels,  $H$  is a height of input planes in pixels, and  $W$  is width in pixels.

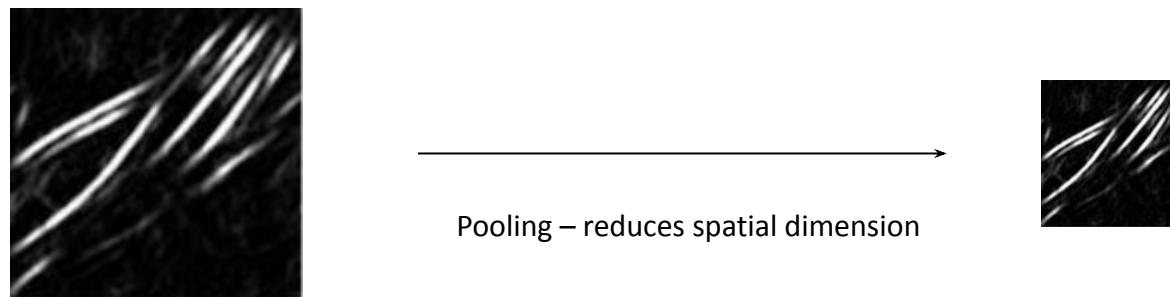
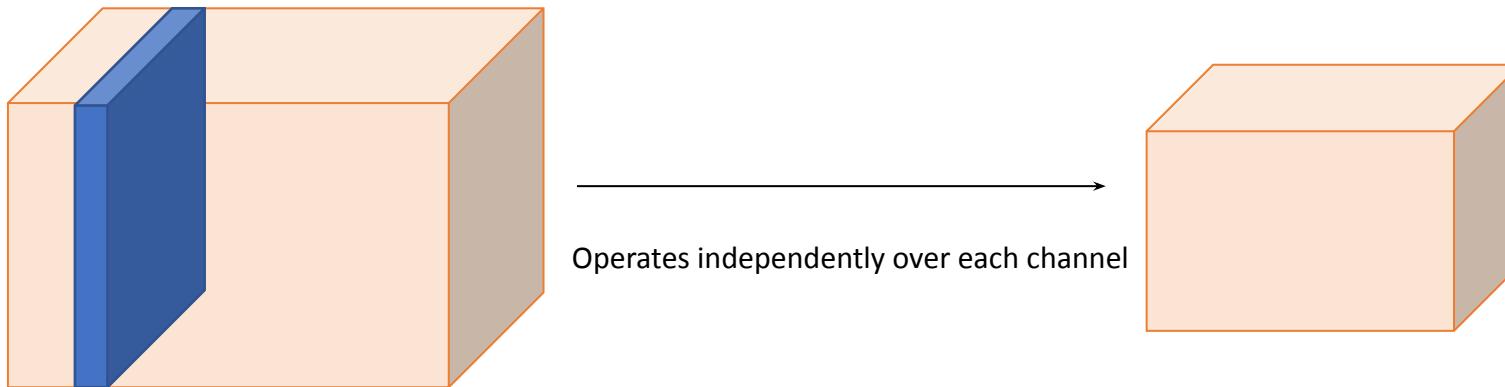
This module supports [TensorFloat32](#).

Padding: “same” or “valid”

# Pooling layers

- Operate **independently** over each channel in the input
- Reduces the input size, creating smaller (and more manageable) representations
- Common pooling layers:
  - max pooling: takes the maximum value within each “patch” in the feature map
  - average pooling: takes the average value of each “patch” in the feature map
- **It does not contain any learnable weights**

# Pooling layers



# Pooling layers

## Examples



Max pooling



Avg pooling

# Pooling layers

Example: max pooling layer, F 2, stride 1

1	3	6	6
20	9	8	4
2	1	4	5
1	12	13	10

20	9	8
20	9	8
12	13	13

# Pooling layers

Example: max pooling layer, F 2, stride 2

1	3	6	6
20	9	8	4
2	1	4	5
1	12	13	10

20	8
12	13

# Pooling layer

- Parameters
  - Filter size (spatial extent): F
  - Stride: S
- Input:  $W_I \times H_I \times D$
- Output:  $W_O \times H_O \times D$
- It has no learnable parameters

$$W_O = \frac{W_I - F}{S} + 1$$

$$H_O = \frac{H_I - F}{S} + 1$$

---

```
CLASS torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False,  
    ceil_mode=False) [SOURCE]
```

Applies a 2D max pooling over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size  $(N, C, H, W)$ , output  $(N, C, H_{out}, W_{out})$  and kernel\_size  $(kH, kW)$  can be precisely described as:

$$out(N_i, C_j, h, w) = \max_{m=0, \dots, kH-1} \max_{n=0, \dots, kW-1} \\ \text{input}(N_i, C_j, \text{stride}[0] \times h + m, \text{stride}[1] \times w + n)$$

If padding is non-zero, then the input is implicitly padded with negative infinity on both sides for padding number of points. dilation controls the spacing between the kernel points. It is harder to describe, but this [link](#) has a nice visualization of what dilation does.

```
CLASS torch.nn.AvgPool2d(kernel_size, stride=None, padding=0, ceil_mode=False,  
    count_include_pad=True, divisor_override=None) [SOURCE]
```



Applies a 2D average pooling over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size  $(N, C, H, W)$ , output  $(N, C, H_{out}, W_{out})$  and *kernel\_size*  $(kH, kW)$  can be precisely described as:

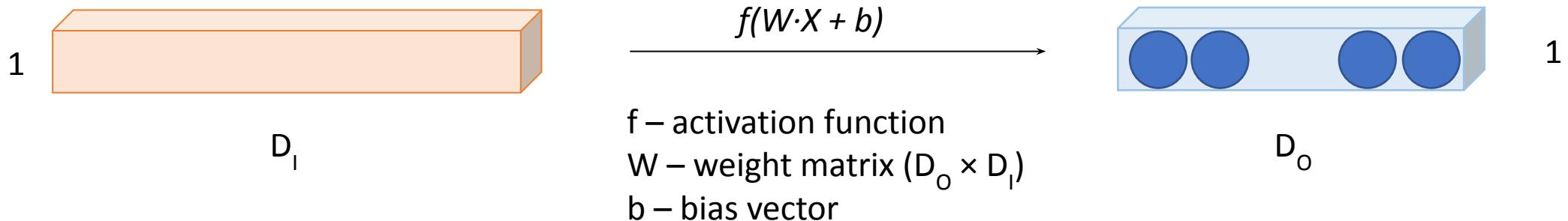
$$out(N_i, C_j, h, w) = \frac{1}{kH * kW} \sum_{m=0}^{kH-1} \sum_{n=0}^{kW-1} input(N_i, C_j, stride[0] \times h + m, stride[1] \times w + n)$$



If padding is non-zero, then the input is implicitly zero-padded on both sides for padding number of points.

# Fully connected layers

- They don't preserve spatial information
  - Linear unit followed by a linearity
- Just in regular NN, contain several neurons that are connected to the entire input volume
  - Each neuron “sees” the entire input volume



# LINEAR

---

CLASS `torch.nn.Linear(in_features, out_features, bias=True, device=None, dtype=None)` [\[SOURCE\]](#)

Applies a linear transformation to the incoming data:  $y = xA^T + b$

This module supports [TensorFloat32](#).

On certain ROCm devices, when using float16 inputs this module will use [different precision](#) for backward.

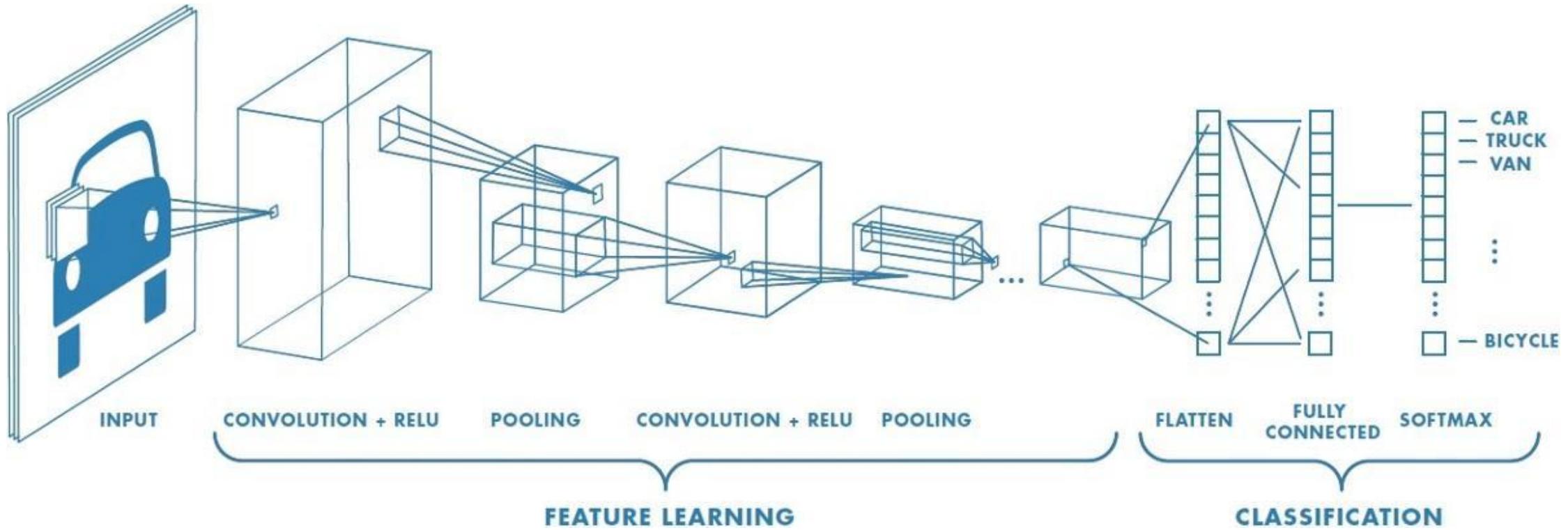
---

## Parameters:

- **`in_features`** ([int](#)) – size of each input sample
- **`out_features`** ([int](#)) – size of each output sample
- **`bias`** ([bool](#)) – If set to `False`, the layer will not learn an additive bias. Default: `True`

- If we have an input volume of  $27 \times 27 \times 5$ , what will be size of this volume if we apply a padding of 2?
- How many parameters (including the bias) does a convolutional layer with 10 filters of size  $5 \times 5$  have?
- How many parameters (including the bias) does a convolutional layer with 10 filters of size  $5 \times 5$  have if the input size is  $32 \times 32 \times 3$ ? What if we use a stride of 2?
- Given an input volume that is  $63 \times 63 \times 16$  that is convolved with 32 filters that are each  $7 \times 7$ , using a stride of 2 and no padding. What is the output volume?
- Given a RGB image of size  $300 \times 300$ , and you use a classical neural network with the first hidden layer of 100 neurons (each one fully connected to the input). How many parameters does this hidden layer have?

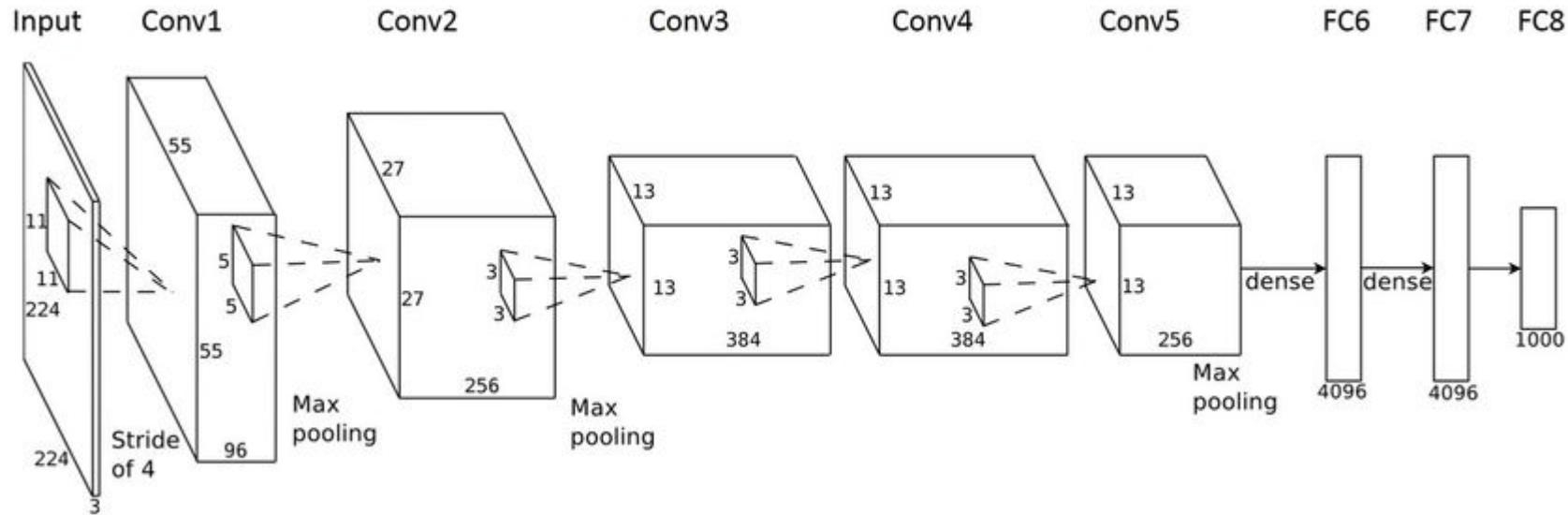
# Typical neural network architecture



# Typical neural network architecture

- Several CONV, POOL and FC layers stacked together
  - [CONV-ReLU-POOL]\*N – [FC]\*K – softmax
  - Recent neural networks change this paradigm
- The trend is to reduce the filter sizes and to increase the depth of the networks
- Another trend is to avoid using POOL and FC layers, and use only CONV layers

# Alexnet example



AlexNet Network - Structural Details													
Input		Output		Layer	Stride	Pad	Kernel size	in	out	# of Param			
227	227	3	55	55	96	conv1	4	0	11	11	3	96	34944
55	55	96	27	27	96	maxpool1	2	0	3	3	96	96	0
27	27	96	27	27	256	conv2	1	2	5	5	96	256	614656
27	27	256	13	13	256	maxpool2	2	0	3	3	256	256	0
13	13	256	13	13	384	conv3	1	1	3	3	256	384	885120
13	13	384	13	13	384	conv4	1	1	3	3	384	384	1327488
13	13	384	13	13	256	conv5	1	1	3	3	384	256	884992
13	13	256	6	6	256	maxpool5	2	0	3	3	256	256	0
					fc6			1	1	9216	4096	37752832	
					fc7			1	1	4096	4096	16781312	
					fc8			1	1	4096	1000	4097000	
<b>Total</b>										<b>62,378,344</b>			

“AlexNet input starts with 227 by 227 by 3 images. And if you read the paper, the paper refers to 224 by 224 by 3 images. But if you look at the numbers, I think that the numbers make sense only of actually 227 by 227.”

# Training a neural network

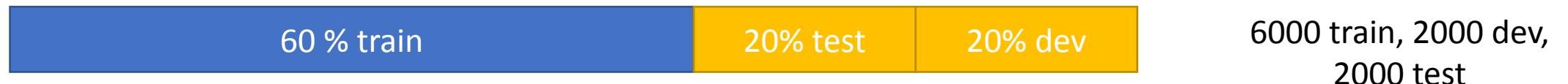
# Train, dev and test sets

- Training set
  - Used to train the model, determines what the network learns
- Development (*dev*) set or validation set
  - Used to evaluate the performance of your models and determine which ones work best
- Test set
  - Used to get an unbiased estimate of the final performance of the model

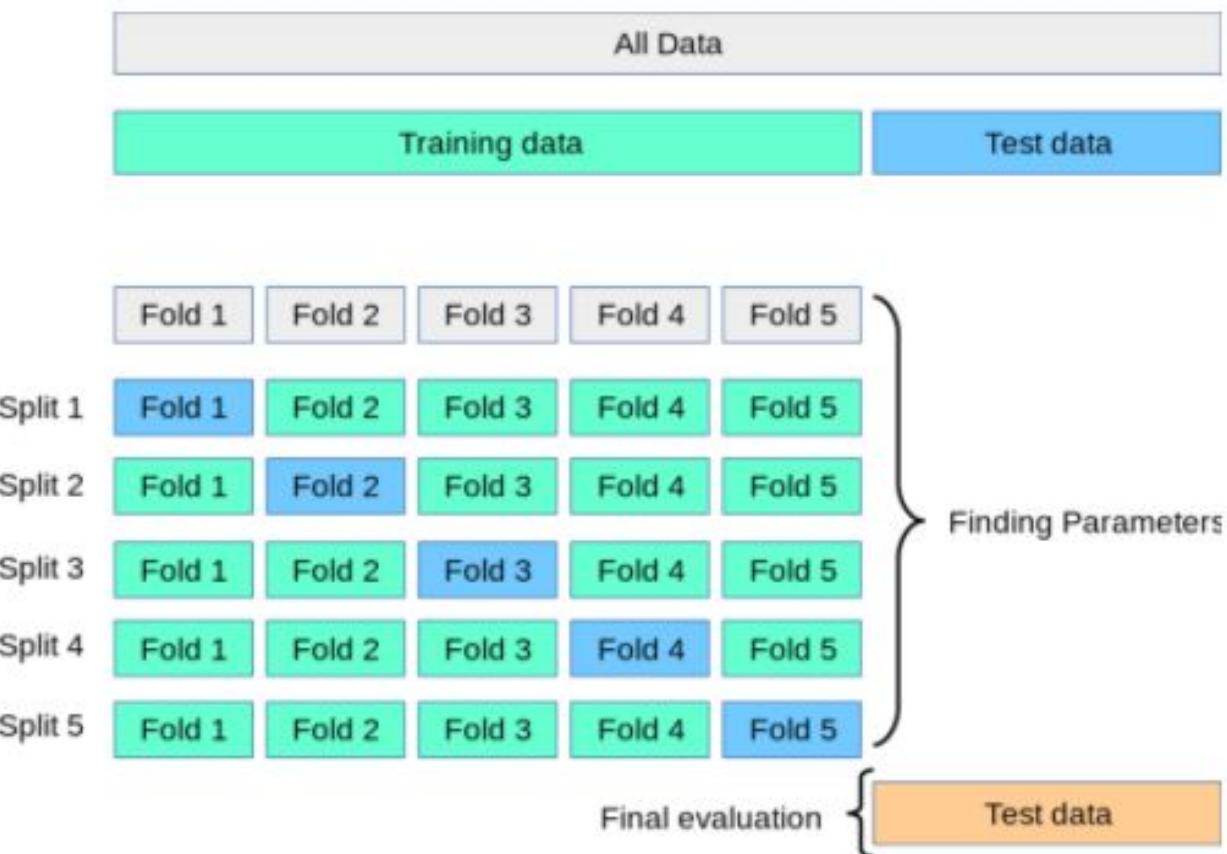
Sometimes it might be ok to not have a test set (only train and dev sets)

# How to split your data into train/dev/test set?

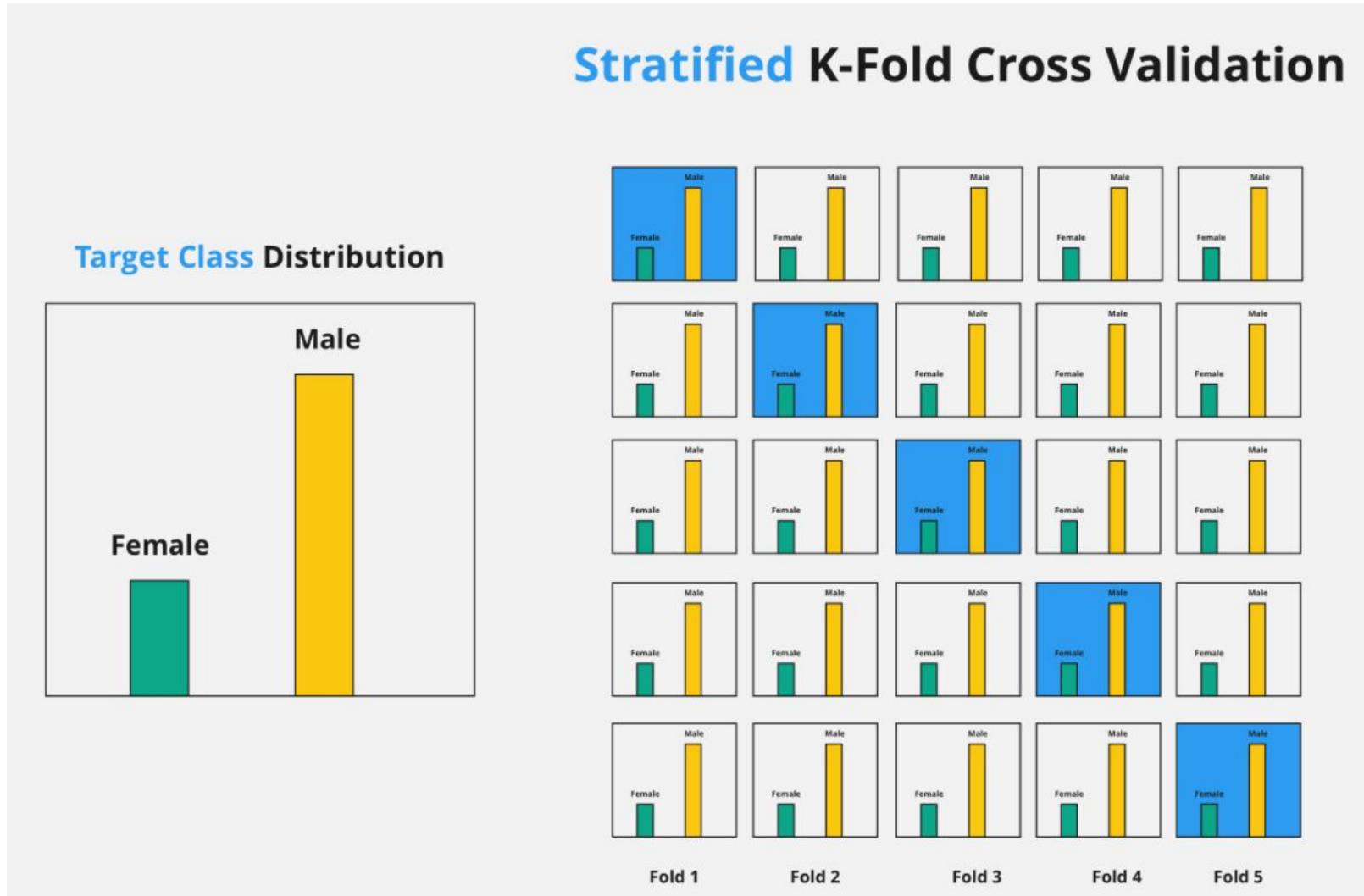
- Before deep learning, when available data was relatively limited (e.g. 10000 images)



# K-fold validation



# Stratified k-fold validation



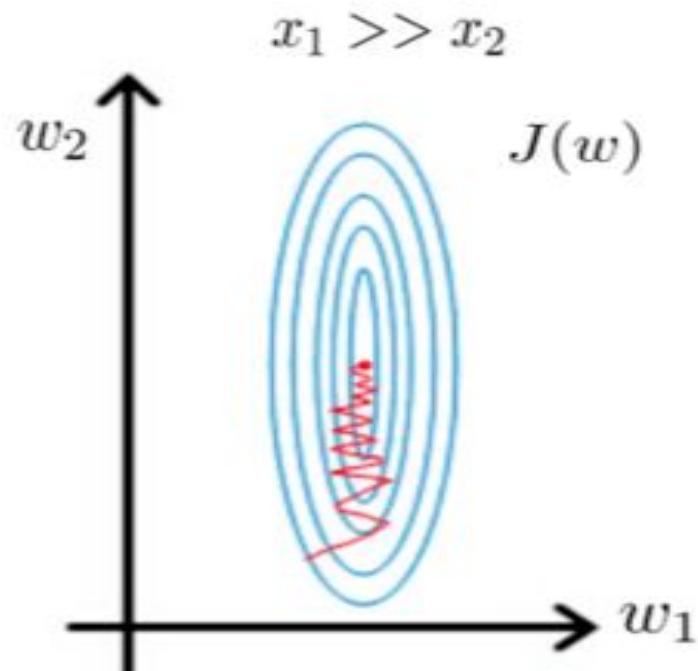
# How to split your data into train/dev/test set?

- Large scale datasets
  - ImageNet > 14 million images
  - VGGFace 2 > 3.3 million images
  - JFT > 300 million images
- E.g. 1 million images
  - 980000 train set, 10000 dev set, 10000 test set -> 98% train, 1% dev, 1% test
- E.g. 4 million images
  - 3980000 train set, 10000 dev set, 10000 test set -> 99.5% train, 0.25% dev, 0.25% test

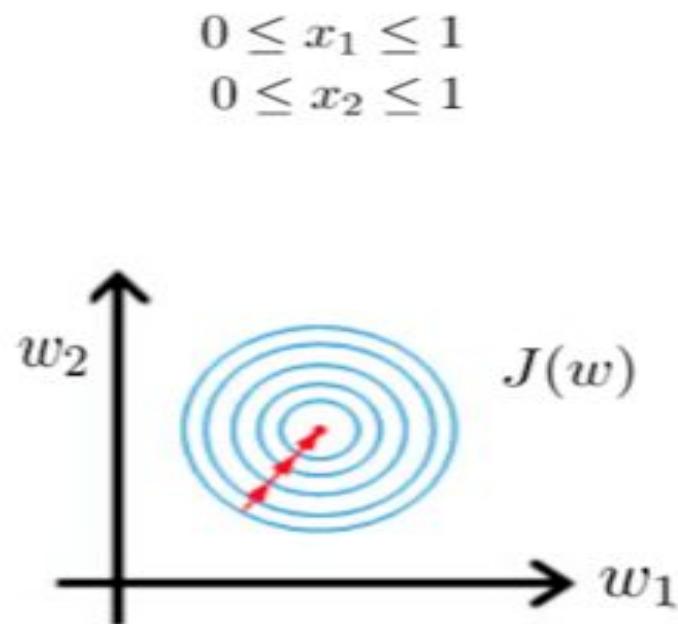
# Data pre-processing

# Importance of feature scaling

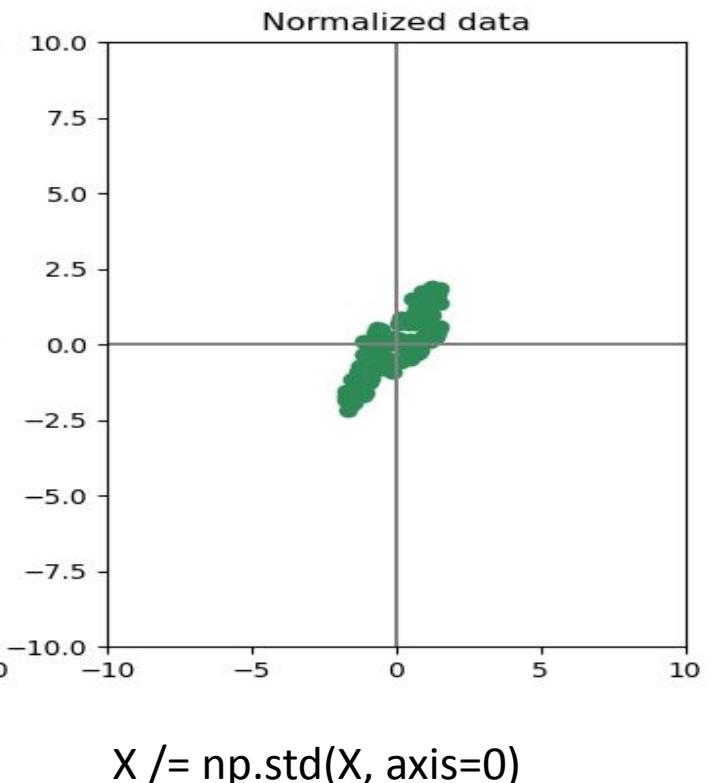
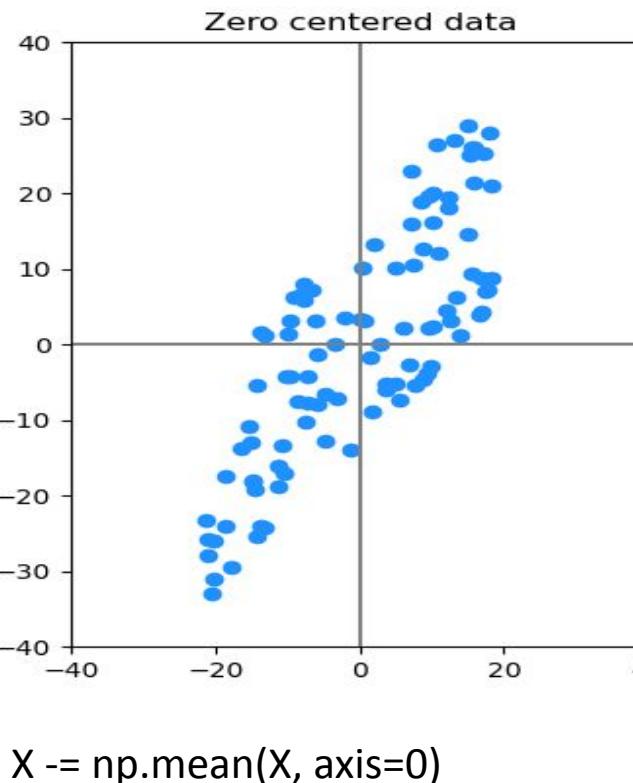
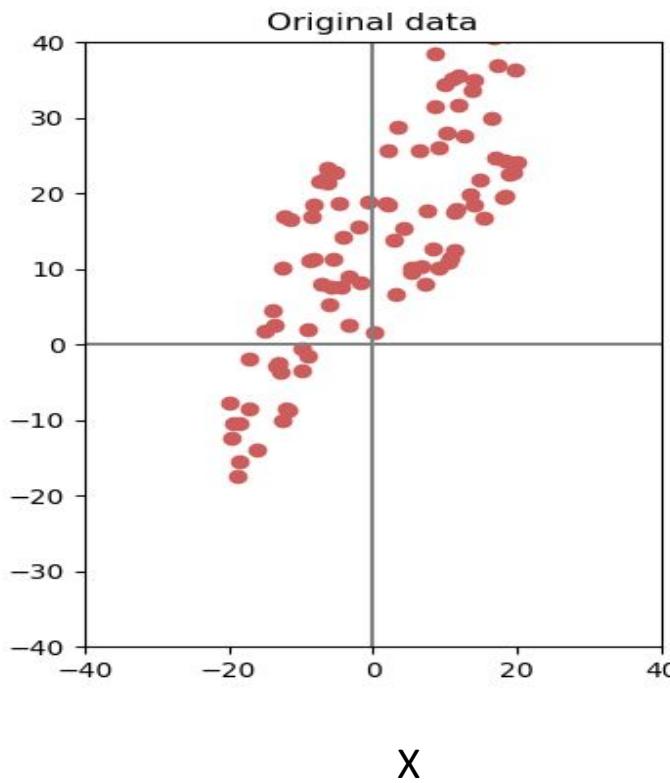
Gradient descent  
without scaling



Gradient descent  
after scaling variables



# Mean subtraction, scaling



# Pre-processing for Images

- Zero center: subtract the mean across every individual feature in the data
  - Mean image
  - Mean across each channel
- Optional: normalize the data such that dimensions are approximately the same scale

[https://github.com/keras-team/keras-applications/blob/master/keras\\_applications/imagenet\\_utils.py](https://github.com/keras-team/keras-applications/blob/master/keras_applications/imagenet_utils.py)

# Training a neural network

*Weights initialization*

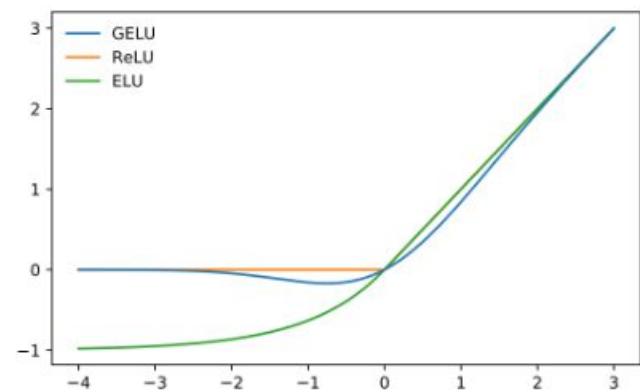
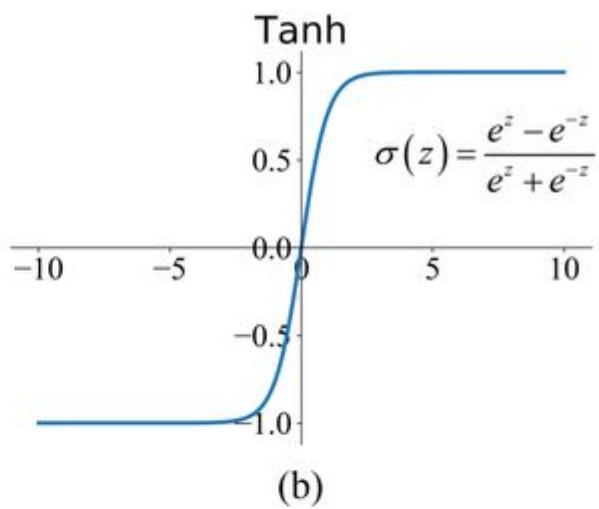
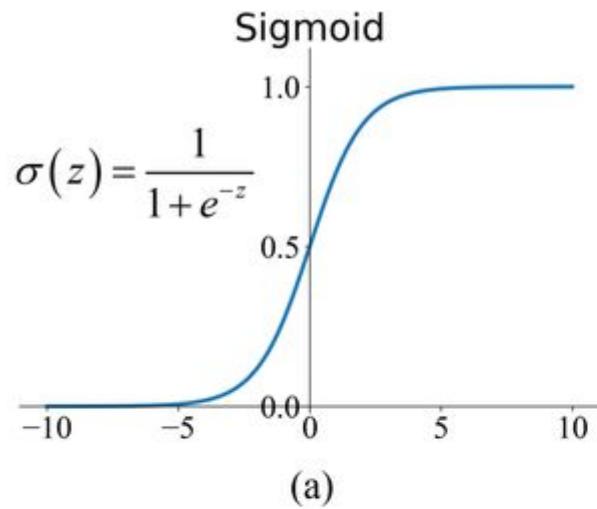
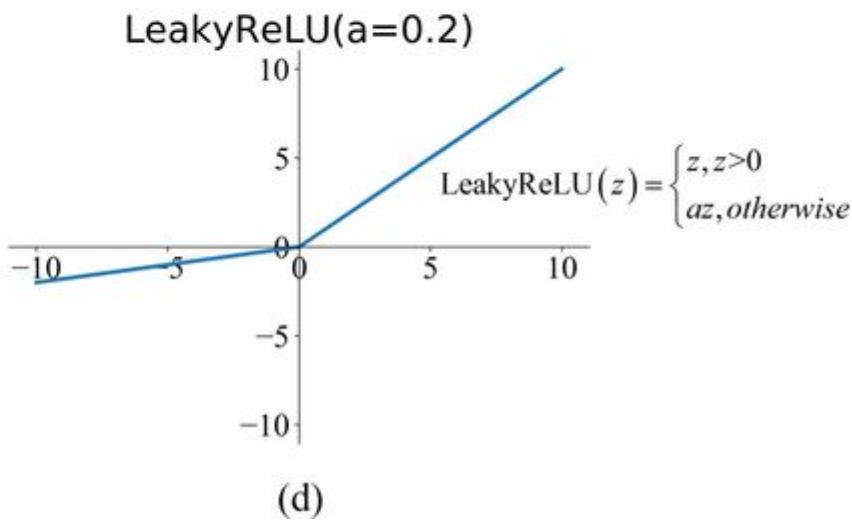
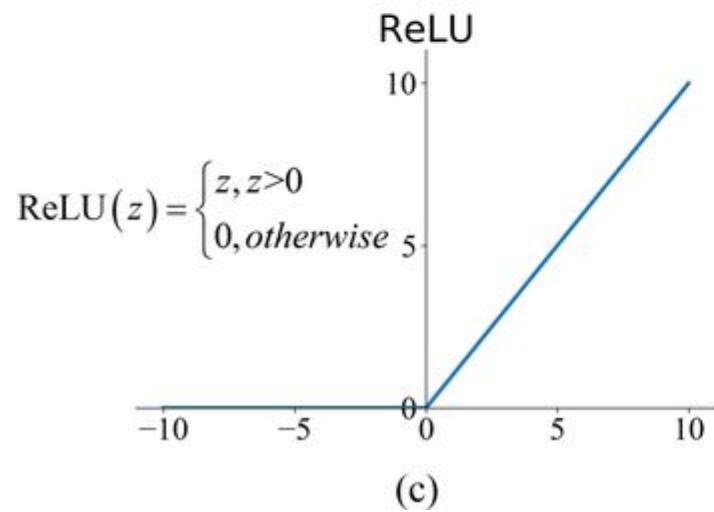
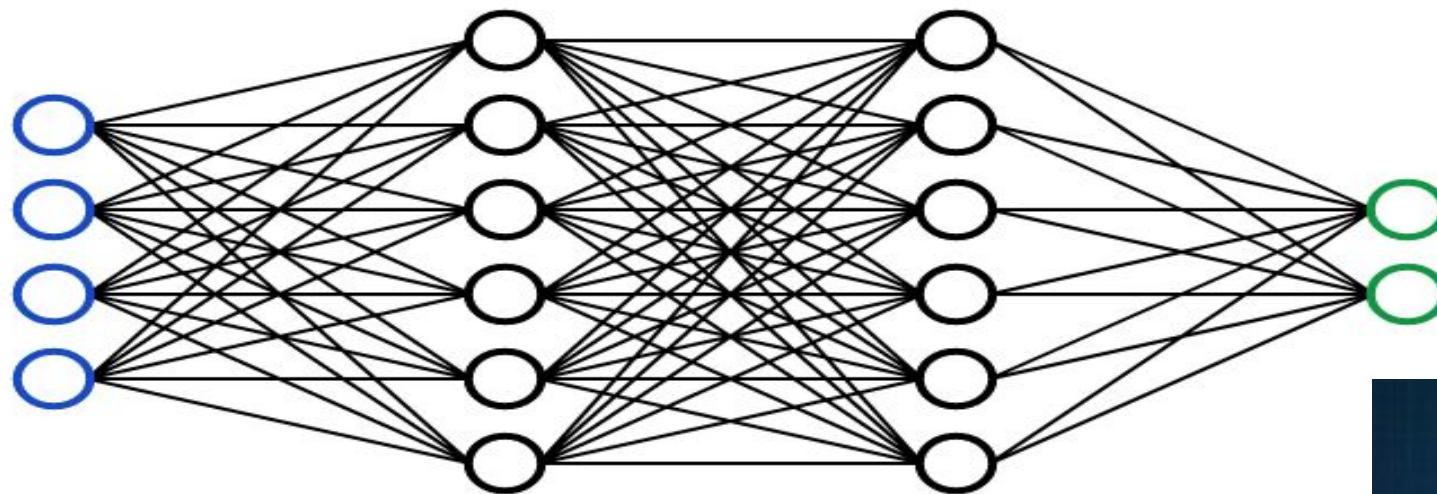


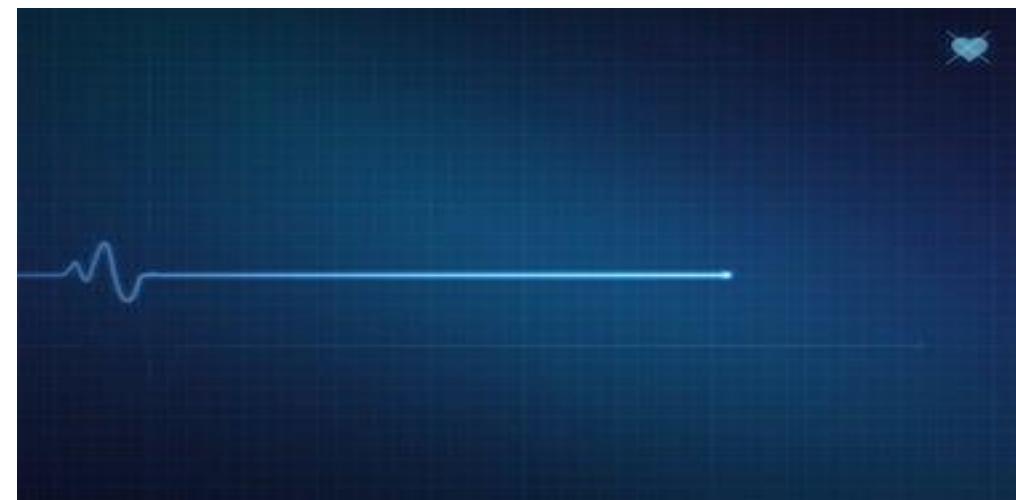
Figure 1: The GELU ( $\mu = 0, \sigma = 1$ ), ReLU, and ELU ( $\alpha = 1$ ).



# How **not** to initialize: zero weights



No symmetry breaking



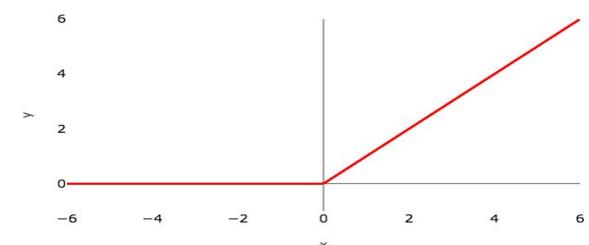
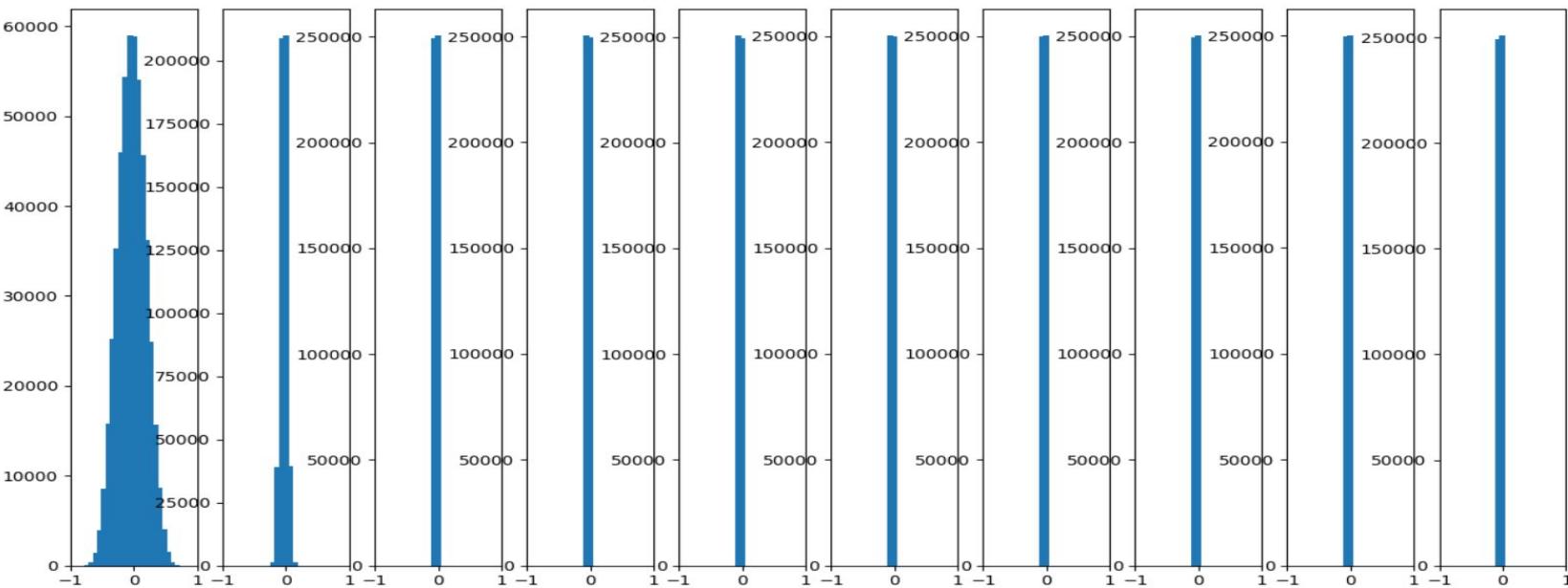
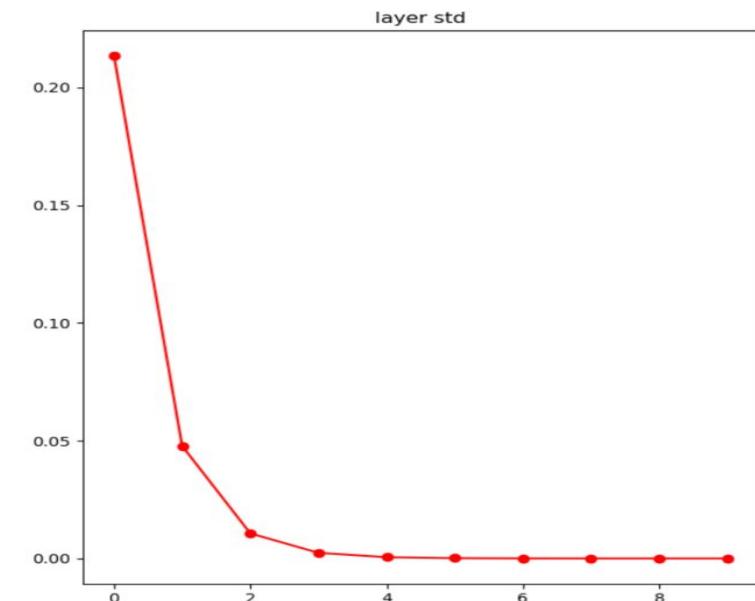
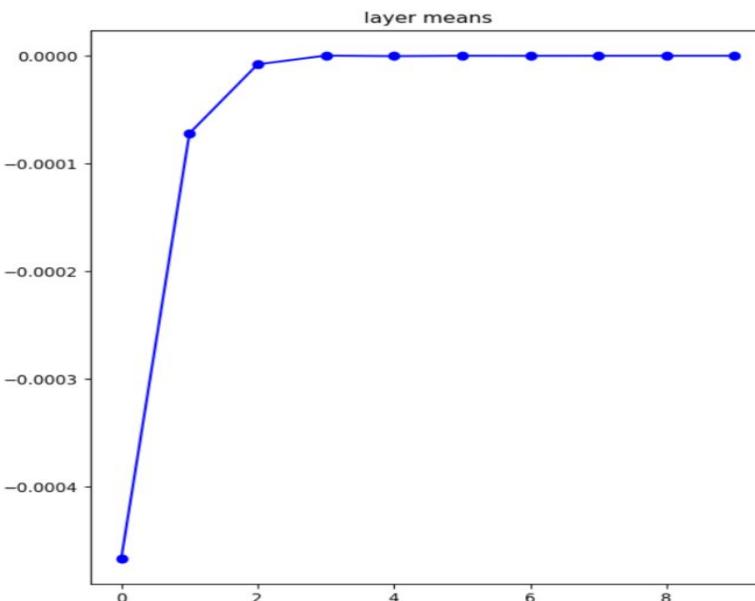
# Weight initialization

Example credit: Andrew Karpathy

[https://colab.research.google.com/drive/1CvCXZIevs6MvQnldyG2UHZe  
vG0bKaeiu?usp=sharing](https://colab.research.google.com/drive/1CvCXZIevs6MvQnldyG2UHZe<br/>vG0bKaeiu?usp=sharing)

```
W = np.random.randn(in, out)*0.01
```

input layer had mean -0.001449 and stddev 0.999223  
hidden layer 0 had mean 0.000289 and stddev 0.213667  
hidden layer 1 had mean -0.000064 and stddev 0.047754  
hidden layer 2 had mean -0.000001 and stddev 0.010667  
hidden layer 3 had mean -0.000003 and stddev 0.002391  
hidden layer 4 had mean 0.000000 and stddev 0.000535  
hidden layer 5 had mean -0.000000 and stddev 0.000119  
hidden layer 6 had mean 0.000000 and stddev 0.000027  
hidden layer 7 had mean -0.000000 and stddev 0.000006  
hidden layer 8 had mean 0.000000 and stddev 0.000001  
hidden layer 9 had mean 0.000000 and stddev 0.000000



# Xavier initialization

- *Understanding the difficulty of training deep feedforward neural networks*, Xavier Glorot Yoshua Bengio  
<https://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf>
- Random initialization from a distribution with a variance of:  
$$\frac{1}{in\_neurons}$$
- If inputs are roughly mean 0 and std 1, this initialization will also cause the outputs to have mean 0 and stddev 1

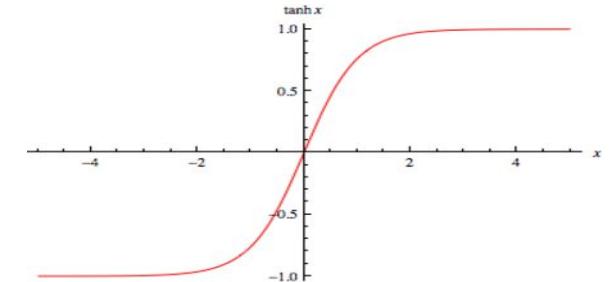
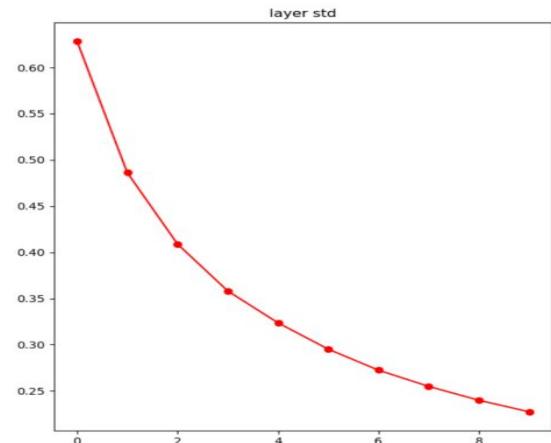
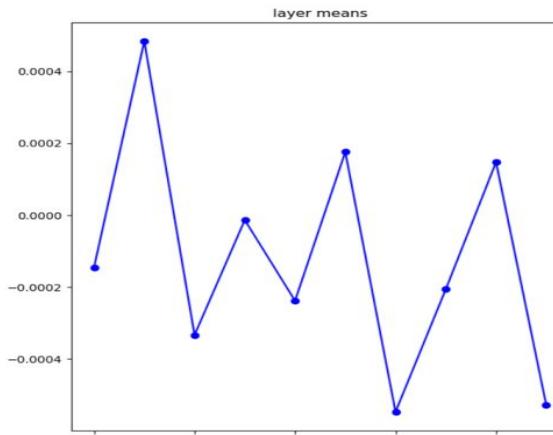
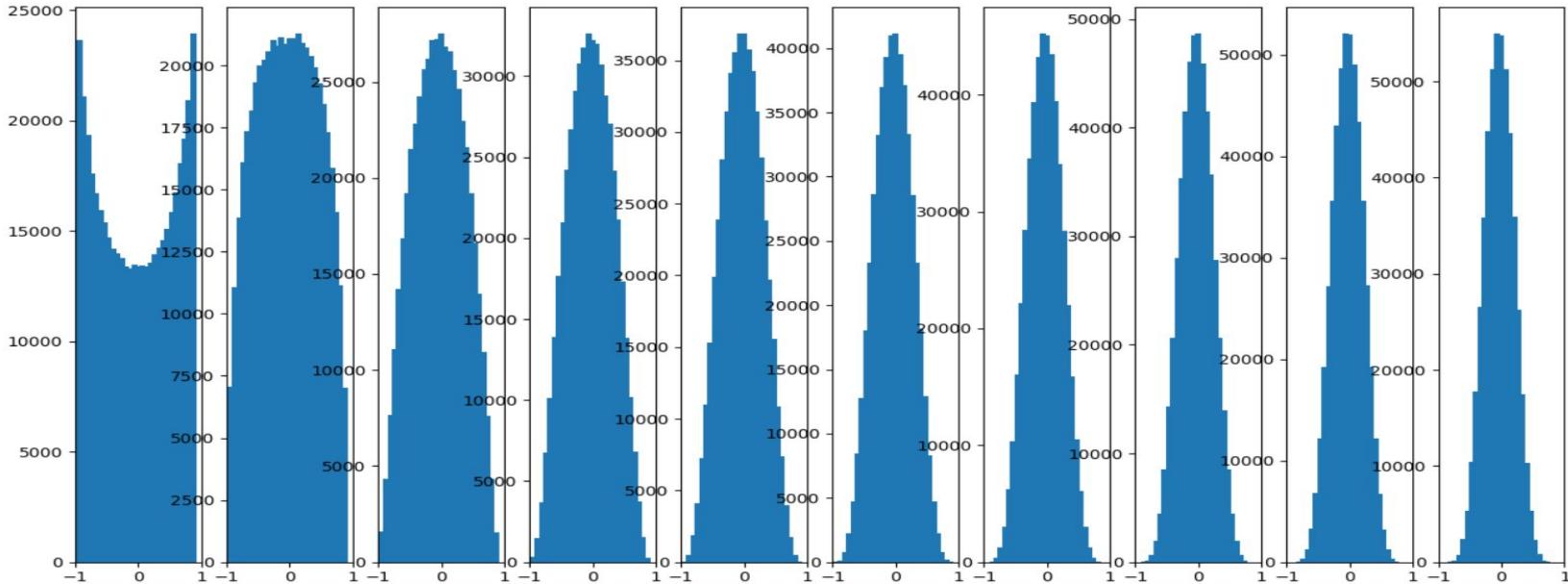
Additional reading:

<https://www.machinecurve.com/index.php/2019/09/16/he-xavier-initialization-activation-functions-choose-wisely/>

# Xavier initialization

$W = \text{np.random.randn(in, out)}/\text{np.sqrt(in)}$

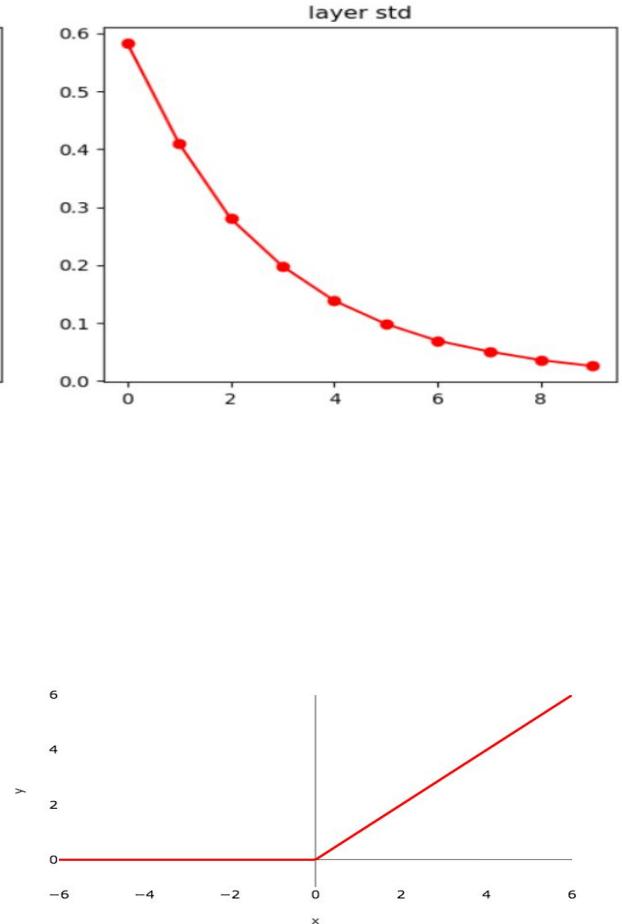
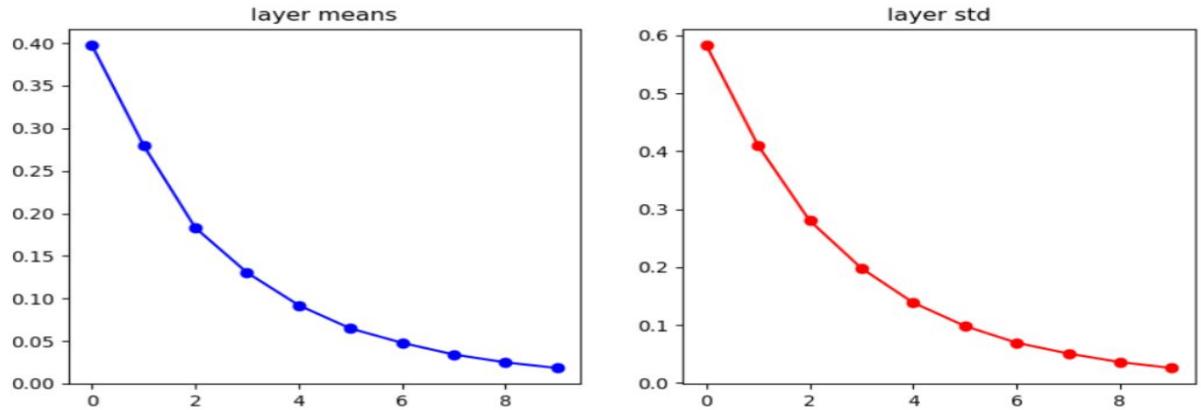
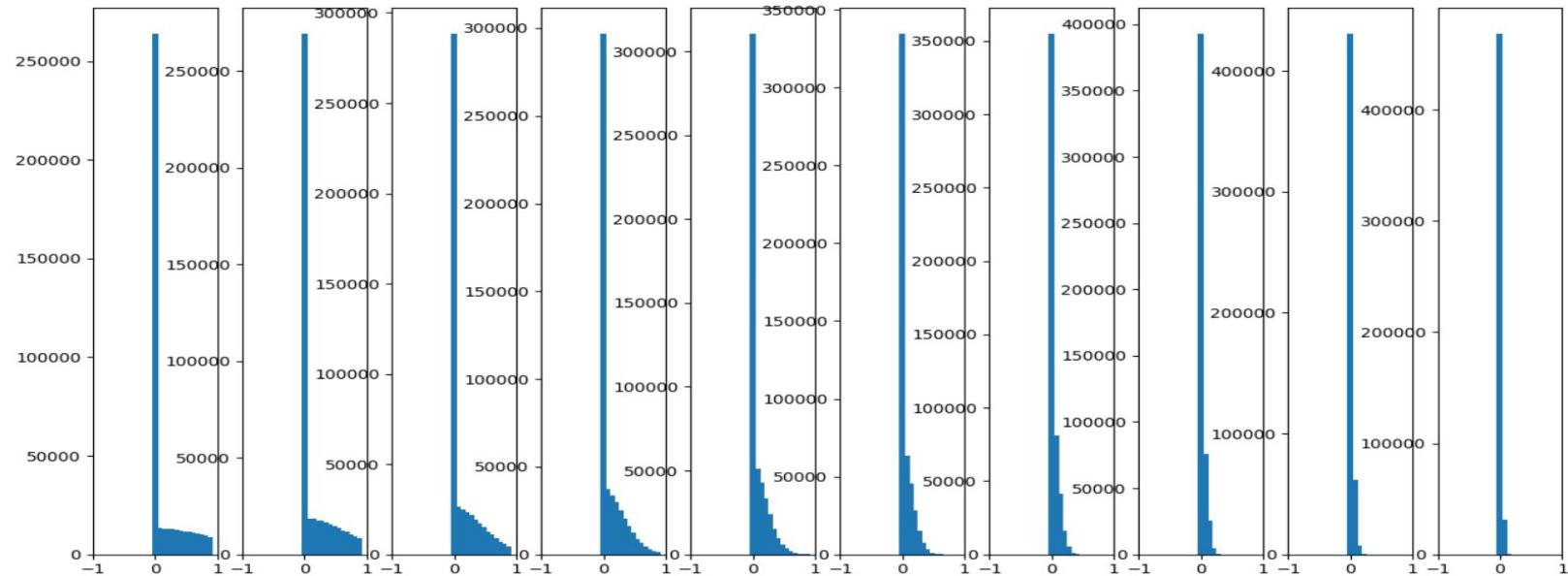
```
hidden layer 0 had mean -0.000145 and stddev 0.628485
hidden layer 1 had mean 0.000485 and stddev 0.486178
hidden layer 2 had mean -0.000333 and stddev 0.408851
hidden layer 3 had mean -0.000013 and stddev 0.358011
hidden layer 4 had mean -0.000238 and stddev 0.323565
hidden layer 5 had mean 0.000176 and stddev 0.295040
hidden layer 6 had mean -0.000547 and stddev 0.272295
hidden layer 7 had mean -0.000206 and stddev 0.254713
hidden layer 8 had mean 0.000148 and stddev 0.239710
hidden layer 9 had mean -0.000529 and stddev 0.227241
```



# Xavier initialization

$W = \text{np.random.randn(in, out)} / \text{np.sqrt(in)}$

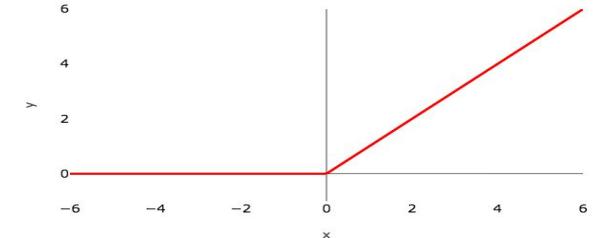
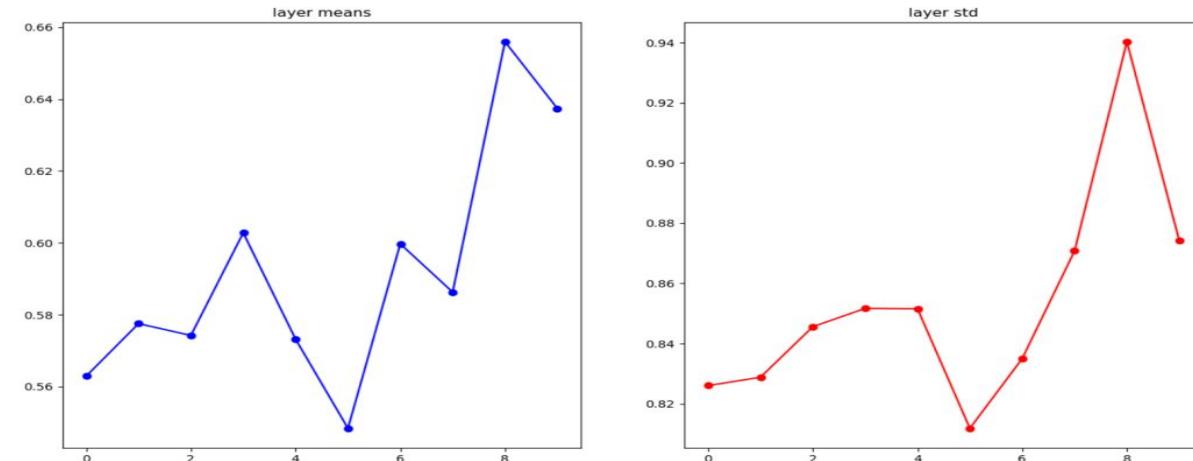
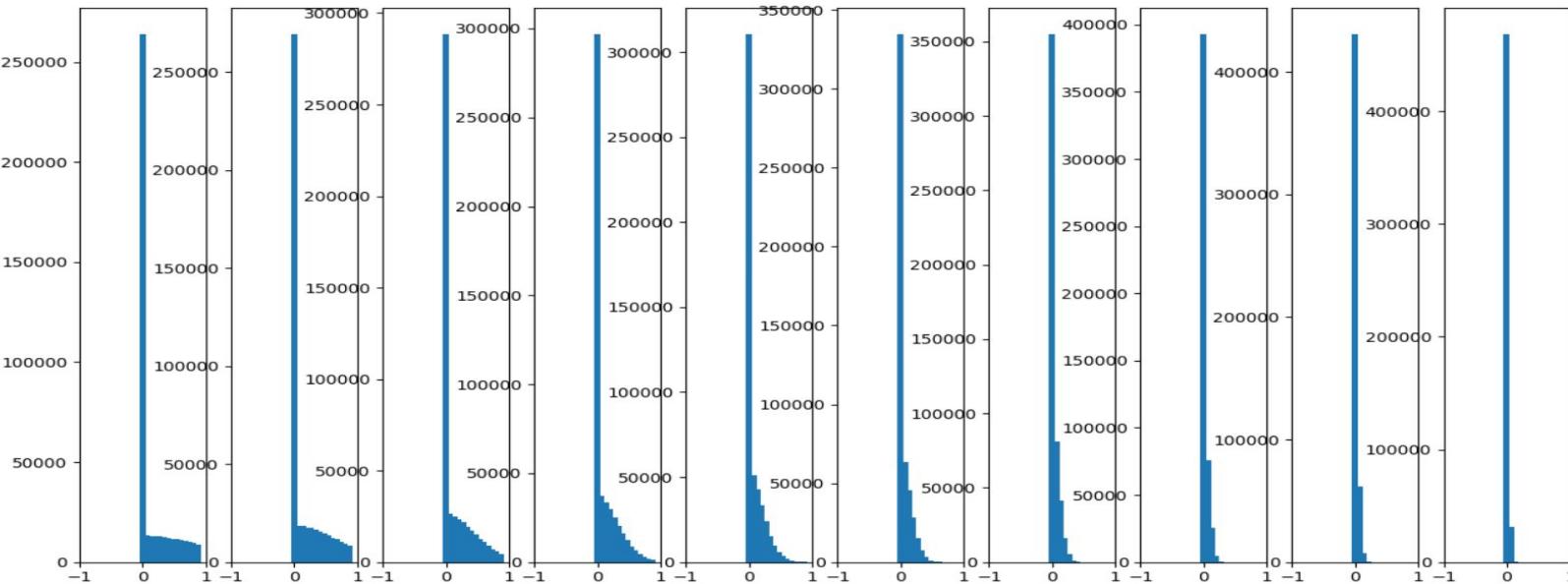
input layer had mean 0.000422 and stddev 1.000922  
hidden layer 0 had mean 0.397685 and stddev 0.583384  
hidden layer 1 had mean 0.280004 and stddev 0.410386  
hidden layer 2 had mean 0.183151 and stddev 0.280291  
hidden layer 3 had mean 0.130450 and stddev 0.198090  
hidden layer 4 had mean 0.092056 and stddev 0.138666  
hidden layer 5 had mean 0.064787 and stddev 0.098349  
hidden layer 6 had mean 0.047867 and stddev 0.069325  
hidden layer 7 had mean 0.034165 and stddev 0.050724  
hidden layer 8 had mean 0.024928 and stddev 0.035902  
hidden layer 9 had mean 0.018272 and stddev 0.025658



# He initialization

$W = \text{np.random.randn(in, out)}/\text{np.sqrt(in/2)}$

input layer had mean -0.001160 and stddev 1.000333  
hidden layer 0 had mean 0.562934 and stddev 0.826029  
hidden layer 1 had mean 0.577537 and stddev 0.828810  
hidden layer 2 had mean 0.574250 and stddev 0.845599  
hidden layer 3 had mean 0.602799 and stddev 0.851682  
hidden layer 4 had mean 0.573220 and stddev 0.851573  
hidden layer 5 had mean 0.548410 and stddev 0.811833  
hidden layer 6 had mean 0.599740 and stddev 0.834997  
hidden layer 7 had mean 0.586302 and stddev 0.870794  
hidden layer 8 had mean 0.655965 and stddev 0.940311  
hidden layer 9 had mean 0.637412 and stddev 0.874220



# “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”

<https://arxiv.org/pdf/1502.01852.pdf>

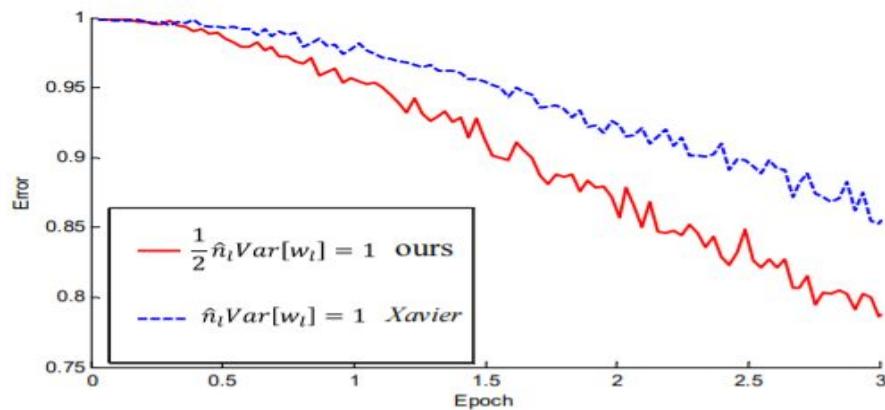


Figure 2. The convergence of a **22-layer** large model (B in Table 3). The x-axis is the number of training epochs. The y-axis is the top-1 error of 3,000 random val samples, evaluated on the center crop. We use ReLU as the activation for both cases. Both our initialization (red) and “Xavier” (blue) [7] lead to convergence, but ours starts reducing error earlier.

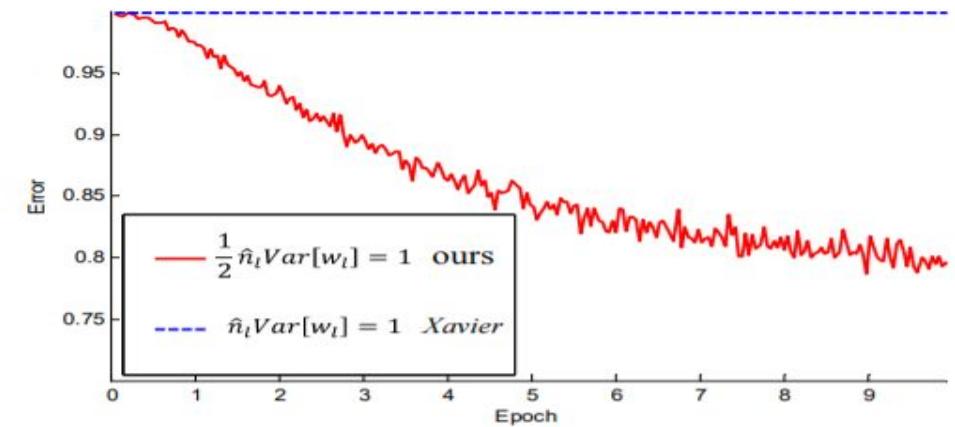
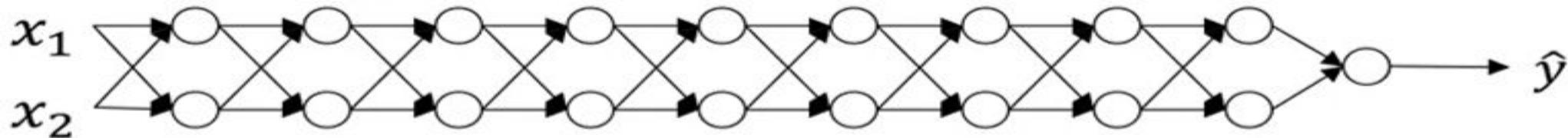


Figure 3. The convergence of a **30-layer** small model (see the main text). We use ReLU as the activation for both cases. Our initialization (red) is able to make it converge. But “Xavier” (blue) [7] completely stalls - we also verify that its gradients are all diminishing. It does not converge even given more epochs.

# Vanishing and exploding gradients



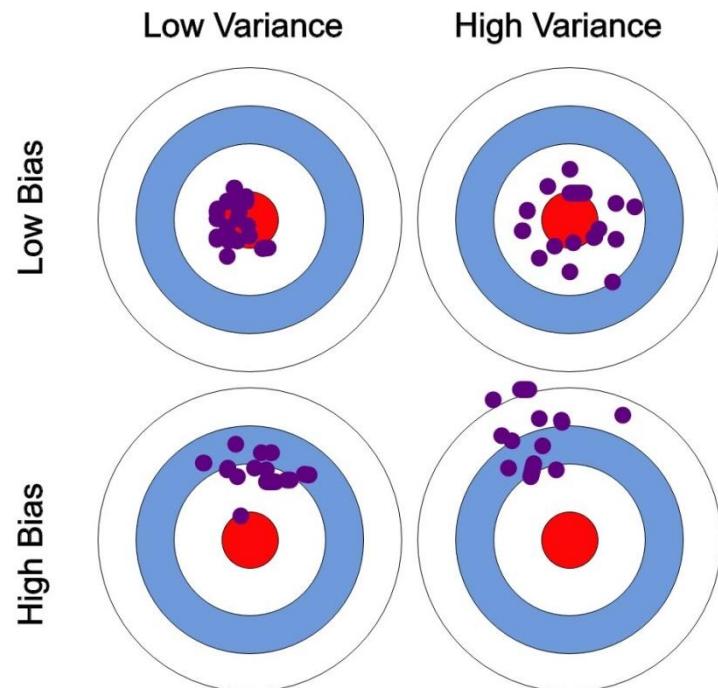
# Computer vision and deep learning

Lecture 5

# Training a neural network

# Bias and Variance

- *Bias*: difference the average prediction of our model and the correct value which we are trying to predict
- *Variance*: the variability of model prediction/ the spread of the predictions



# Bias and Variance

- High bias: under-fitting the data
  - Model pays little attention to training data and over simplifies the problem
- High variance: over-fitting the data
  - Model performs well on the training set, but not on test set

# Dealing with variance and bias problems

- High bias?

# Dealing with variance and bias problems

- High bias? (train set performance)
  - Try a more complex model/different NN architecture
  - Train longer
- High variance?

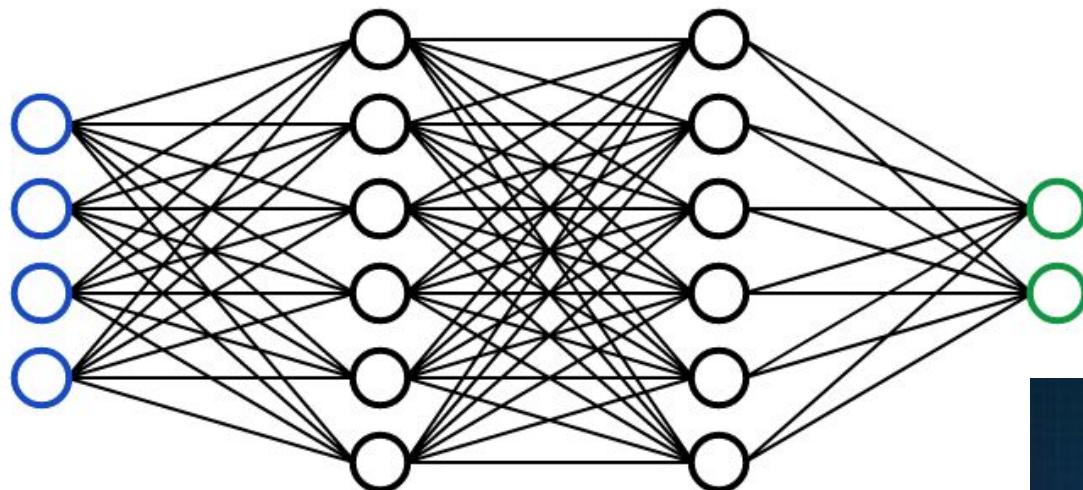
# Dealing with variance and bias problems

- High bias? (train set performance)
  - Try a more complex model/different NN architecture
  - Train longer
- High variance? (validation set performance)
  - Add more data
  - Add more regularization techniques
  - Try a different NN architecture

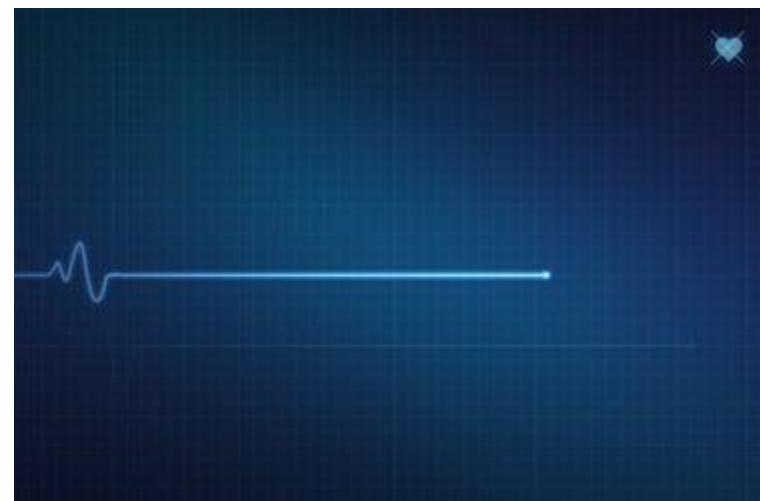
# Training a neural network

*Weights initialization*

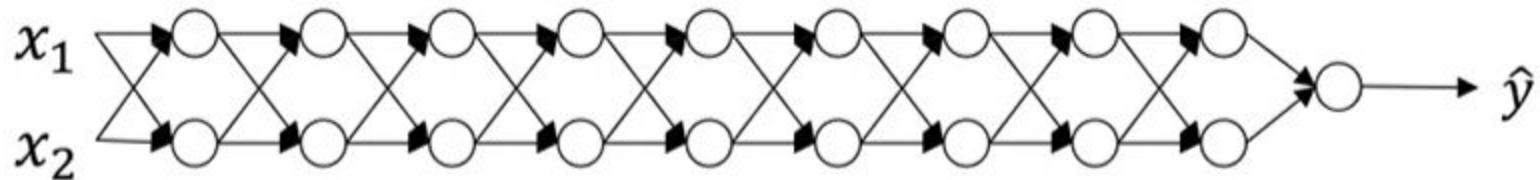
# How **not** to initialize: zero weights



No symmetry breaking



# Vanishing and exploding gradients



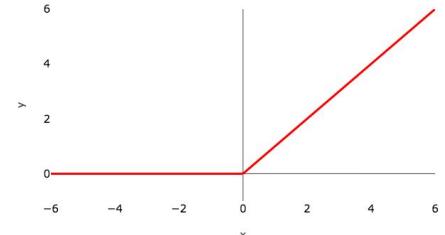
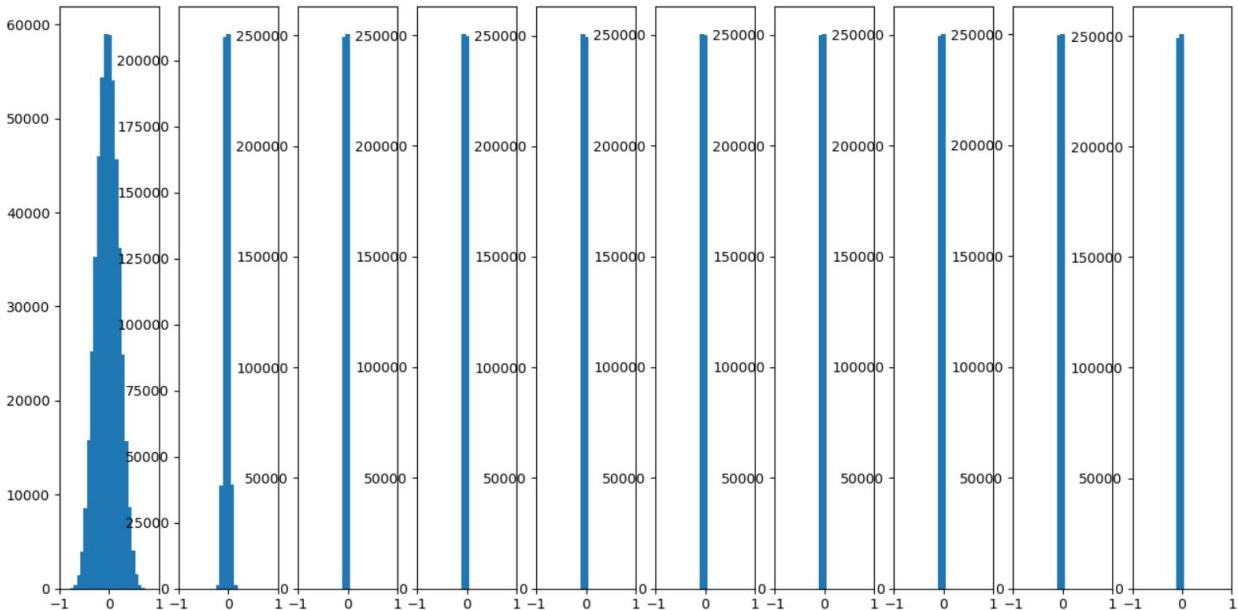
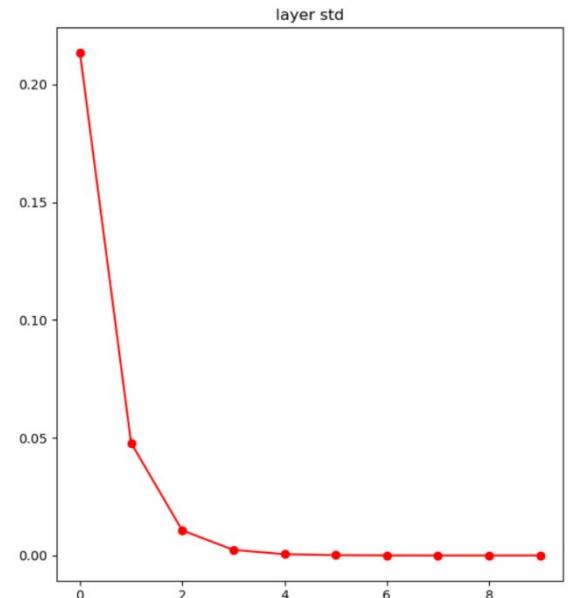
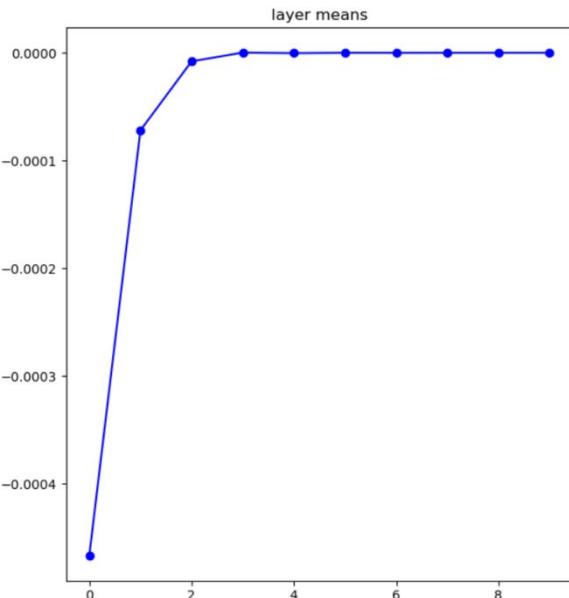
# Weight initialization

Example credit: Andrew Karpathy

<https://colab.research.google.com/drive/1CvCXZIevs6MvQnldyG2UHZevG0bKaeiu?usp=sharing>

`W = np.random.randn(in, out)*0.01`

input layer had mean -0.001449 and stddev 0.999223  
hidden layer 0 had mean 0.000289 and stddev 0.213667  
hidden layer 1 had mean -0.000064 and stddev 0.047754  
hidden layer 2 had mean -0.000001 and stddev 0.010667  
hidden layer 3 had mean -0.000003 and stddev 0.002391  
hidden layer 4 had mean 0.000000 and stddev 0.000535  
hidden layer 5 had mean -0.000000 and stddev 0.000119  
hidden layer 6 had mean 0.000000 and stddev 0.000027  
hidden layer 7 had mean -0.000000 and stddev 0.000006  
hidden layer 8 had mean 0.000000 and stddev 0.000001  
hidden layer 9 had mean 0.000000 and stddev 0.000000



# Xavier initialization

- *Understanding the difficulty of training deep feedforward neural networks*, Xavier Glorot Yoshua Bengio  
<https://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf>
- Random initialization from a distribution with a variance of:  
$$\frac{1}{in\_neurons}$$
- If inputs are roughly mean 0 and std 1, this initialization will also cause the outputs to have mean 0 and stddev 1

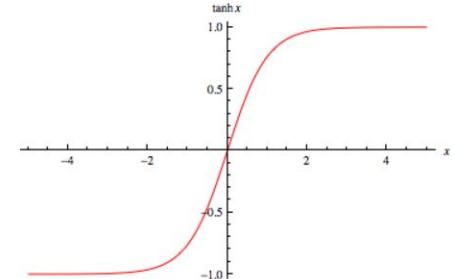
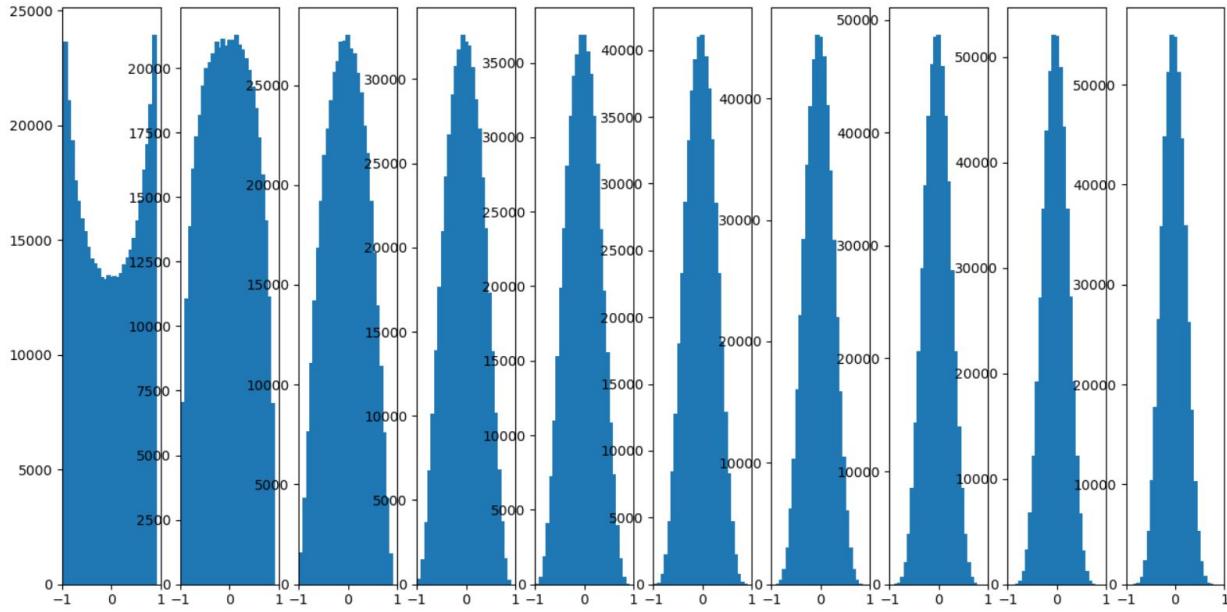
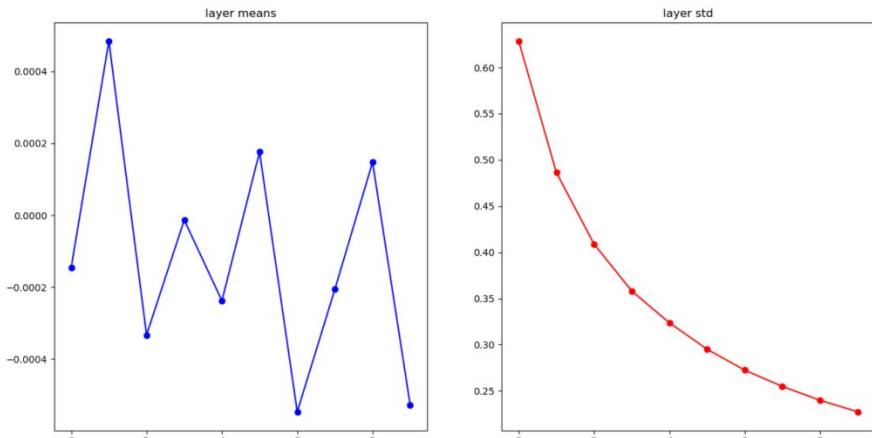
Additional reading:

<https://www.machinecurve.com/index.php/2019/09/16/he-xavier-initialization-activation-functions-choose-wisely/>

# Xavier initialization

$W = \text{np.random.randn(in, out)}/\text{np.sqrt(in)}$

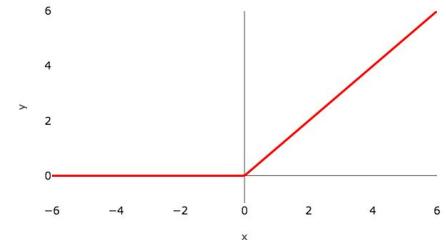
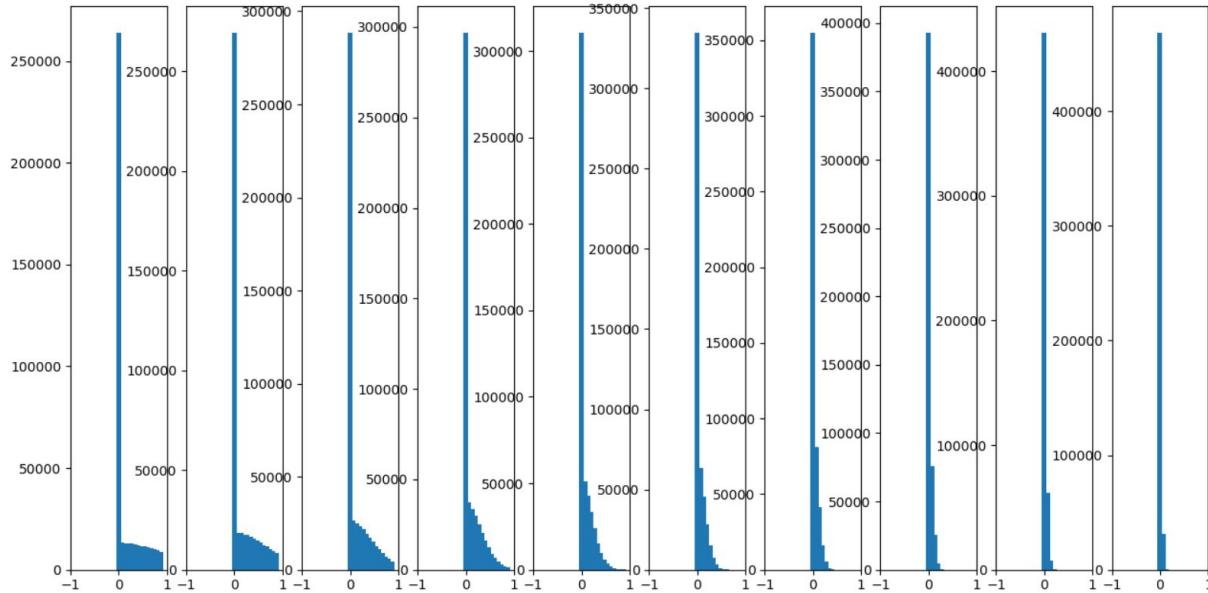
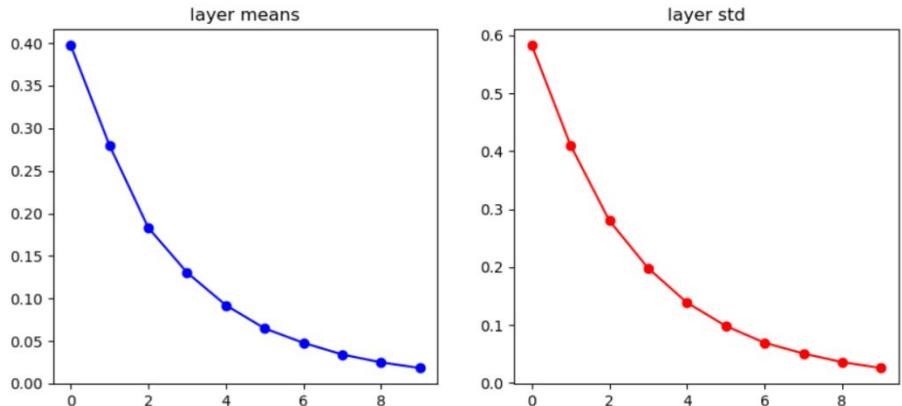
hidden layer 0 had mean -0.000145 and stddev 0.628485  
hidden layer 1 had mean 0.000485 and stddev 0.486178  
hidden layer 2 had mean -0.000333 and stddev 0.408851  
hidden layer 3 had mean -0.000013 and stddev 0.358011  
hidden layer 4 had mean -0.000238 and stddev 0.323565  
hidden layer 5 had mean 0.000176 and stddev 0.295040  
hidden layer 6 had mean -0.000547 and stddev 0.272295  
hidden layer 7 had mean -0.000206 and stddev 0.254713  
hidden layer 8 had mean 0.000148 and stddev 0.239710  
hidden layer 9 had mean -0.000529 and stddev 0.227241



# Xavier initialization

$W = \text{np.random.randn(in, out)}/\text{np.sqrt(in)}$

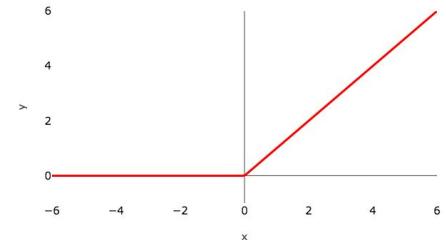
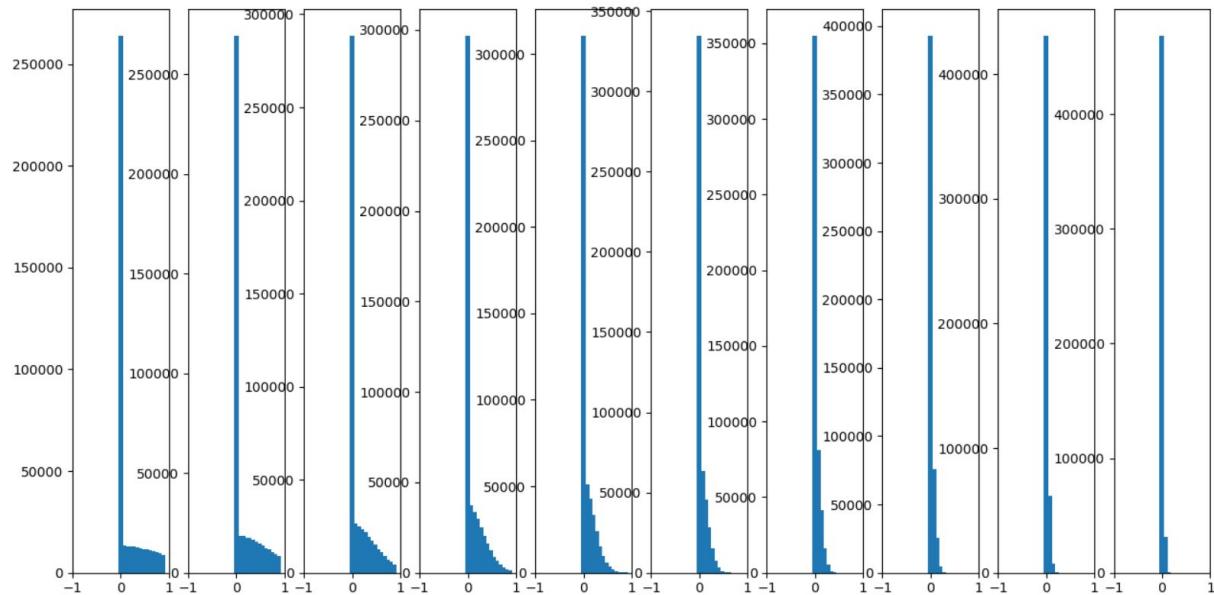
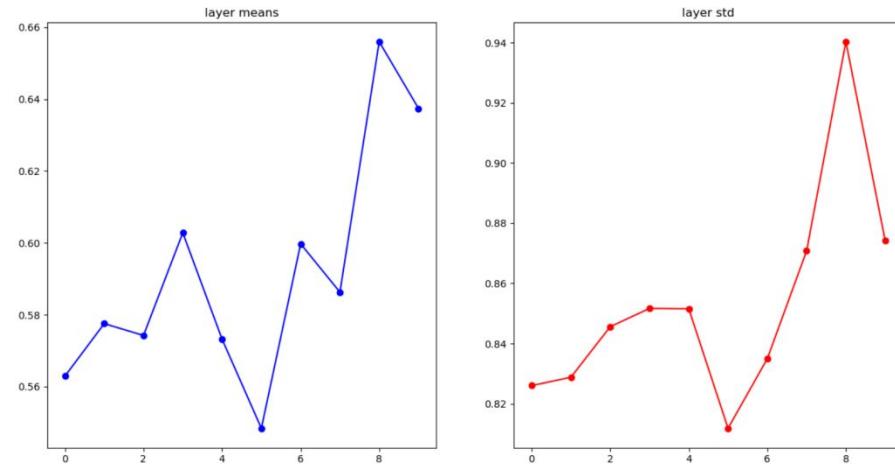
input layer had mean 0.000422 and stddev 1.000922  
hidden layer 0 had mean 0.397685 and stddev 0.583384  
hidden layer 1 had mean 0.280004 and stddev 0.410386  
hidden layer 2 had mean 0.183151 and stddev 0.280291  
hidden layer 3 had mean 0.130450 and stddev 0.198090  
hidden layer 4 had mean 0.092056 and stddev 0.138666  
hidden layer 5 had mean 0.064787 and stddev 0.098349  
hidden layer 6 had mean 0.047867 and stddev 0.069325  
hidden layer 7 had mean 0.034165 and stddev 0.050724  
hidden layer 8 had mean 0.024928 and stddev 0.035902  
hidden layer 9 had mean 0.018272 and stddev 0.025658



# He initialization

$W = \text{np.random.randn(in, out)}/\text{np.sqrt(in/2)}$

input layer had mean -0.001160 and stddev 1.000333  
hidden layer 0 had mean 0.562934 and stddev 0.826029  
hidden layer 1 had mean 0.577537 and stddev 0.828810  
hidden layer 2 had mean 0.574250 and stddev 0.845599  
hidden layer 3 had mean 0.602799 and stddev 0.851682  
hidden layer 4 had mean 0.573220 and stddev 0.851573  
hidden layer 5 had mean 0.548410 and stddev 0.811833  
hidden layer 6 had mean 0.599740 and stddev 0.834997  
hidden layer 7 had mean 0.586302 and stddev 0.870794  
hidden layer 8 had mean 0.655965 and stddev 0.940311  
hidden layer 9 had mean 0.637412 and stddev 0.874220



# “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”

<https://arxiv.org/pdf/1502.01852.pdf>

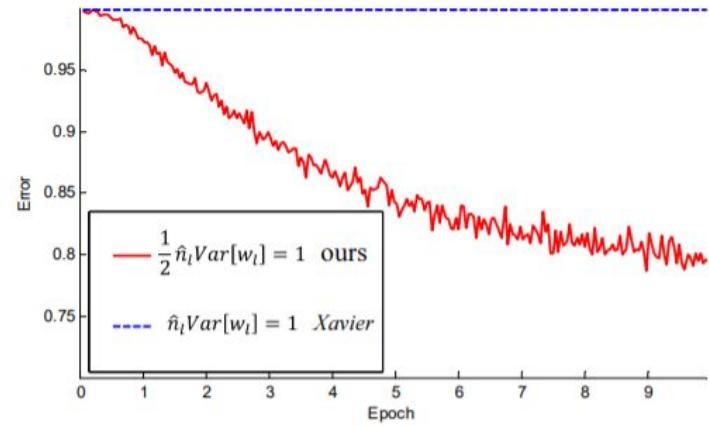
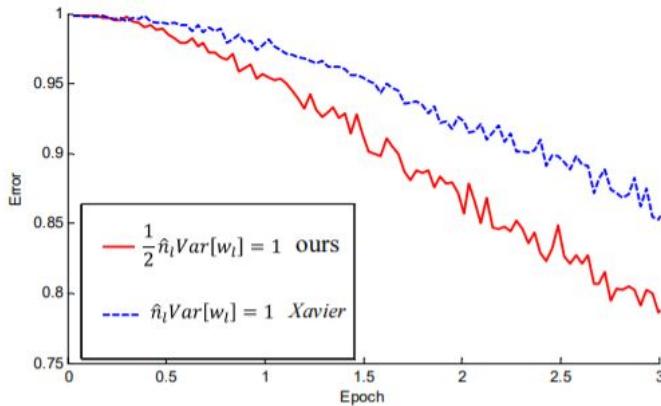


Figure 2. The convergence of a **22-layer** large model (B in Table 3). The x-axis is the number of training epochs. The y-axis is the top-1 error of 3,000 random val samples, evaluated on the center crop. We use ReLU as the activation for both cases. Both our initialization (red) and “Xavier” (blue) [7] lead to convergence, but ours starts reducing error earlier.

Figure 3. The convergence of a **30-layer** small model (see the main text). We use ReLU as the activation for both cases. Our initialization (red) is able to make it converge. But “Xavier” (blue) [7] completely stalls - we also verify that its gradients are all diminishing. It does not converge even given more epochs.

# Torch initialization

- `torch.nn.init.zeros_(tensor)`
- `torch.nn.init.xavier_normal_(tensor, gain=1.0)`
- `torch.nn.init.sparse_(tensor, sparsity, std=0.01)`
- `torch.nn.init.kaiming_normal_(tensor, a=0, mode='fan_in', nonlinearity='leaky_relu')`

```
class FeedforwardNeuralNetModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(FeedforwardNeuralNetModel, self).__init__()
        # Linear function
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        # Linear weight, W, Y = WX + B
        nn.init.xavier_normal_(self.fc1.weight)
        # Non-linearity
        self.relu = nn.ReLU()
        # Linear function (readout)
        self.fc2 = nn.Linear(hidden_dim, output_dim)
        nn.init.xavier_normal_(self.fc2.weight)
```

# Training a neural network

*Regularization*

# Regularization

- Remember regularization techniques
  - L1 regularization
  - L2 regularization

# Regularization

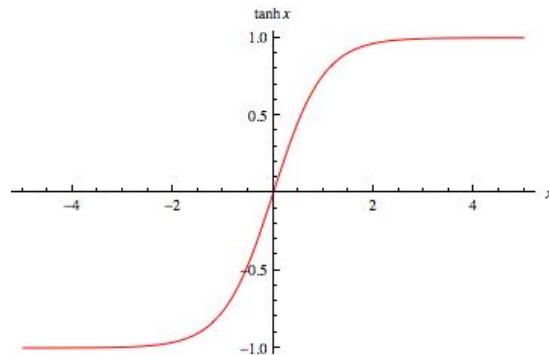
- Weight decay

$$J = \frac{1}{m} \sum_1^m L(\hat{y}_i, y_i) + \lambda \frac{1}{2 \cdot m} \sum_1^l \|W_l\|_F^2$$

$$\|W_l\|_F^2 = \sum_{i=1}^{n_{l-1}} \sum_{j=1}^{n_l} (w_{ij})^2$$

# Regularization

- Why does regularization reduce overfitting?



# Torch weight decay

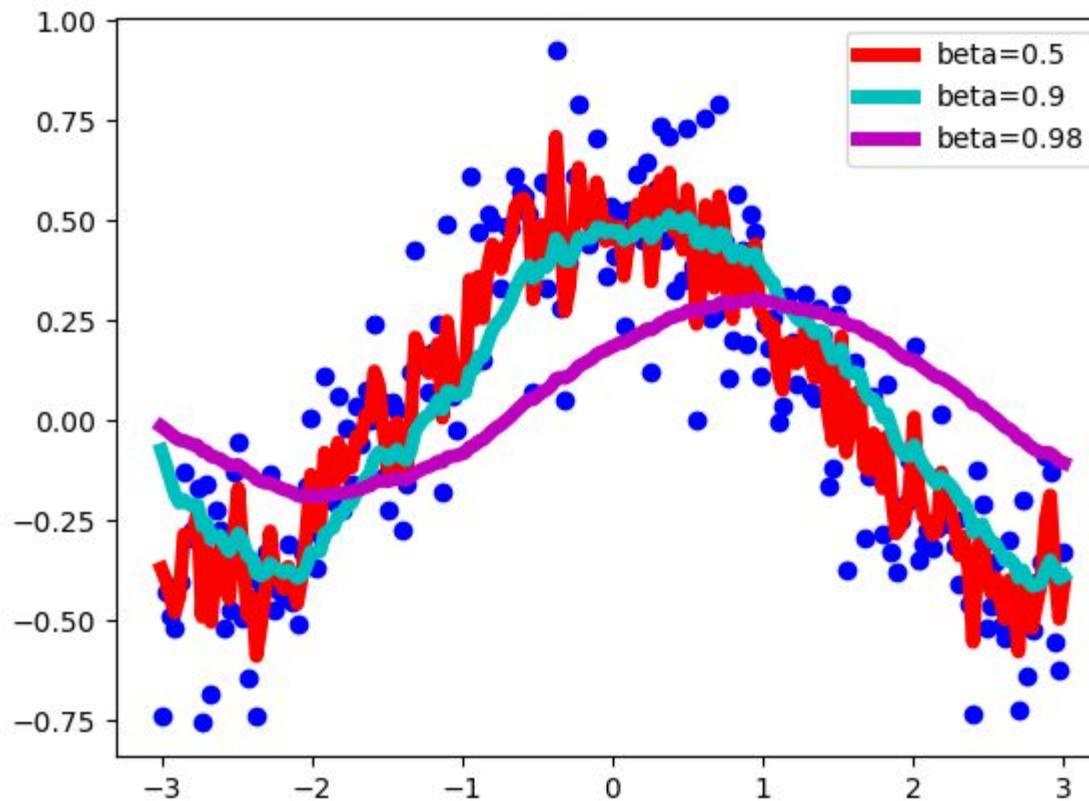
- You can specify weight decay when creating the *optimizer* object. For example:

```
optimizer =  
    torch.optim.SGD(model.parameters(),  
        lr=1e-3, weight_decay = 0.5)
```

*Exponential weighted averages*

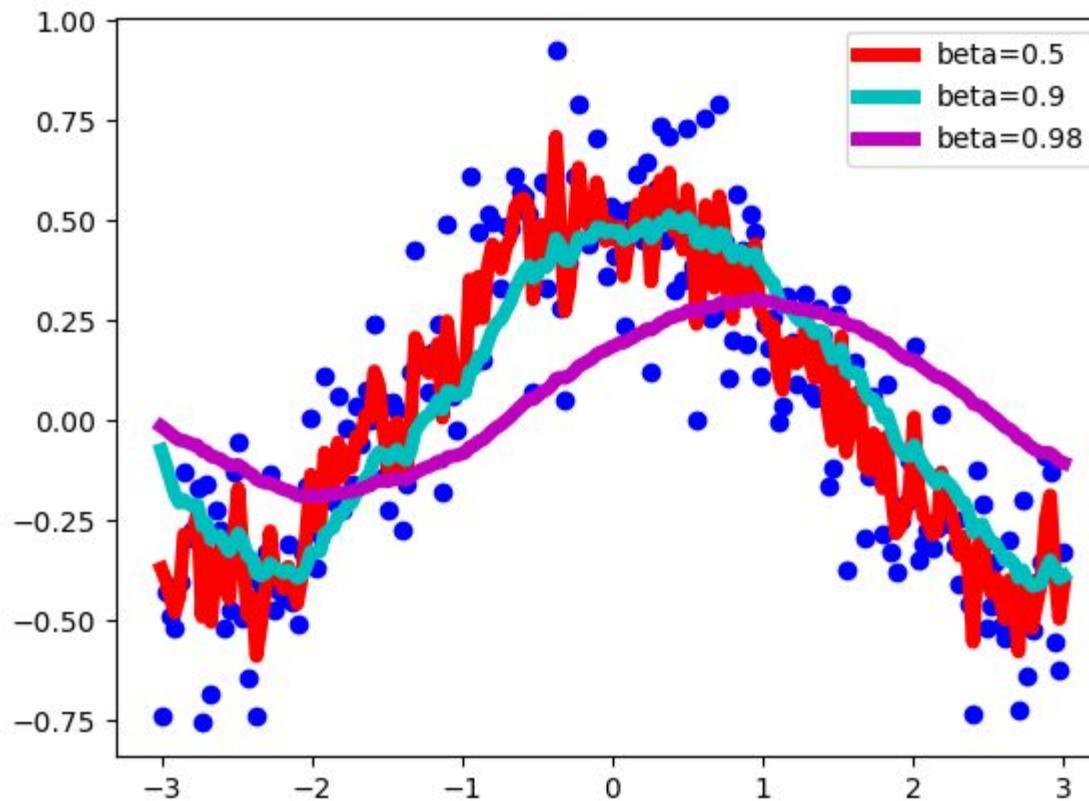
<https://colab.research.google.com/drive/1CTSctxtN1JDvGZOswpLIG9ZkC4-s0ym2?usp=sharing>

# Exponential weighted averages



$$V_t = \beta \cdot V_{t-1} + (1 - \beta) \cdot x$$

# Exponential weighted averages



$V_t$  approximates over:  $\frac{1}{1 - \beta}$  samples

<https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d>

# Exponential weighted average

$$V_t = \beta V_{t-1} + (1 - \beta) S_t$$

$$V_{t-1} = \beta V_{t-2} + (1 - \beta) S_{t-1}$$

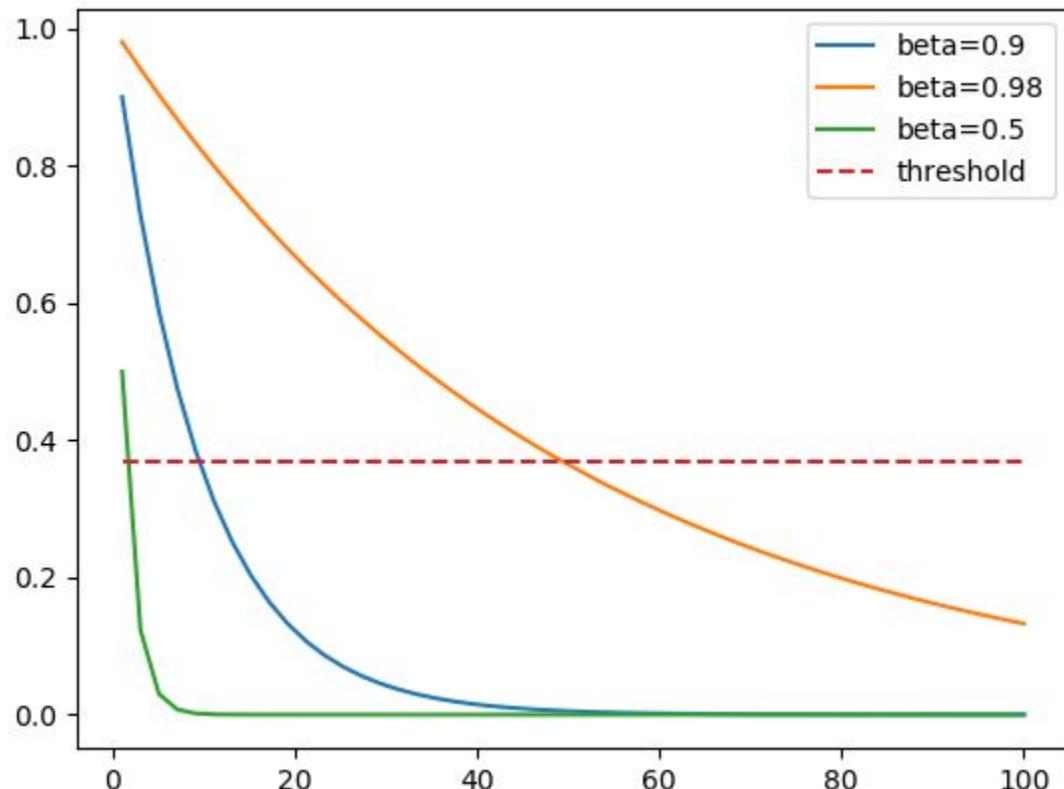
$$V_{t-2} = \beta V_{t-3} + (1 - \beta) S_{t-2}$$

$$V_t = \beta(\beta(\beta V_{t-3} + (1 - \beta) S_{t-2}) + (1 - \beta) S_{t-1}) + (1 - \beta) S_t$$

$$V_t = \beta\beta(1 - \beta) S_{t-2} + \dots + \beta(1 - \beta) S_{t-1} + \dots + (1 - \beta) S_t$$

The coefficients add up to approximately 1

# Exponential weighted average



threshold  $1/e$

$$(1 - \varepsilon)^{1/\varepsilon} = 1/e$$

# Exponential weighted average

## Bias correction

- The first couple of iterations will provide a pretty bad averages because we don't have enough values yet to average over
- Instead of using  $V_t$  use the bias corrected version of it:

$$\tilde{V}_t = \frac{V_t}{1 - \beta^t}$$

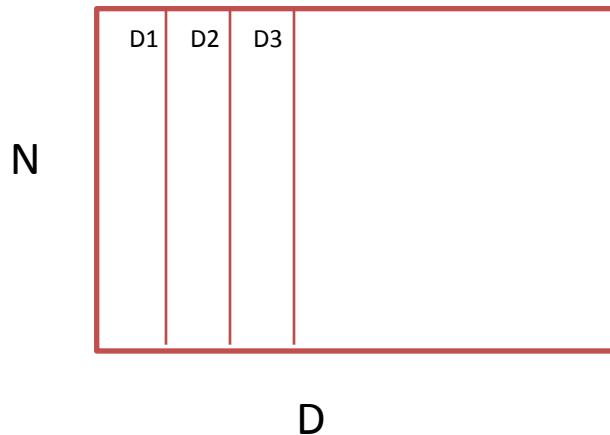
<https://www.youtube.com/watch?v=IAq96T8FkTw>

<https://www.youtube.com/watch?v=NxTFlzBjS-4>

<https://www.youtube.com/watch?v=lWzo8CajF5s> – bias correction

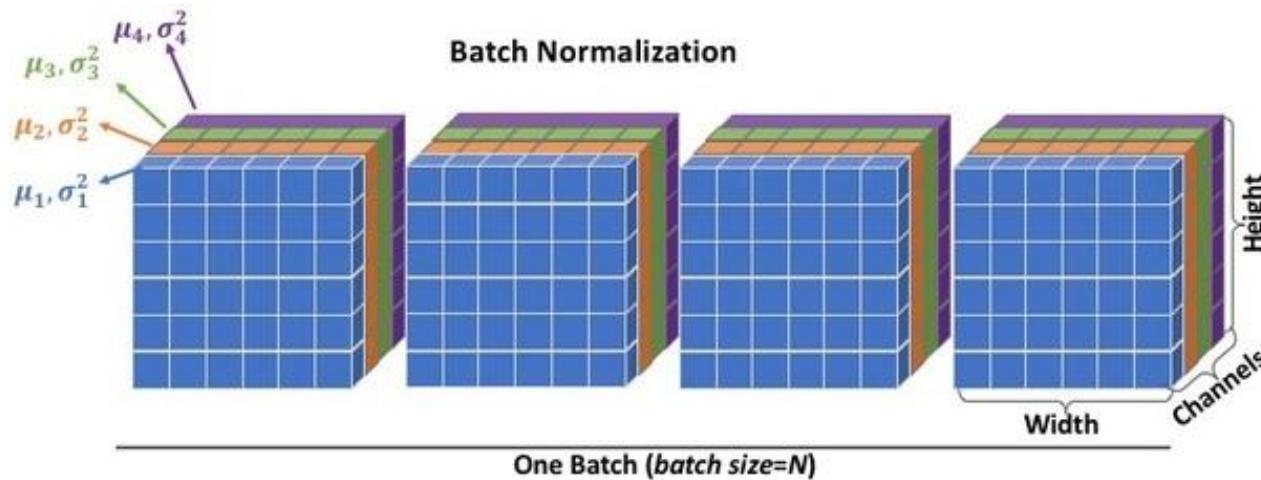
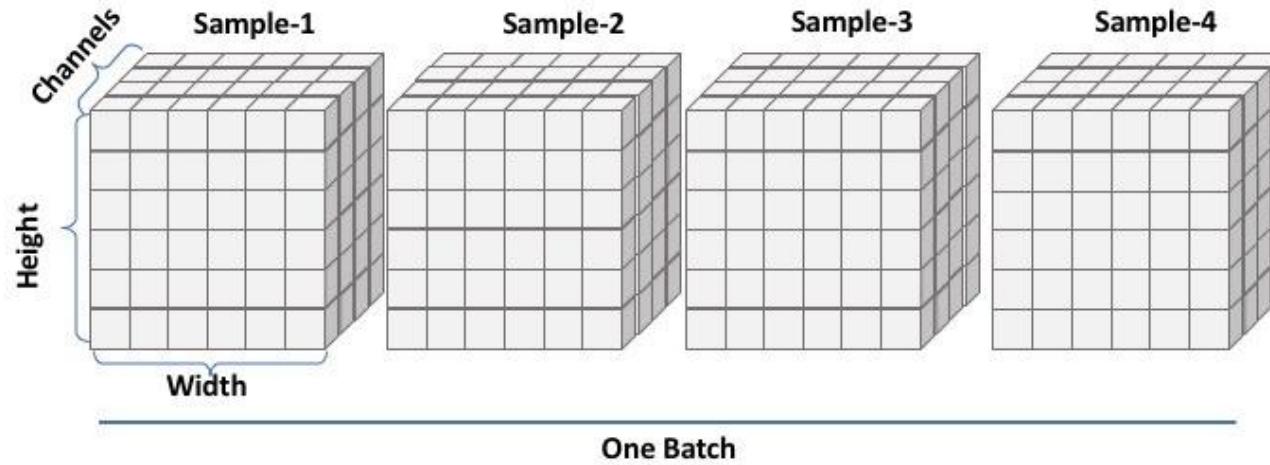
# Batch normalization

- Compute mean and variance across each dimension



- Normalize

# Batch normalization



# Batch normalization

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_1 \dots m\}$ ;

Parameters to be learned:  $\gamma, \beta$

**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

**Algorithm 1:** Batch Normalizing Transform, applied to activation  $x$  over a mini-batch.

algorithm source: <https://arxiv.org/abs/1502.03167>

Simple idea: just make the output of the **linear units** have 0 mean and unit standard deviation

Added them just before the activation function

# Batch normalization

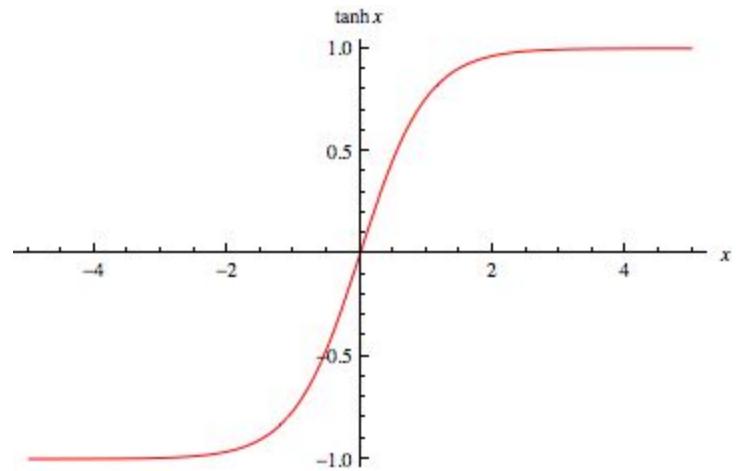
Normalize:

$$\mu = \frac{1}{m} \sum_i z^{(i)} \quad \sigma^2 = \frac{1}{m} \sum_i (z^{(i)} - \mu)^2$$

$$z_{norm}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

Allow the network to modify the range

$$\tilde{z}^{(i)} = \gamma z_{norm}^{(i)} + \beta$$



# Batch normalization

Normalize:

$$\mu = \frac{1}{m} \sum_i z^{(i)} \quad \sigma^2 = \frac{1}{m} \sum_i (z^{(i)} - \mu)^2$$

$$z_{norm}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

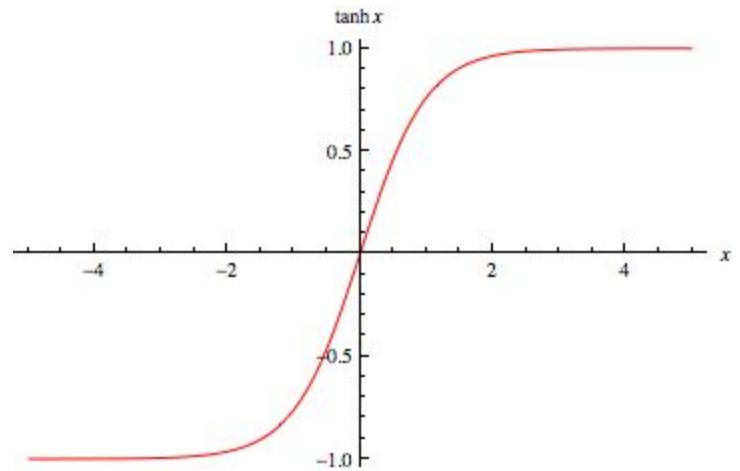
Allow the network to modify the range

$$\tilde{z}^{(i)} = \gamma z_{norm}^{(i)} + \beta$$

Can learn the identity function

$$\gamma = \sqrt{\sigma^2}$$

$$\beta = \mu$$



# Batch normalization at test time

- We cannot compute the mean and standard deviation on a batch of samples
- Instead we use a static mean and standard deviation were computed empirically during training
  - Compute the mean and standard deviation on the entire training set
  - **Estimate the mean and standard deviation using exponential weighted averages**

# Batch normalization (advantages)

- Reduces the dependence on weights initialization
- Improves gradient flow through the network
- Allows you to set a higher learning rate
- Slight regularization effect

# Batch normalization in pytorch

## BATCHNORM2D

```
CLASS torch.nn.BatchNorm2d(num_features, eps=1e-05, momentum=0.1, affine=True,  
    track_running_stats=True, device=None, dtype=None) [SOURCE]
```

Applies Batch Normalization over a 4D input (a mini-batch of 2D inputs with additional channel dimension) as described in the paper [Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift](#) .

$$y = \frac{x - \mathbb{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

The mean and standard-deviation are calculated per-dimension over the mini-batches and  $\gamma$  and  $\beta$  are learnable parameter vectors of size C (where C is the input size). By default, the elements of  $\gamma$  are set to 1 and the elements of  $\beta$  are set to 0. The standard-deviation is calculated via the biased estimator, equivalent to `torch.var(input, unbiased=False)`.

Also by default, during training this layer keeps running estimates of its computed mean and variance, which are then used for normalization during evaluation. The running estimates are kept with a default `momentum` of 0.1.

If `track_running_stats` is set to `False`, this layer then does not keep running estimates, and batch statistics are instead used during evaluation time as well.

# Other normalization methods

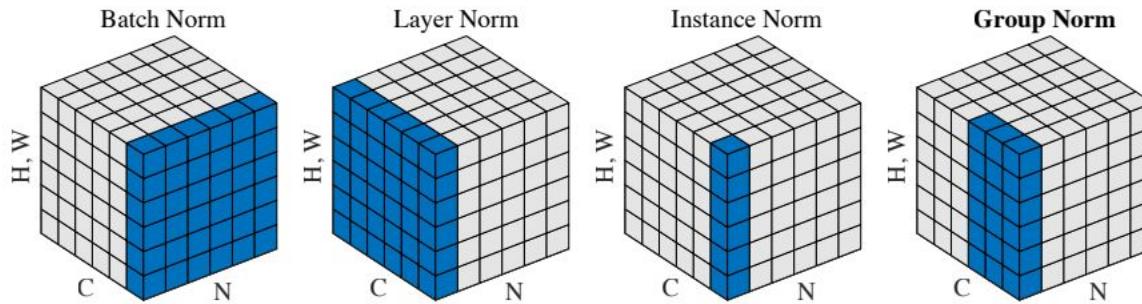
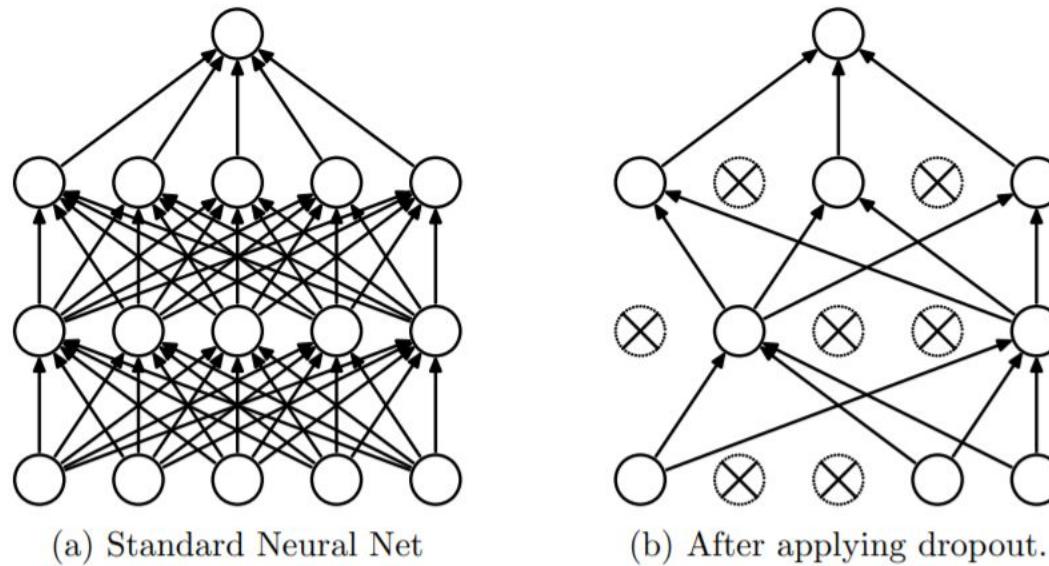


Figure 2. **Normalization methods.** Each subplot shows a feature map tensor, with  $N$  as the batch axis,  $C$  as the channel axis, and  $(H, W)$  as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

# Dropout

# Dropout (2014)

Randomly ***drop units*** (and their connections) from the neural network ***during training***



Srivastava, Nitish, et al. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 2014, 15.1: 1929-1958.

# Dropout

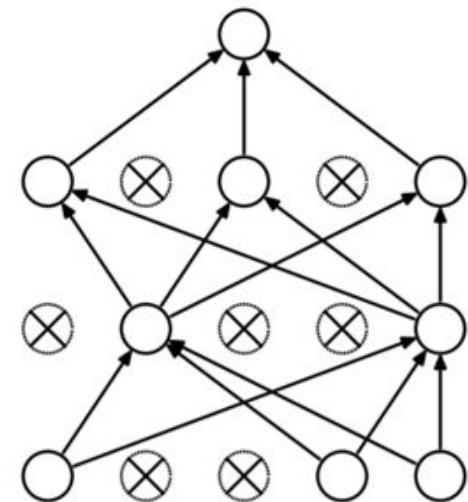
keep\_prop – probability of keeping a neuron; larger value, less dropout

```
# LINEAR -> RELU -> LINEAR -> RELU -> LINEAR -> SIGMOID
Z1 = np.dot(W1, X) + b1
A1 = relu(Z1)

# Step 1: initialize matrix D1 = np.random.rand(..., ...)
D1 = np.random.rand(A1.shape[0], A1.shape[1])
# Step 2: convert entries of D1 to 0 or 1 (using keep_prob as the threshold)
D1 = (D1 < keep_prob).astype(int)
# Step 3: shut down some neurons of A1
A1 = A1*D1

Z2 = np.dot(W2, A1) + b2
A2 = relu(Z2)

# Step 1: initialize matrix D2 = np.random.rand(..., ...)
D2 = np.random.rand(A2.shape[0], A2.shape[1])
# Step 2: convert entries of D2 to 0 or 1 (using keep_prob as the threshold)
D2 = (D2 < keep_prob).astype(int)
# Step 3: shut down some neurons of A2
A2 = A2*D2
```



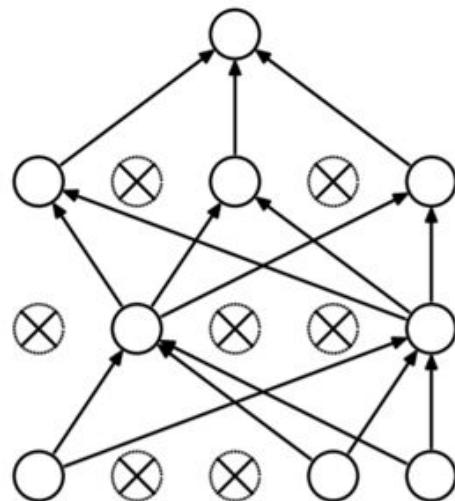
# Dropout visualization

# Dropout intuition

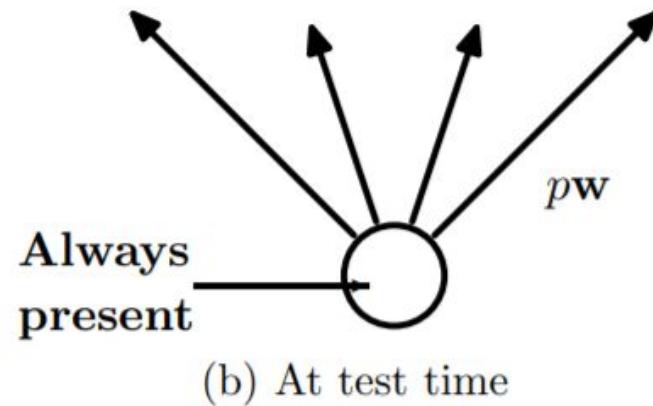
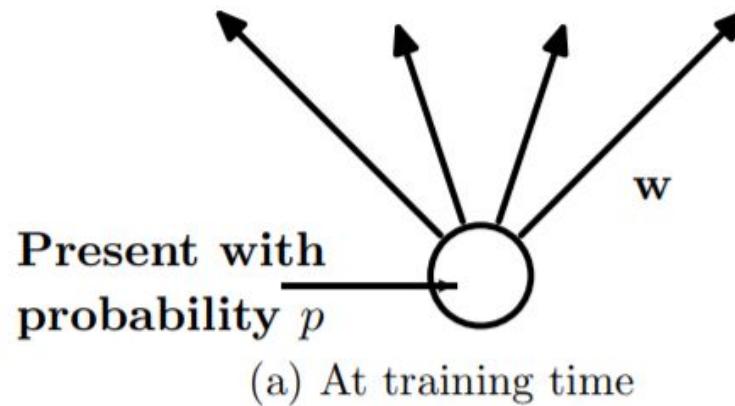
- At each iteration you actually modify your model:
  - train a different model that uses only a subset of your neurons
  - neurons thus become less sensitive to the activation of one other specific neuron (others neuron might be shut down at any time)

# Dropout intuition

- Train a large ensemble of models (with shared parameters)
- Forces the model to have a redundant representation



# Dropout at test time



# Dropout at test time

- The neurons are always turned on
- Scale the activation for each neuron
  - output at test time = expected output at training time

# Dropout at test time

keep\_prop – probability of keeping a neuron; larger value, less dropout

```
# LINEAR -> RELU -> LINEAR -> RELU -> LINEAR -> SIGMOID
Z1 = np.dot(w1, X) + b1
A1 = relu(Z1)

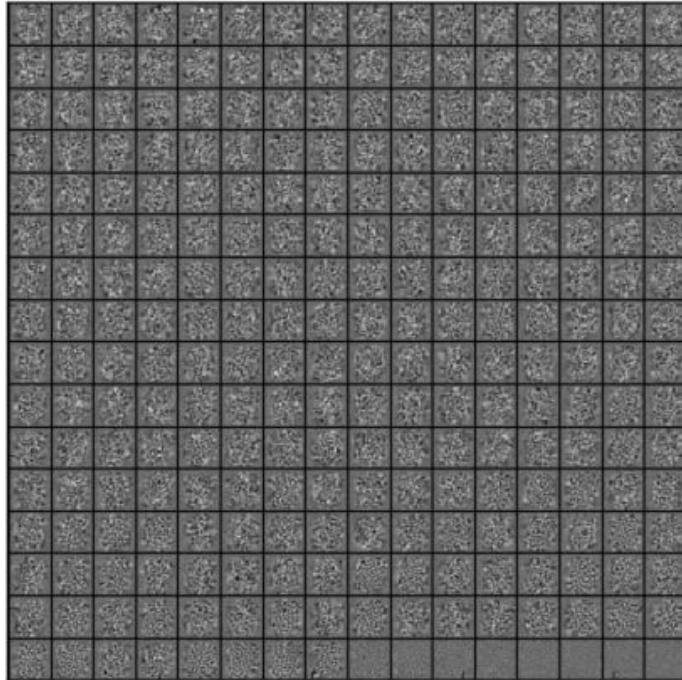
D1 = np.random.rand(A1.shape[0], A1.shape[1])           # Step 1: initialize matrix D1 = np.random.rand(..., ...)
D1 = (D1 < keep_prob).astype(int)                     # Step 2: convert entries of D1 to 0 or 1 (using keep_prob as the threshold)
A1 = A1*D1                                           # Step 3: shut down some neurons of A1
A1 = A1/keep_prob                                     # Step 4: scale the value of neurons that haven't been shut down

Z2 = np.dot(w2, A1) + b2
A2 = relu(Z2)

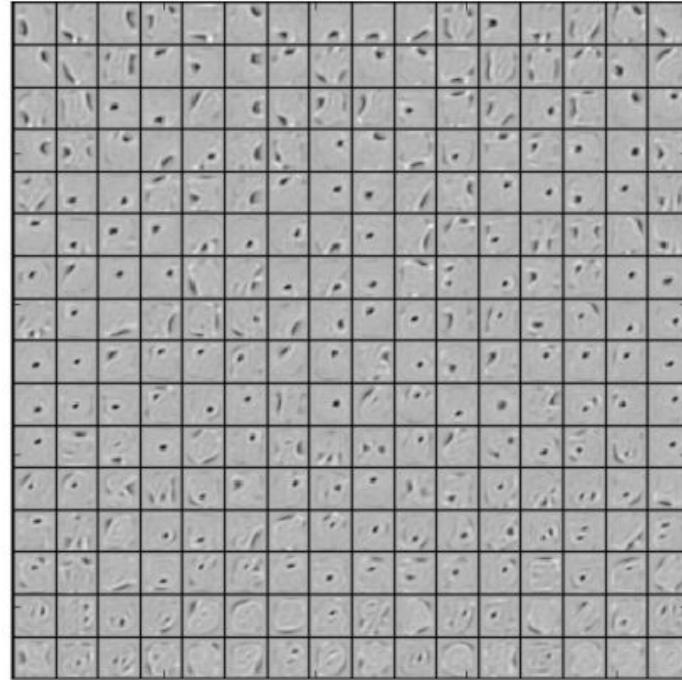
D2 = np.random.rand(A2.shape[0], A2.shape[1])           # Step 1: initialize matrix D2 = np.random.rand(..., ...)
D2 = (D2 < keep_prob).astype(int)                     # Step 2: convert entries of D2 to 0 or 1 (using keep_prob as the threshold)
A2 = A2*D2                                           # Step 3: shut down some neurons of A2
A2 = A2/keep_prob                                     # Step 4: scale the value of neurons that haven't been shut down

Z3 = np.dot(w3, A2) + b3
A3 = sigmoid(Z3)
```

# Dropout in practice



(a) Without dropout



(b) Dropout with  $p = 0.5$ .

Figure 7: Features learned on MNIST with one hidden layer autoencoders having 256 rectified linear units.

A hidden unit cannot rely on other specific units to correct its mistakes. It must perform well in a wide variety of different contexts provided by the other hidden units.

# Dropout in torch

## DROPOUT

---

CLASS `torch.nn.Dropout(p=0.5, inplace=False)` [\[SOURCE\]](#)

During training, randomly zeroes some of the elements of the input tensor with probability `p` using samples from a Bernoulli distribution. Each channel will be zeroed out independently on every forward call.

This has proven to be an effective technique for regularization and preventing the co-adaptation of neurons as described in the paper [Improving neural networks by preventing co-adaptation of feature detectors](#).

Furthermore, the outputs are scaled by a factor of  $\frac{1}{1-p}$  during training. This means that during evaluation the module simply computes an identity function.

# Data augmentation



Crop



Symmetry



Rotation



Scale



Original



Noise



Hue



Obstruction



Blur

# Data augmentation

Image



Mixup



Cutout



CutMix



# Training a neural network

*Optimization algorithms*

# Parameters update – gradient descent

Vanilla gradient descent:

1. Forward propagation through the network
2. Compute loss
3. Back-propagate to compute gradients
4. Update the parameters using the gradient

# Parameters update – gradient descent

1. Batch gradient descent
  - “classical” implementation of gradient descent
  - Use vectorization and process the entire training set at the same time
2. Split the training set into *mini-batches*

# Parameters update – gradient descent

$$X = [ X^{(1)} \ X^{(2)} \ \dots \ X^{(1000)} \ X^{(1001)} \ \dots \ X^{(2000)} \ \dots \ X^{(M)} ]$$

$$Y = [ Y^{(1)} \ Y^{(2)} \ \dots \ Y^{(1000)} \ Y^{(1001)} \ \dots \ Y^{(2000)} \ \dots \ Y^{(M)} ]$$

$$X - (n_x, M)$$

$$Y - (1, M)$$

Minibatch  $t$ :  $X^{\{t\}}, Y^{\{t\}}$

Example  $i$ :  $X^{(i)}, Y^{(i)}$

# Parameters update – gradient descent

- “Classical” gradient descent
  - Use vectorization and process the entire training set at the same time
- Split the training set into *mini-batches*

```
while True:  
    batch = sample_batch(data)  
    W_gradient = compute_gradient(loss_func,  
batch, W)  
    # update parameters  
    W += -lr*W_gradient
```

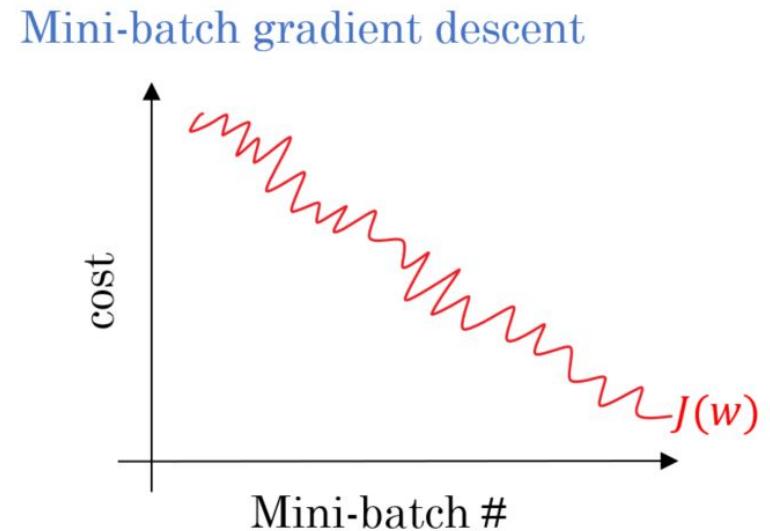
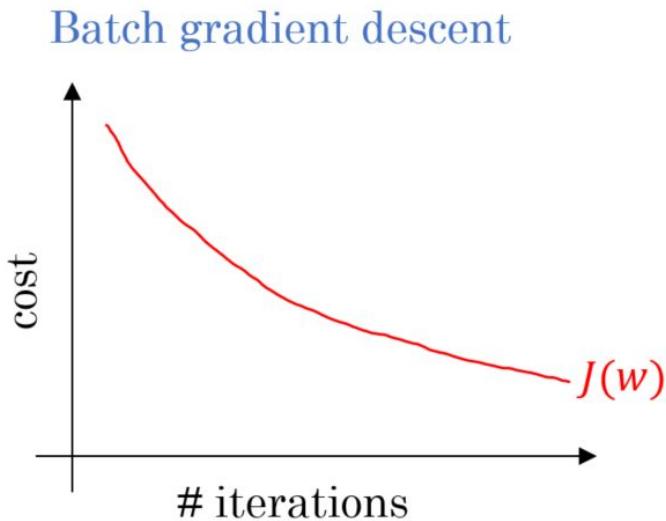
# Mini-batch gradient descent

- *Epoch*: single pass through the entire training set

```
270/270 [=====] - 10s 37ms/step - loss: 0.0451 - mae_metric: 32.8503
Epoch 37/400
270/270 [=====] - 10s 37ms/step - loss: 0.0555 - mae_metric: 37.9888
Epoch 38/400
270/270 [=====] - 10s 37ms/step - loss: 0.0557 - mae_metric: 39.1687
Epoch 39/400
270/270 [=====] - 10s 37ms/step - loss: 0.0471 - mae_metric: 31.8123
Epoch 40/400
270/270 [=====] - 10s 36ms/step - loss: 0.0293 - mae_metric: 32.6505
Epoch 41/400
270/270 [=====] - 10s 37ms/step - loss: 0.0412 - mae_metric: 32.2138
Epoch 42/400
270/270 [=====] - 10s 37ms/step - loss: 0.0601 - mae_metric: 37.9935
Epoch 43/400
220/270 [======>.....] - ETA: 1s - loss: 0.0550 - mae_metric: 33.8031
```

# Size of the mini-batches

- Stochastic gradient descent ( $m=1$ )
- Mini-batch gradient descent
- Batch gradient descent ( $m = M$ )



# Size of the mini-batches

- Stochastic gradient descent ( $m=1$ )
  - no vectorization, lose speedup
- Mini-batch gradient descent
  - Fastest learning: use vectorization and doesn't take too much time to update the weights
  - $2^6, 2^7, \dots, 2^{10}$
- Batch gradient descent ( $m = M$ )
  - Too much time per iteration

# Parameters update – mini-batch gradient descent

**foreach** batch  $t$ :

1. Forward propagation through the network
2. Compute loss
3. Backpropagate to compute gradients
4. Update the parameters using the gradient

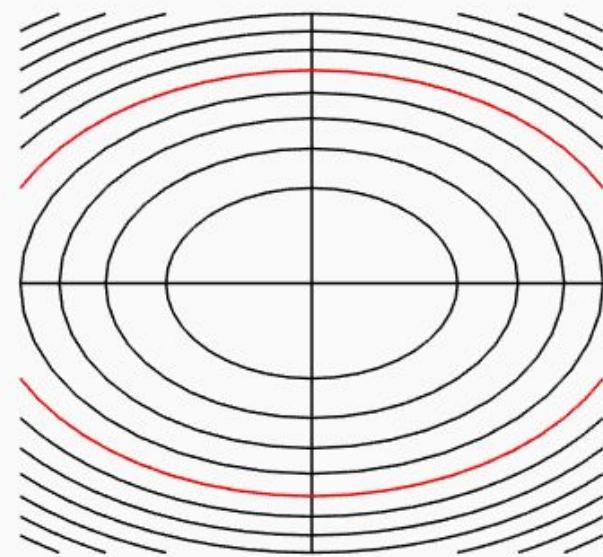
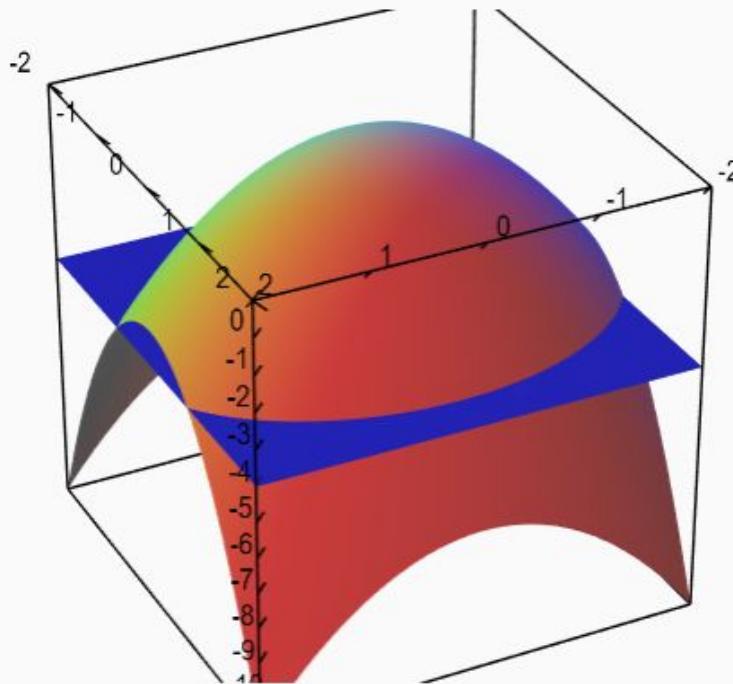
# Computer Vision and Deep Learning

Lecture 6

# Today's agenda

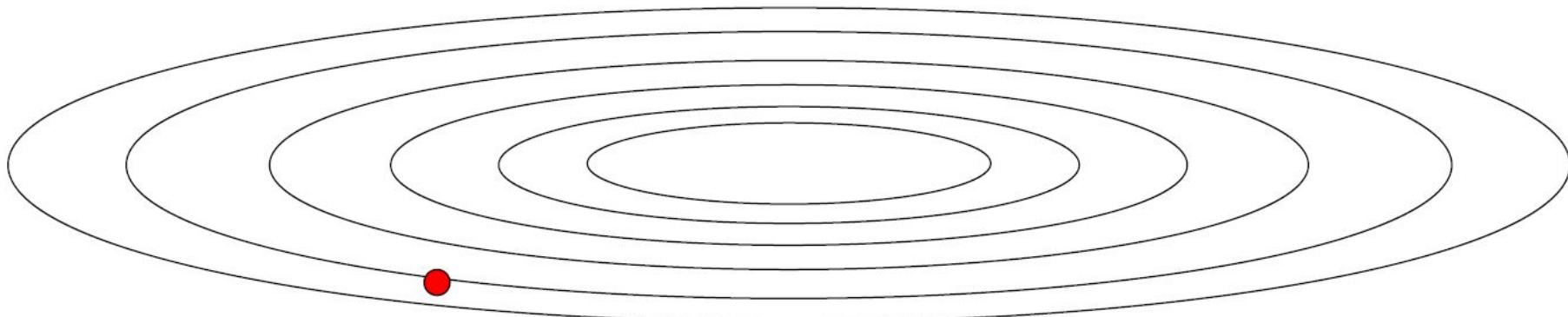
- Optimization algorithms
- Convolutional neural networks:
  - History
  - Case studies
- How to read a research paper?

# Level sets of a surface



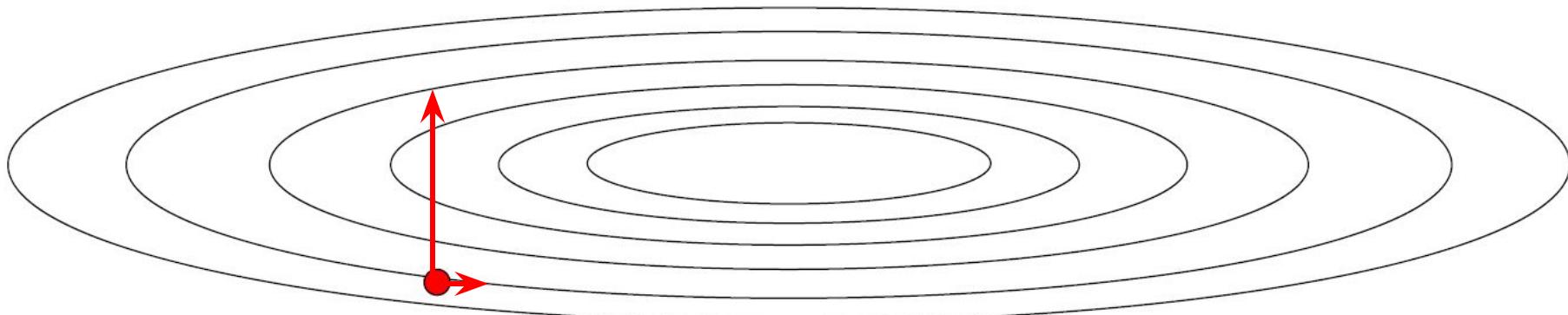
# Problems with gradient descent

What if loss changes abruptly on one direction and slower in the other direction?



# Problems with gradient descent

What if loss changes abruptly on one direction and slower in the other direction?

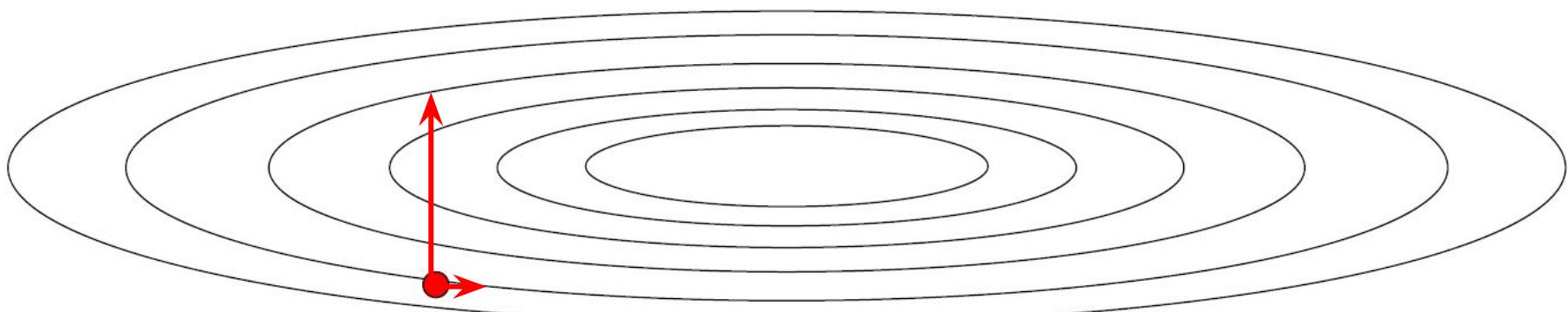


# Problems with gradient descent

What if loss changes abruptly on one direction and slower in the other direction?

- Small gradient horizontally
- Large gradient vertically

Slow progress along the horizontal direction, jitter along the vertical direction (the steeper one)



# Gradient descent with momentum

Compute an exponentially weighted average of the gradients and use this average to update the parameters of the network

# Gradient descent with momentum

## Update rule

Gradient descent update:

$$W = W - \alpha \cdot dW$$



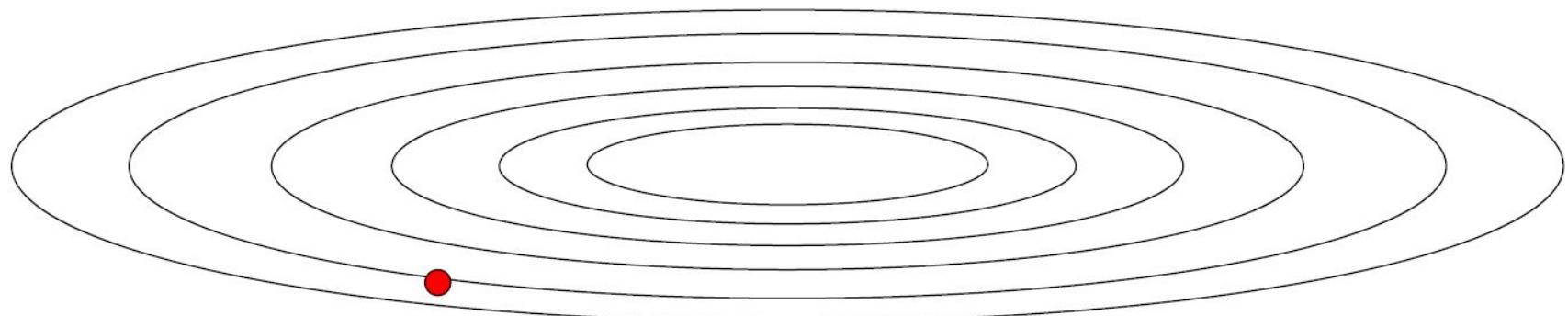
Momentum update:

$$\begin{aligned} v_{dW} &= \beta \cdot v_{dW} + (1 - \beta) \cdot dW \\ W &= W - \alpha \cdot v_{dW} \end{aligned}$$

$\beta$  – hyper-parameter; common value 0.9

# Gradient descent with momentum

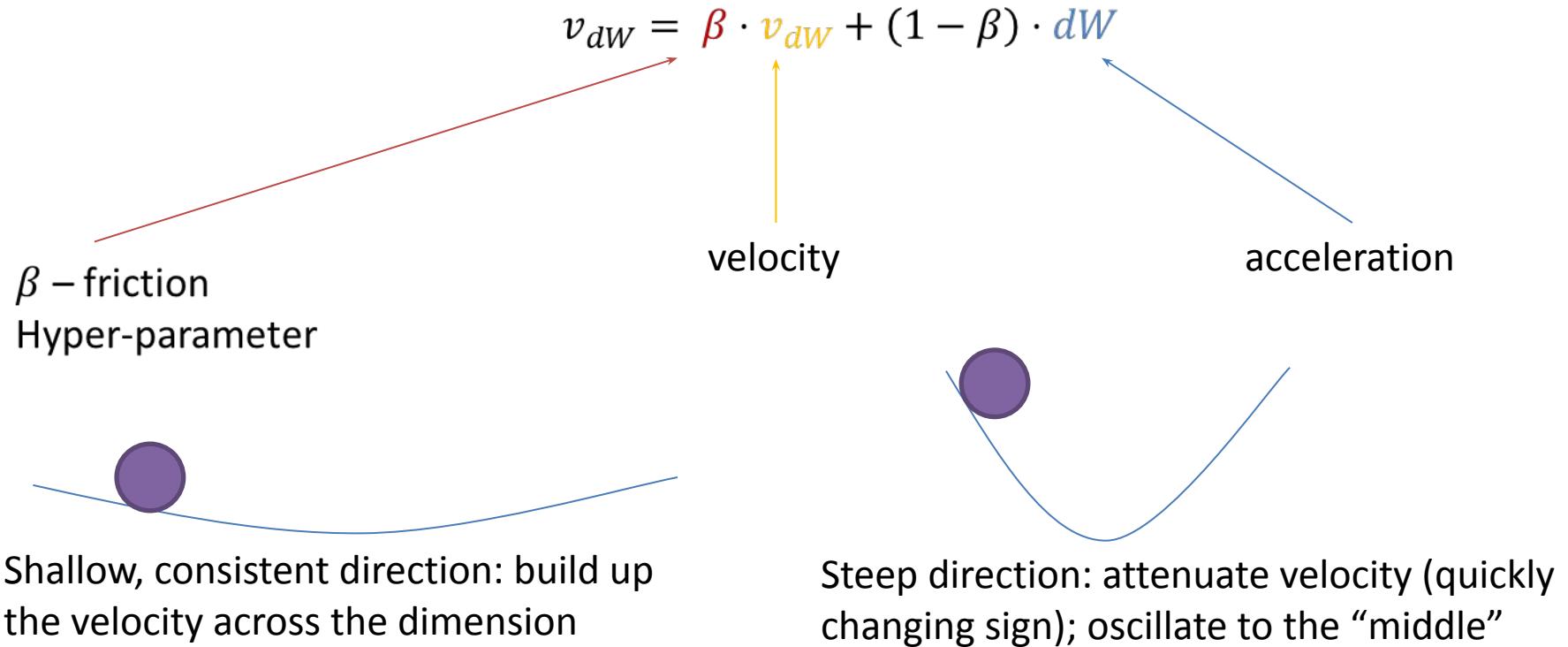
Take more straightforward part (damp oscillations)



Slower learning vertically, faster learning horizontally

# Gradient descent with momentum

- Intuition
  - Ball rolling down the loss function with friction
  - Gradient: the force the ball is feeling ( $F = ma$ )



# Gradient descent with momentum

## Update rule - variant

- 

$$v_{dW} = \beta \cdot v_{dW} + (1 - \beta)dW$$



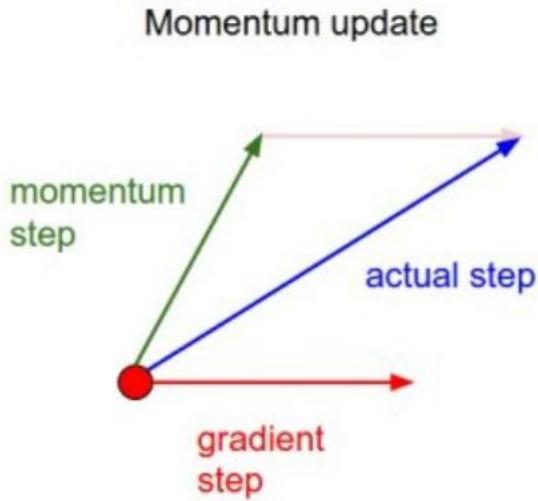
$$v_{dW} = \beta \cdot v_{dW} + dW$$

$$W = W - \alpha \cdot v_{dW}$$

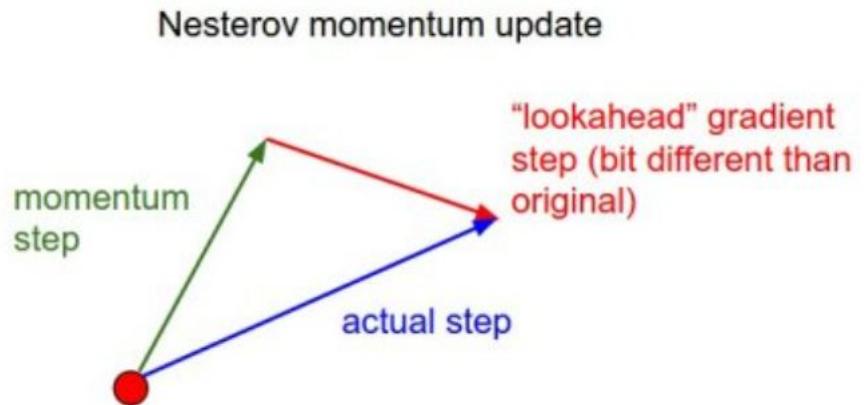
$\beta$  – hyper-parameter; common value 0.9

# Nesterov momentum

$$v_{dW} = \beta \cdot v_{dW} + dW$$



$$v_{t+1} = \mu v_t - \eta \nabla l(\theta)$$
$$\theta_{t+1} = \theta_t + v_{t+1}$$



$$v_{t+1} = \mu v_t - \eta \nabla l(\theta + \mu v_t)$$
$$\theta_{t+1} = \theta_t + v_{t+1}$$

Image source: <https://cs231n.github.io/neural-networks-3/#sgd>

<https://dominikschmidt.xyz/nesterov-momentum/>

# AdaGrad

- $$cache += dW^2$$
$$W = W - \alpha \cdot dW / (\sqrt{cache} + 1e^{-7})$$

cache – same size as W

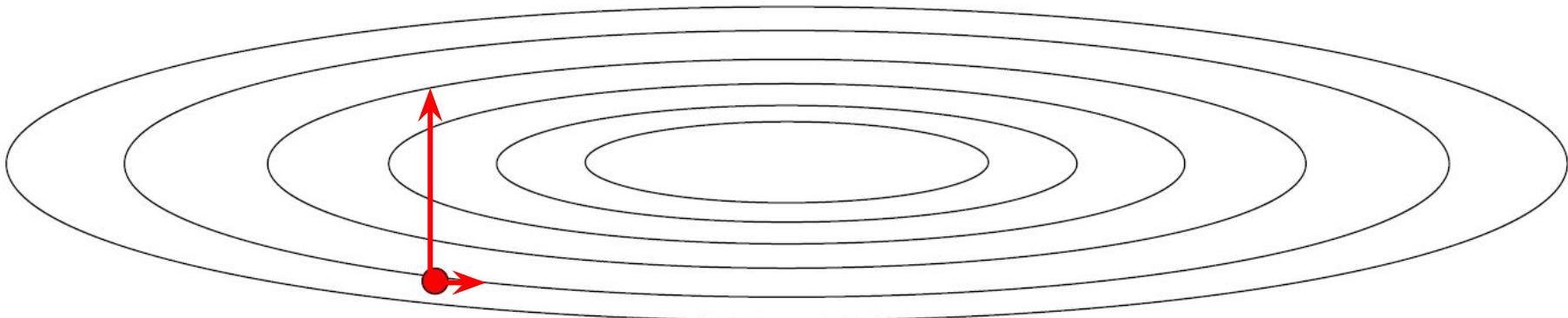
Element wise scaling of the gradient based on the “historical” sum of squares in each dimension

Per parameter adaptive learning rate

# AdaGrad

$$cache += dW^2$$

$$W = W - \alpha \cdot dW / (\sqrt{cache} + 1e^{-7})$$



## Equalizing effect

- Larger learning rate on “shallow” directions than on steeper directions

# RMSProp

Adagrad

$$\begin{aligned} cache &+= dW^2 \\ W &= W - \alpha \cdot dW / (\sqrt{cache} + 1e^{-7}) \end{aligned}$$



RMSProp

$$\begin{aligned} cache &= \beta_2 \cdot cache + (1 - \beta_2) \cdot dW^2 \\ W &= W - \alpha \cdot dW / (\sqrt{cache} + 1e^{-7}) \end{aligned}$$

Introduced by Geoffrey Hinton on a lecture on Coursera:

- [17] Duchi J, Hazan E and Singer Y 2011 *The Journal of Machine Learning Research* 12 2641  
[15] Tieleman T and Hinton E 2012 Lecture 6.5 - rmsprop, COURSERA: Neural networks for machine learning.

# Adam

- Combine RMSProp with momentum

momentum  
(velocity)

$$\begin{aligned} m &= \beta_1 \cdot m + (1 - \beta_1) \cdot dW \\ v &= \beta_2 \cdot v + (1 - \beta_2) \cdot dW^2 \\ W &= W - \alpha \cdot m / (\sqrt{v} + 1e^{-7}) \end{aligned}$$

RMSProp like

RMSProp

$$\begin{aligned} cache &= \beta_2 \cdot cache + (1 - \beta_2) \cdot dW^2 \\ W &= W - \alpha \cdot dW / (\sqrt{cache} + 1e^{-7}) \end{aligned}$$

# Adam

- Combine RMSProp with momentum

momentum  
(velocity)

$$\begin{aligned} m &= \beta_1 \cdot m + (1 - \beta_1) \cdot dW \\ v &= \beta_2 \cdot v + (1 - \beta_2) \cdot dW^2 \\ W &= W - \alpha \cdot m / (\sqrt{v} + 1e^{-7}) \end{aligned}$$

RMSProp like

RMSProp

$$\begin{aligned} cache &= \beta_2 \cdot cache + (1 - \beta_2) \cdot dW^2 \\ W &= W - \alpha \cdot dW / (\sqrt{cache} + 1e^{-7}) \end{aligned}$$

Noisy gradients

# Adam

- Combine RMSProp with momentum  
for  $t$  in range(0, iterations):

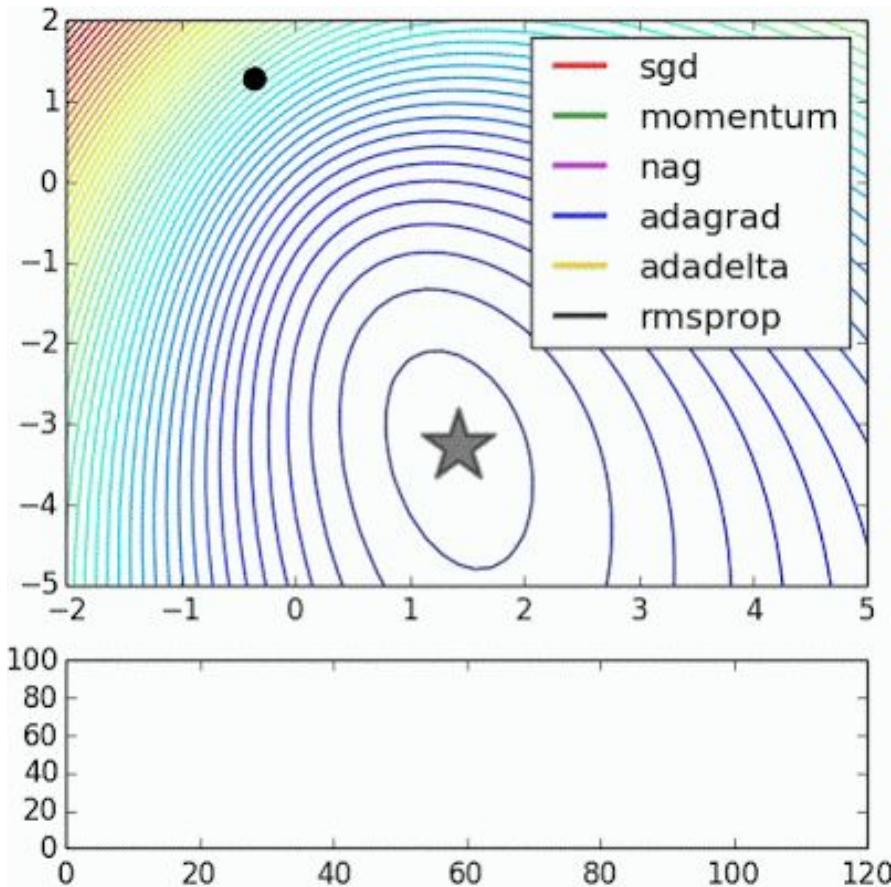
$$m = \beta_1 \cdot m + (1 - \beta_1) \cdot dW$$

$$v = \beta_2 \cdot v + (1 - \beta_2) \cdot dW^2$$

$$\begin{cases} m = m / (1 - \beta_1^t) \\ v = v / (1 - \beta_2^t) \end{cases}$$

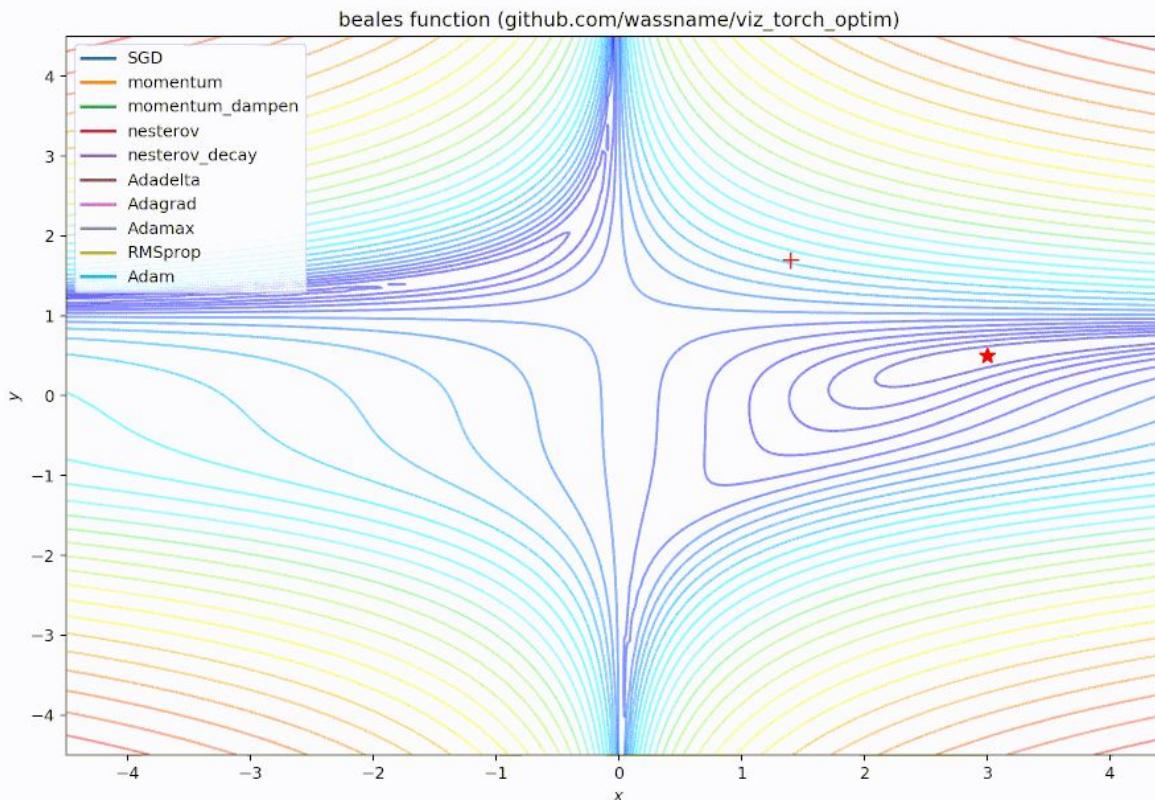
Bias correction

$$W = -\alpha \cdot m / (\sqrt{v} + 1e^{-7})$$



<http://www.denizyuret.com/2015/03/alec-radford-animations-for.html>

# Optimization algorithms



[https://awesomeopensource.com/project/3springs/viz\\_torch\\_optim](https://awesomeopensource.com/project/3springs/viz_torch_optim)

# Optimizers in torch

## TORCH.OPTIM ↴

`torch.optim` is a package implementing various optimization algorithms. Most commonly used methods are already supported, and the interface is general enough, so that more sophisticated ones can be also easily integrated in the future.

### How to use an optimizer

To use `torch.optim` you have to construct an optimizer object, that will hold the current state and will update the parameters based on the computed gradients.

### Constructing it

To construct an `Optimizer` you have to give it an iterable containing the parameters (all should be `Variable`s) to optimize. Then, you can specify optimizer-specific options such as the learning rate, weight decay, etc.

Example:

```
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
optimizer = optim.Adam([var1, var2], lr=0.0001)
```

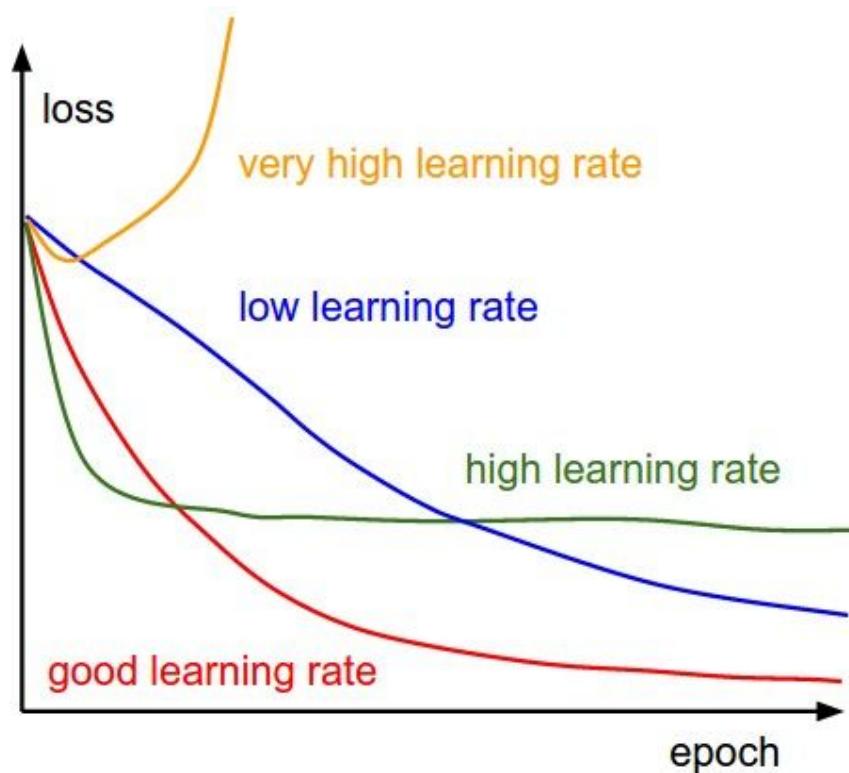
# Optional additional reading

- <https://ruder.io/optimizing-gradient-descent/index.html#otherrecentoptimizers>
- <https://www.youtube.com/watch?v=k8fTYJPd3I>
- [https://www.youtube.com/watch?v= e-LFe\\_igno](https://www.youtube.com/watch?v= e-LFe_igno)
- <https://www.coursera.org/lecture/deep-neural-network/the-problem-of-local-optima-RFANA>

# Training a neural network

*Learning rate scheduling*

# Learning rate



# Learning rate decay

- Learning rate decay over time!
  - Use a larger learning rate at the beginning and progressively reduce the learning rate
  - $t$  epoch,  $k$  – decay rate
- Step decay:
  - reduce learning rate by half after some epochs
- Exponential decay

$$\alpha = \alpha_0 \cdot e^{-kt}$$

- $1/t$  decay

$$\alpha = \alpha_0 / (1 + kt)$$

# keras optimizers schedules

## How to adjust learning rate

`torch.optim.lr_scheduler` provides several methods to adjust the learning rate based on the number of epochs.

`torch.optim.lr_scheduler.ReduceLROnPlateau` allows dynamic learning rate reducing based on some validation measurements.

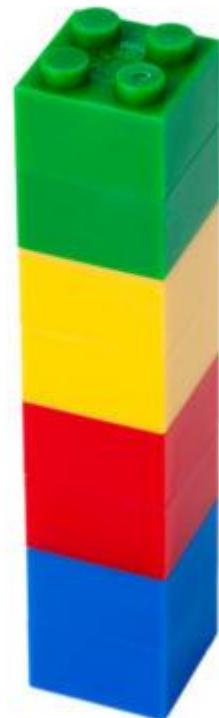
```
model = [Parameter(torch.randn(2, 2, requires_grad=True))]
optimizer = SGD(model, 0.1)
scheduler = ExponentialLR(optimizer, gamma=0.9)

for epoch in range(20):
    for input, target in dataset:
        optimizer.zero_grad()
        output = model(input)
        loss = loss_fn(output, target)
        loss.backward()
        optimizer.step()
    scheduler.step()
```

# Previously

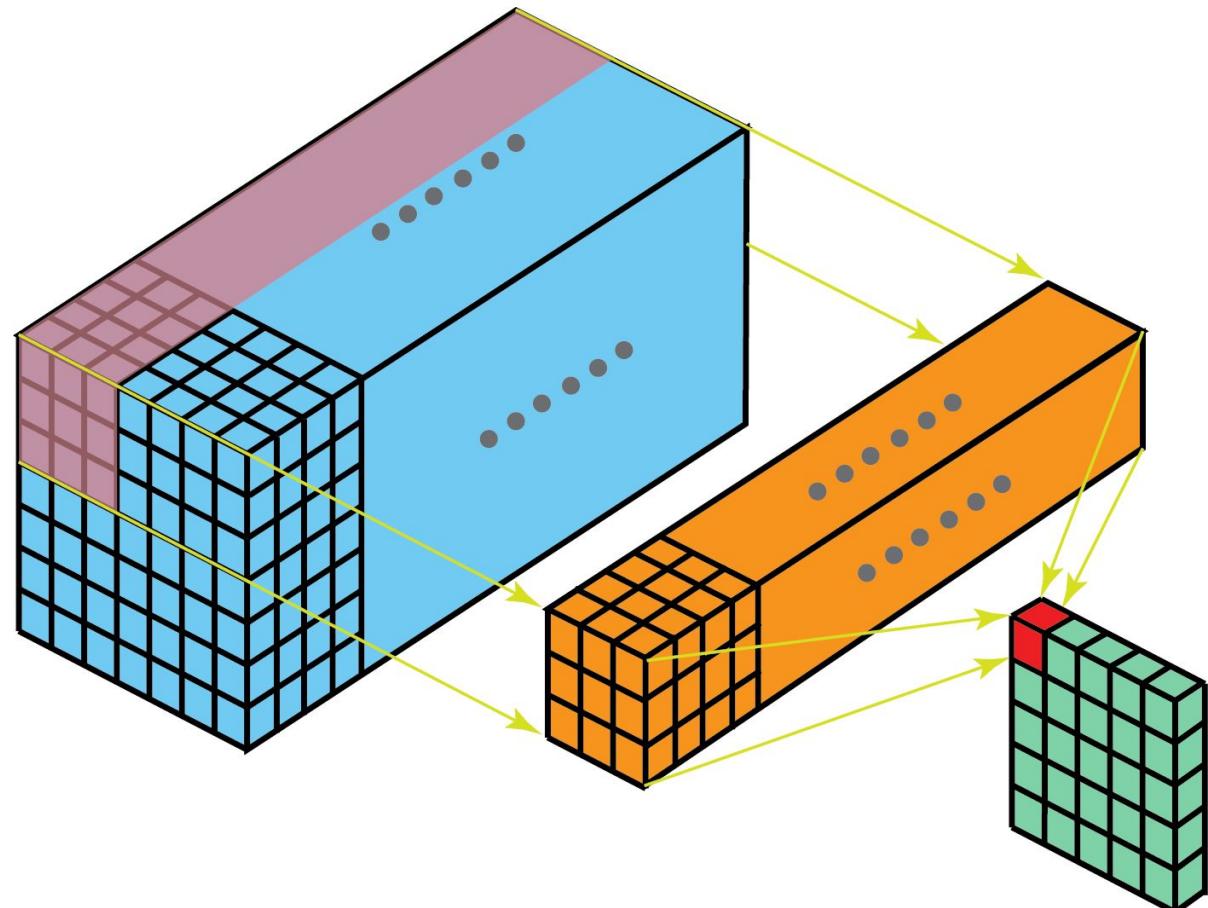
Convolutional neural networks

- Convolutional neural networks
  - Convolutional layers
  - Pooling layers
  - Fully connected layers



# Convolution layers

Shared parameters  
Sparsity of connections



# Pooling layers

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

Max Pool  
→

Filter - (2 x 2)  
Stride - (2, 2)

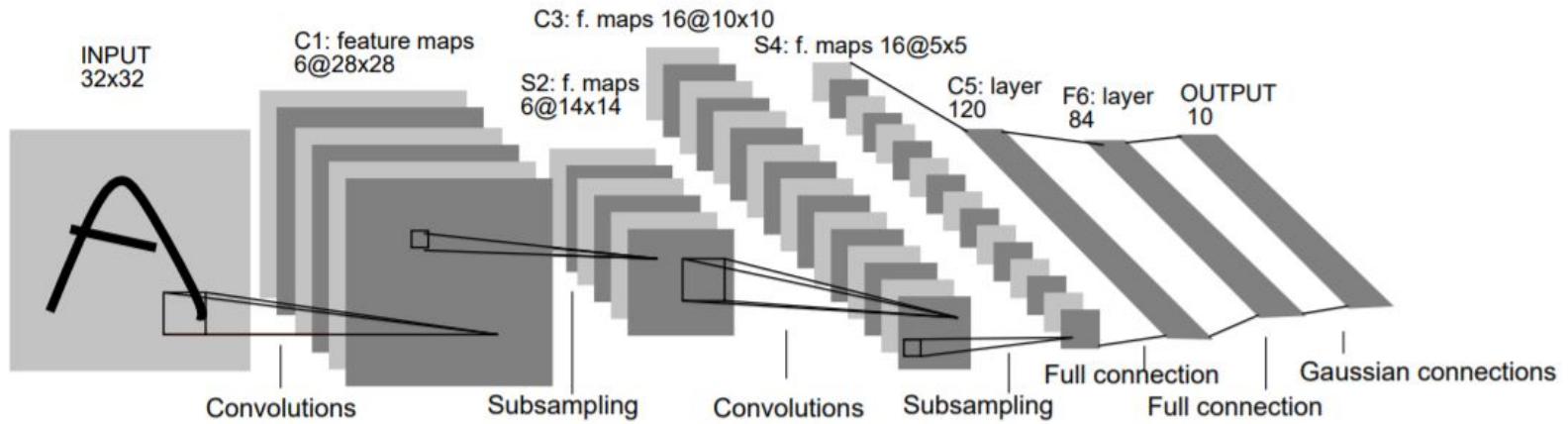
9	7
8	6

# Previously

Convolutional neural networks

- Training a neural network
  - Good initialization
  - Batch normalization
  - Regularization
  - Batch training
    - Stochastic gradient descent
    - Mini-batch gradient descent
    - Batched gradient descent
  - Optimizers

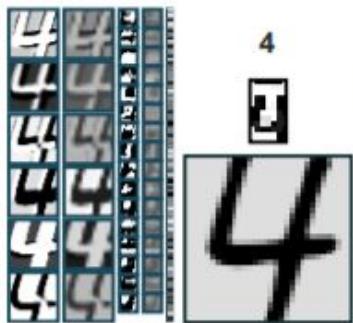
# LeNet 5



Conv filters have size 5x5 and are applied at stride 1

Subsampling (Pooling) layers have size 2x2 and are applied at stride 2  
[CONV-POOL-CONV-POOL-FC-FC]

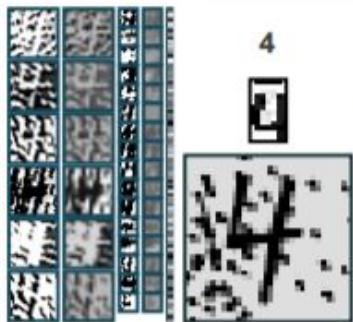
# LeNet 5



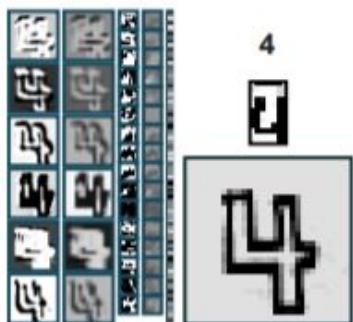
4



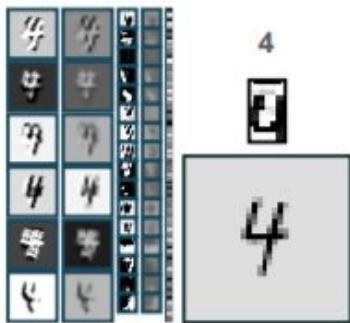
4



4



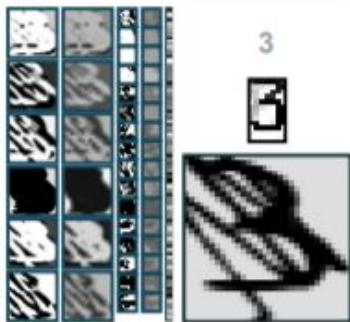
4



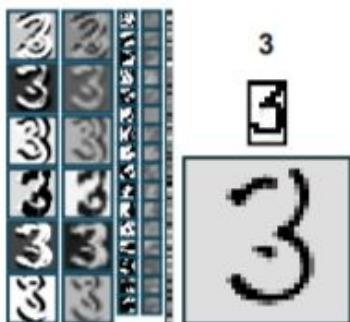
4



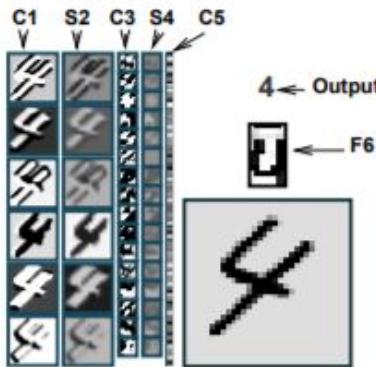
3



3



3



4 ← Output

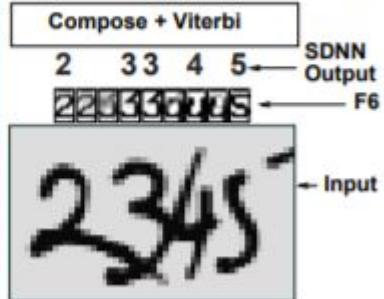


F6 ←



2345

Answer



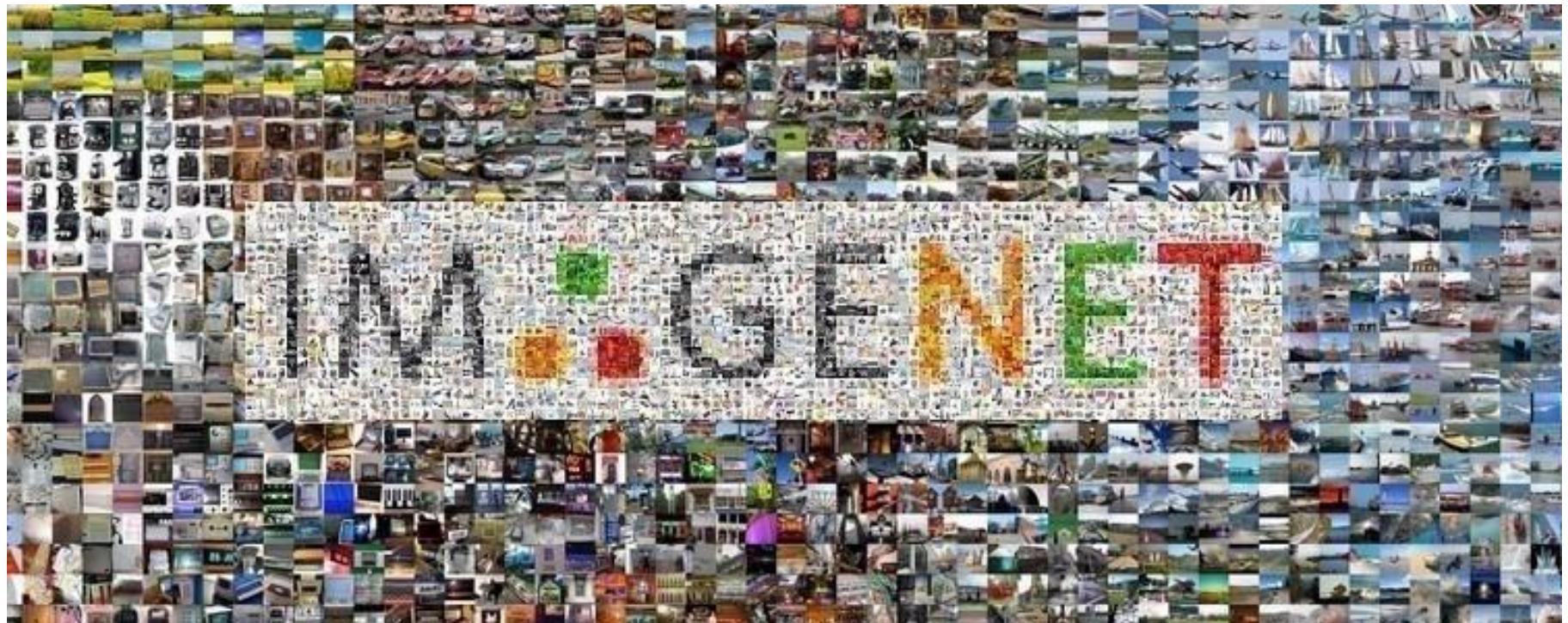
Compose + Viterbi

SDNN Output

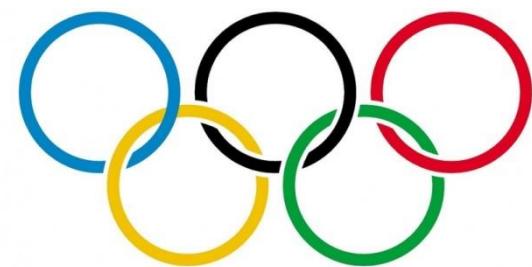
F6

Input

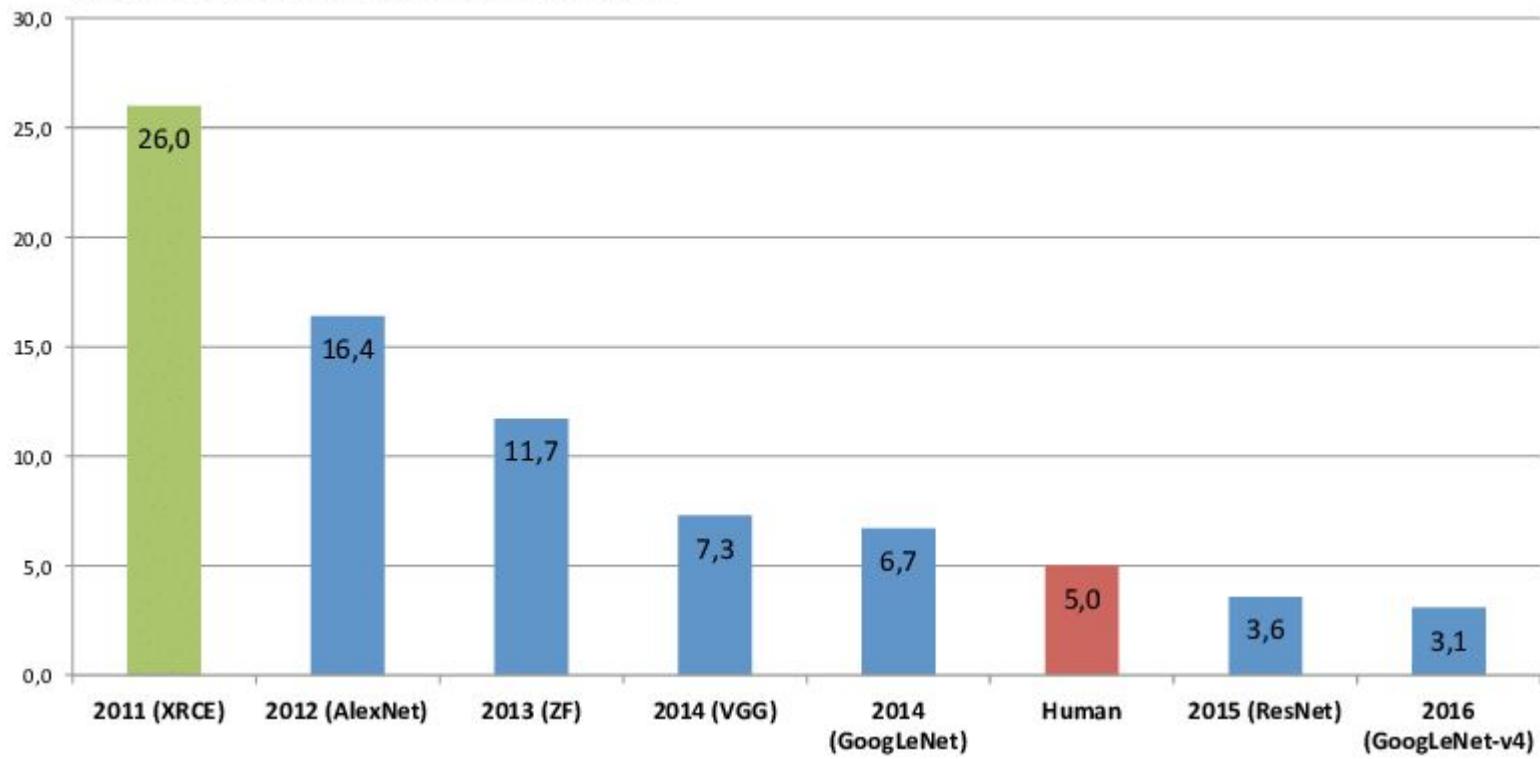
# 2012



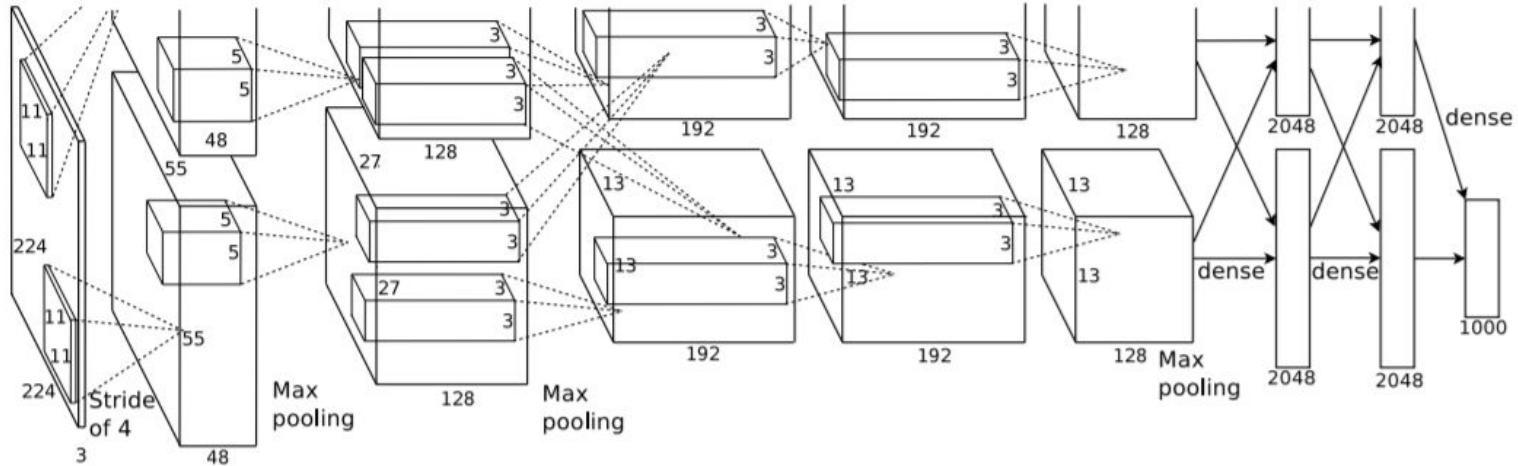
14,197,122 images  
1000 categories



### ImageNet Classification Error (Top 5)



# Alexnet, 2012



A single GTX 580 GPU has only 3GB of memory, which limits the maximum size of the networks that can be trained on it.

# Alexnet, 2012

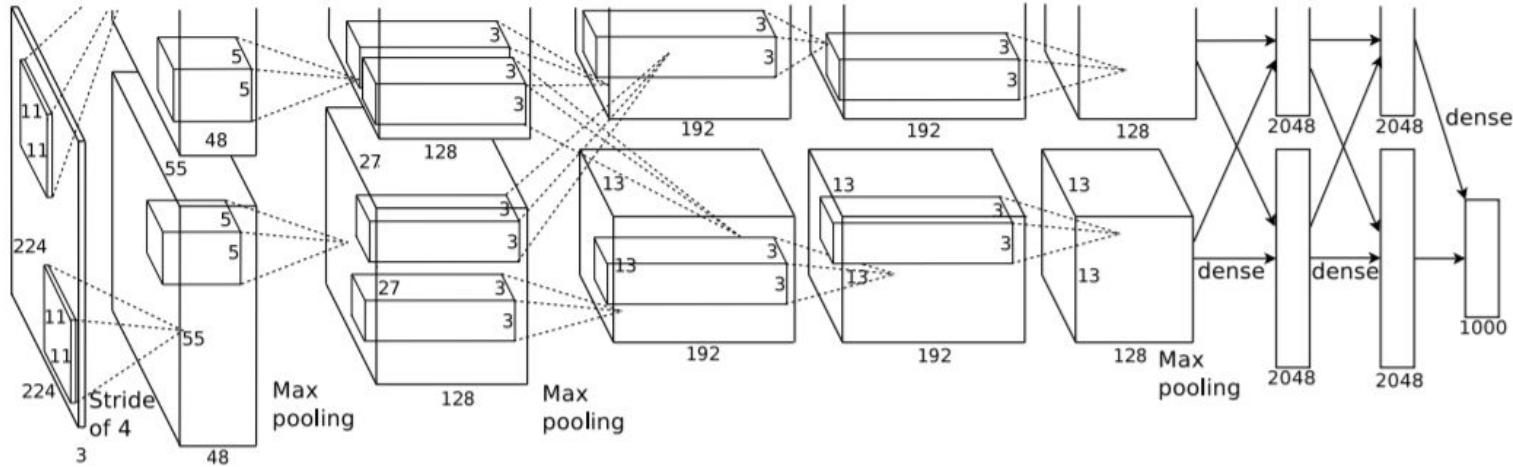


Image size: (227, 227, 3)

First layer: 96 filter of size 11x11

$\beta$  - hyper-parameter; common value 0.9

**What is the output volume size  
and the number of parameter in  
this layer?**

# Alexnet, 2012

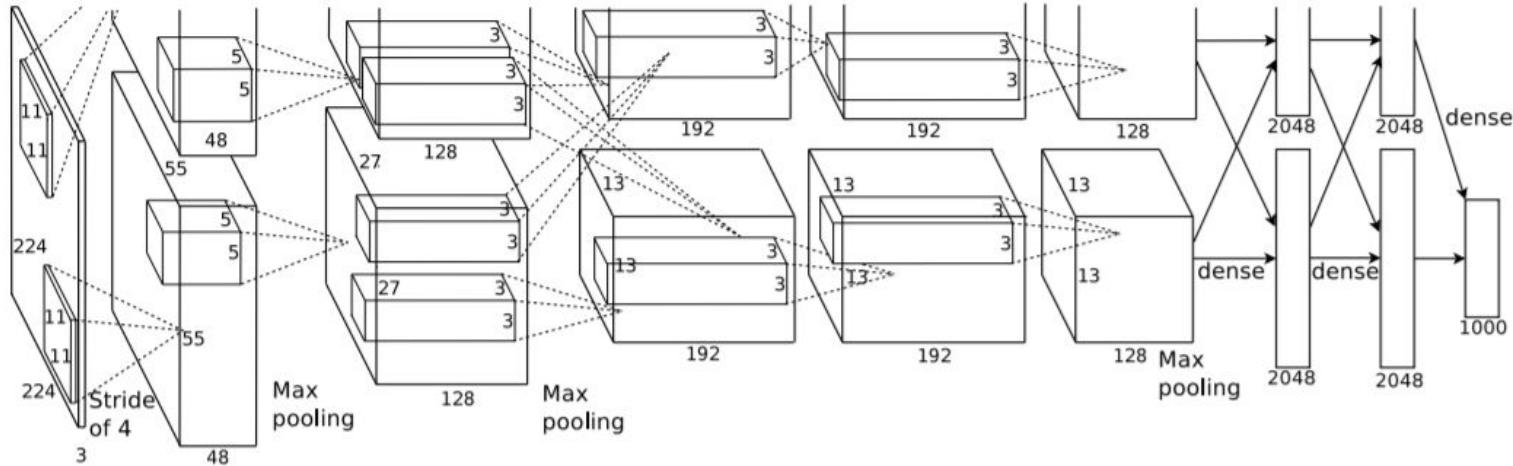


Image size: (227, 227, 3)

First layer: 96 filter of size 11x11

**Output volume: 55x55x96**

**Parameters:  $(11 \times 11 \times 3) \times 96 \sim 35K$**

$\beta$  - hyper-parameter; common value 0.9

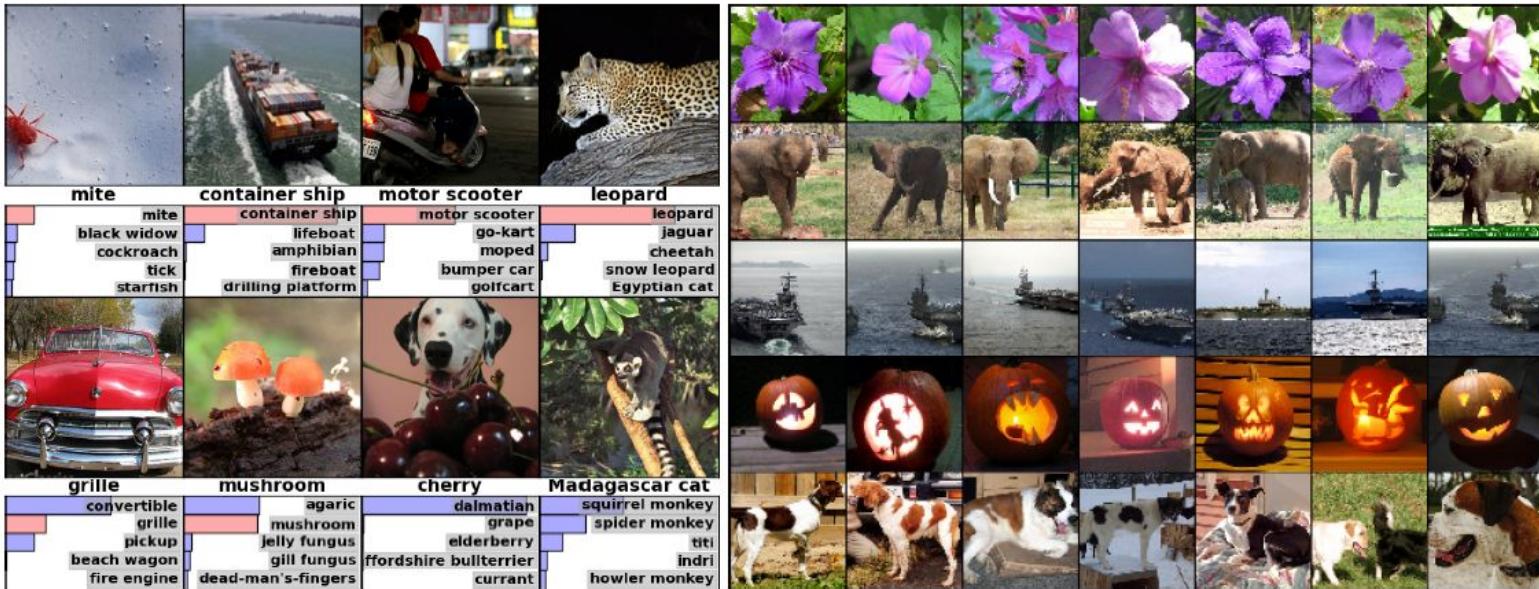
# Alexnet

AlexNet Network - Structural Details													
Input			Output			Layer	Stride	Pad	Kernel size	in	out	# of Param	
227	227	3	55	55	96	conv1	4	0	11	11	3	96	34944
55	55	96	27	27	96	maxpool1	2	0	3	3	96	96	0
27	27	96	27	27	256	conv2	1	2	5	5	96	256	614656
27	27	256	13	13	256	maxpool2	2	0	3	3	256	256	0
13	13	256	13	13	384	conv3	1	1	3	3	256	384	885120
13	13	384	13	13	384	conv4	1	1	3	3	384	384	1327488
13	13	384	13	13	256	conv5	1	1	3	3	384	256	884992
13	13	256	6	6	256	maxpool5	2	0	3	3	256	256	0
						fc6			1	1	9216	4096	37752832
						fc7			1	1	4096	4096	16781312
						fc8			1	1	4096	1000	4097000
<b>Total</b>											<b>62,378,344</b>		

# Alexnet

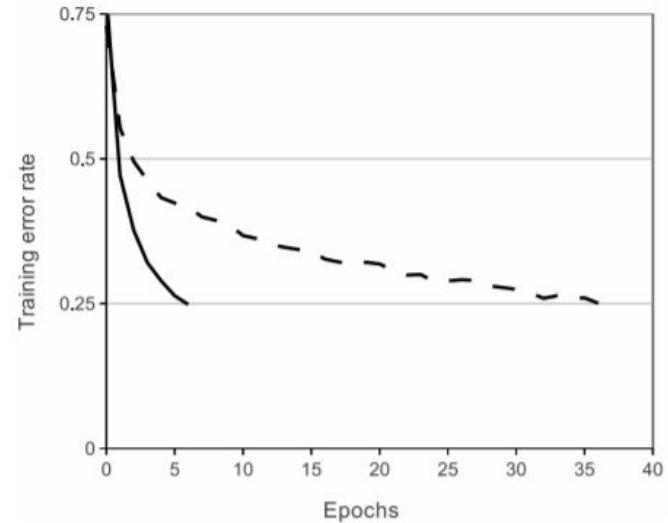
Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
<i>SIFT + FVs [7]</i>	—	—	26.2%
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	<b>16.4%</b>
1 CNN*	39.0%	16.6%	—
7 CNNs*	36.7%	15.4%	<b>15.3%</b>

Table 2: Comparison of error rates on ILSVRC-2012 validation and test sets. In *italics* are best results achieved by others. Models with an asterisk\* were “pre-trained” to classify the entire ImageNet 2011 Fall release. See Section 6 for details.



# Alexnet – key features

- First use of ReLU
- Overlapped max pooling
- Used normalization layers
  - Not used anymore



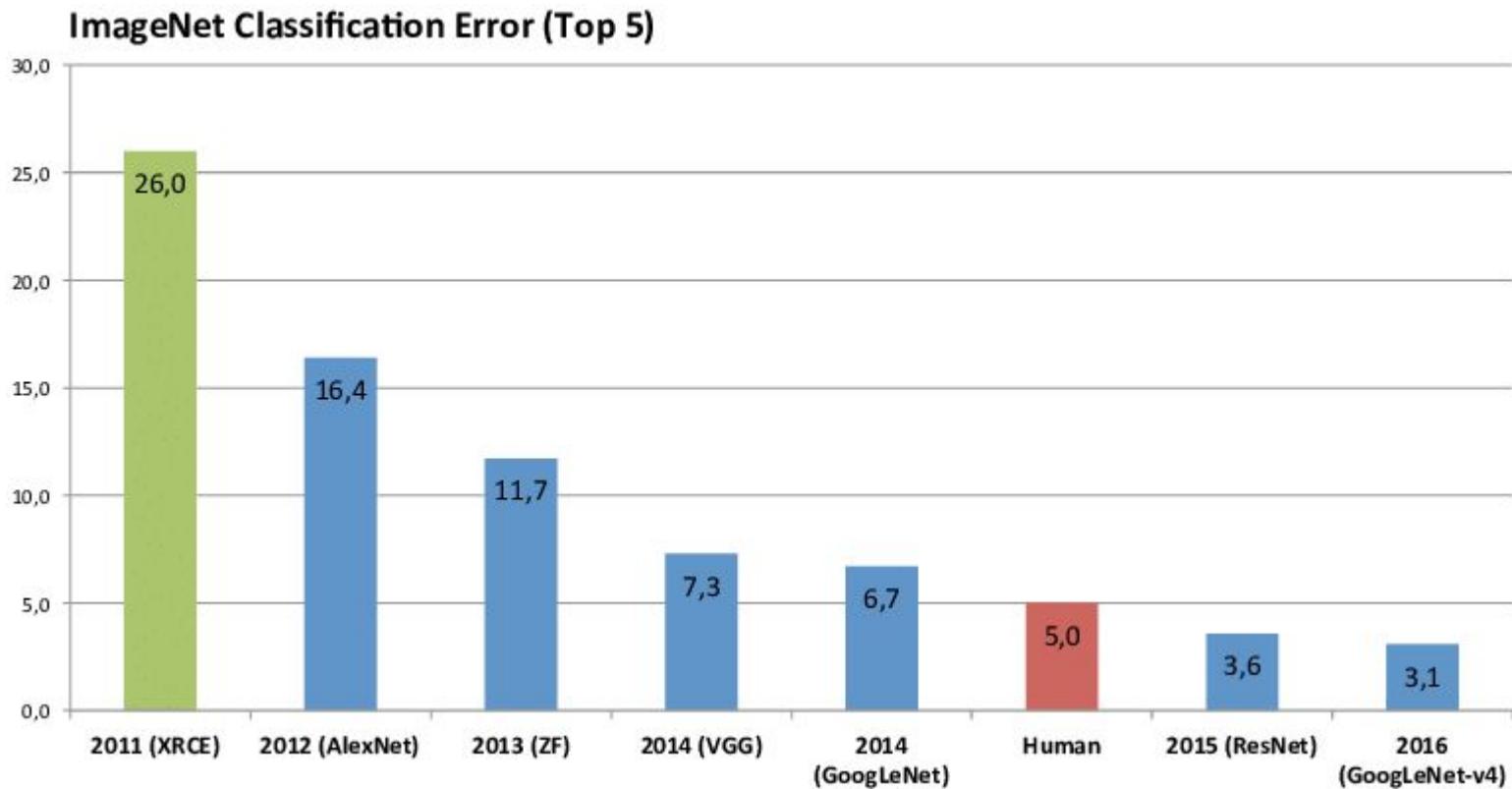
$$b_{x,y}^i = a_{x,y}^i / \left( k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a_{x,y}^j)^2 \right)^\beta$$

ReLU (solid line) effect on training vs tanh (dashed line)

# Alexnet – key features

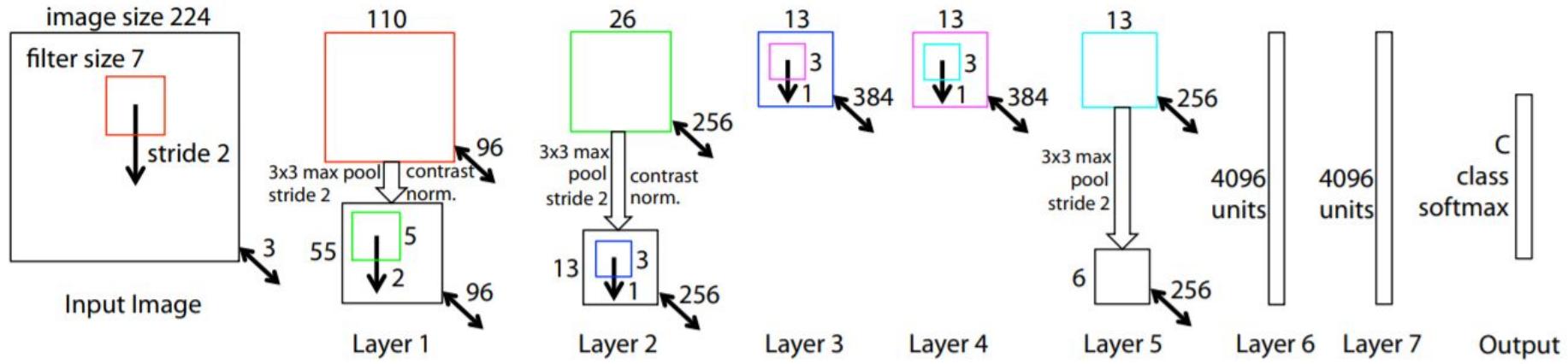
- Training setup
  - Dropout 0.5
  - Data augmentation
  - Batch size 128
  - Gradient descent with momentum (beta = 0.9)
  - Initial learning rate: 1e-2, reduced manually, when a plateau was reached
  - 7 Alexnet ensemble: 18.2% -> 15.4%

# ZFNet, 2013



Built on top of Alexnet, improved hyperparameters

# ZFNet



CONV1: change from (11x11 stride 4) to (7x7 stride 2)

CONV3, 4, 5: instead of 384, 384, 256 filters use 512, 1024, 512

# ZFNet

Error %	Val Top-1	Val Top-5	Test Top-5
(Gunji et al., 2012)	-	-	26.2
(Krizhevsky et al., 2012), 1 convnet	40.7	18.2	--
(Krizhevsky et al., 2012), 5 convnets	38.1	16.4	16.4
(Krizhevsky et al., 2012)*, 1 convnets	39.0	16.6	--
(Krizhevsky et al., 2012)*, 7 convnets	36.7	15.4	15.3
Our replication of			
(Krizhevsky et al., 2012), 1 convnet	40.5	18.1	--
1 convnet as per Fig. 3	38.4	16.5	--
5 convnets as per Fig. 3 – (a)	36.7	15.3	15.3
1 convnet as per Fig. 3 but with layers 3,4,5: 512,1024,512 maps – (b)	37.5	16.0	16.1
6 convnets, (a) & (b) combined	<b>36.0</b>	<b>14.7</b>	<b>14.8</b>

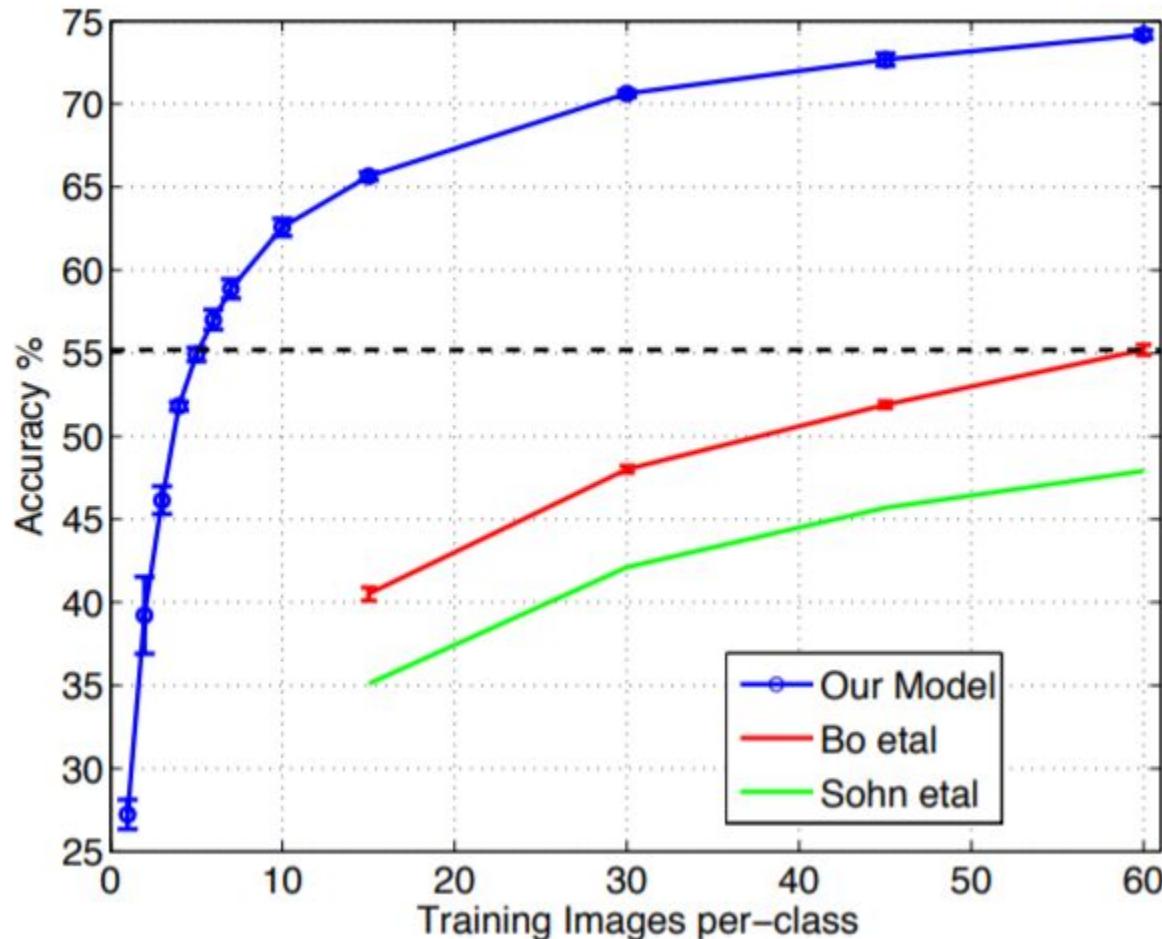
CONV1: change from (11x11 stride 4) to (7x7 stride 2)

CONV3, 4, 5: instead of 384, 384, 256 filters use 512, 1024, 512

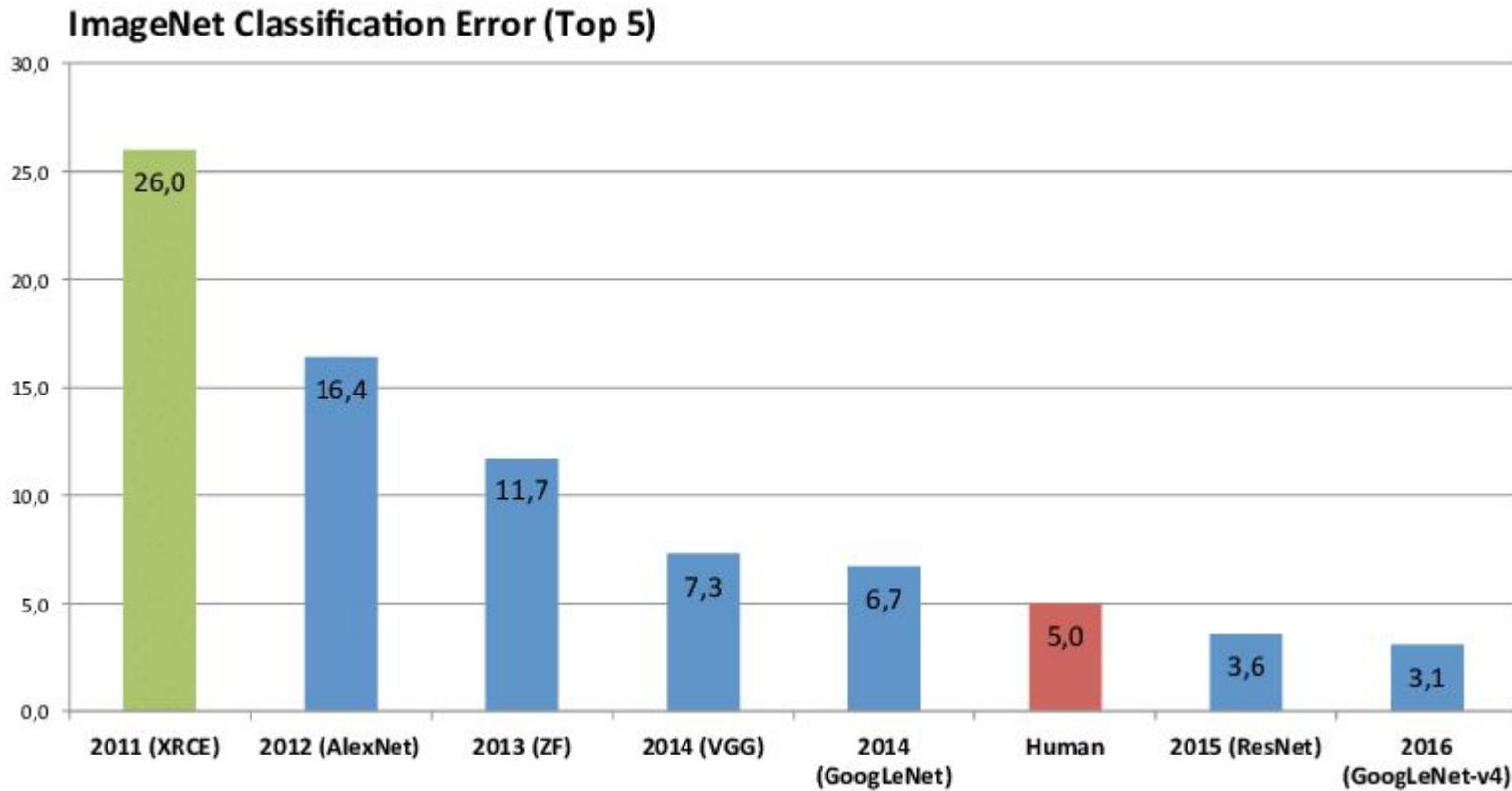
**ImageNet top 5 error: 15.4% -> 14.8%**

# ZFNet – feature generalization

- Caltech 256



# VGG, 2014



# VGG

- Simplifies the network architectures
  - Always use 3x3 filters for convolutions
  - Always use max pooling of filter size 2x2 and a stride of 2

# VGG

- Always use 3x3 filters for convolutions
- Always use max pooling of filter size 2x2 and a stride of 2

**Top 5 accuracy: 11.7  $\rightarrow$  7.3**

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 $\times$ 224 RGB image)					
conv3-64	conv3-64 <b>LRN</b>	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	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 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: Number of parameters (in millions).

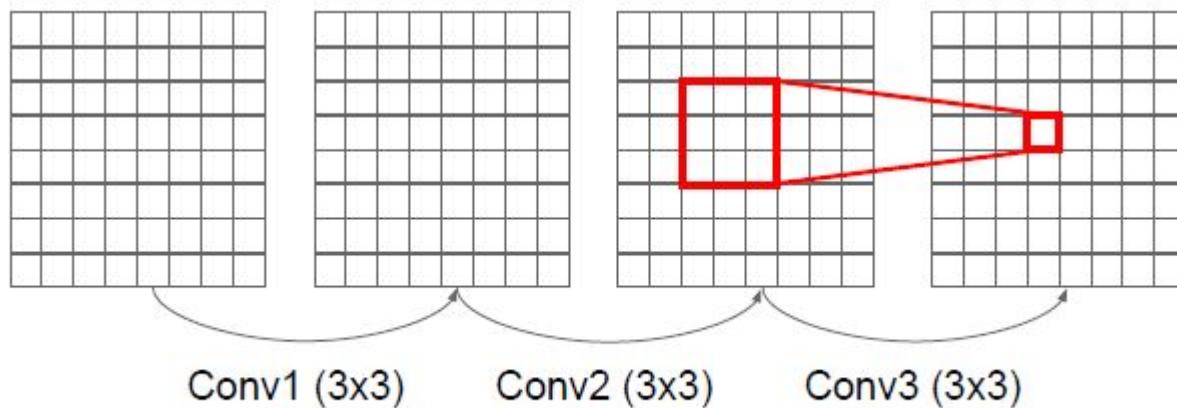
Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

# VGG

Layer (type)	Output Shape	Param #
<hr/>		
conv2d_1 (Conv2D)	(None, 224, 224, 64)	1792
conv2d_2 (Conv2D)	(None, 224, 224, 64)	36928
max_pooling2d_1 (MaxPooling2D)	(None, 112, 112, 64)	0
conv2d_3 (Conv2D)	(None, 112, 112, 128)	73856
conv2d_4 (Conv2D)	(None, 112, 112, 128)	147584
max_pooling2d_2 (MaxPooling2D)	(None, 56, 56, 128)	0
conv2d_5 (Conv2D)	(None, 56, 56, 256)	295168
conv2d_6 (Conv2D)	(None, 56, 56, 256)	590080
conv2d_7 (Conv2D)	(None, 56, 56, 256)	590080
max_pooling2d_3 (MaxPooling2D)	(None, 28, 28, 256)	0
conv2d_8 (Conv2D)	(None, 28, 28, 512)	1180160
conv2d_9 (Conv2D)	(None, 28, 28, 512)	2359808
conv2d_10 (Conv2D)	(None, 28, 28, 512)	2359808
max_pooling2d_4 (MaxPooling2D)	(None, 14, 14, 512)	0
conv2d_11 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_12 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_13 (Conv2D)	(None, 14, 14, 512)	2359808
max_pooling2d_5 (MaxPooling2D)	(None, 7, 7, 512)	0
flatten_1 (Flatten)	(None, 25088)	0
dense_1 (Dense)	(None, 4096)	102764544
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 4096)	16781312
dropout_2 (Dropout)	(None, 4096)	0
dense_3 (Dense)	(None, 2)	8194
<hr/>		
Total params: 134,268,738		
Trainable params: 134,268,738		
Non-trainable params: 0		

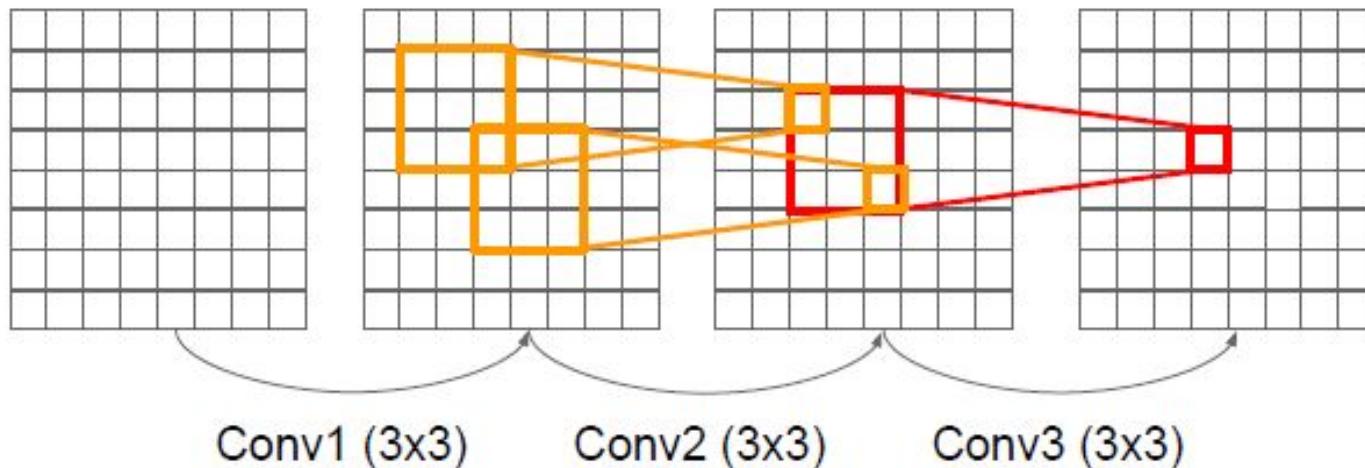
# VGG

- “*It is easy to see that a stack of two  $3 \times 3$  conv. layers has an effective receptive field of  $5 \times 5$ ; three such layers have a  $7 \times 7$  effective receptive field.*”



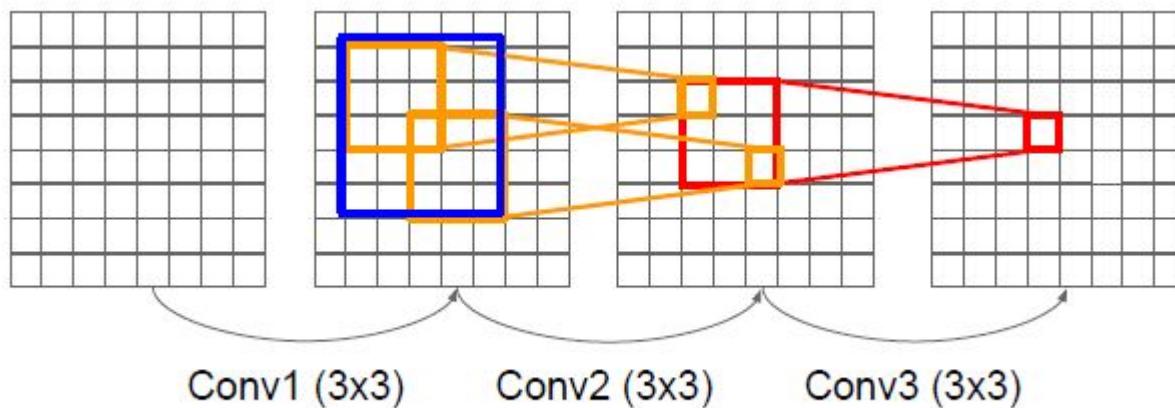
# VGG

- “*It is easy to see that a stack of two  $3 \times 3$  conv. layers has an effective receptive field of  $5 \times 5$ ; three such layers have a  $7 \times 7$  effective receptive field.*”



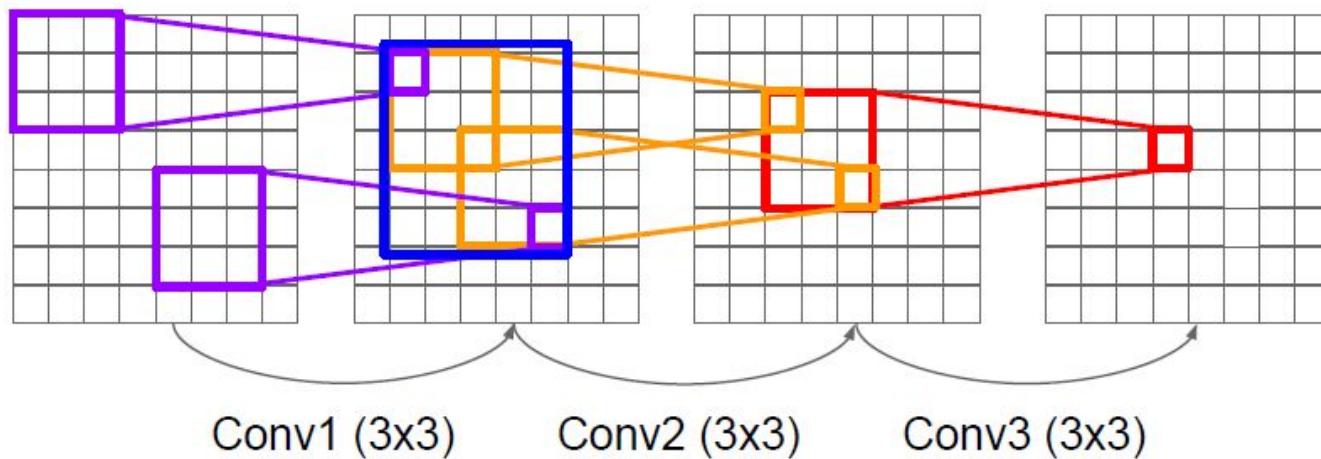
# VGG

- “*It is easy to see that a stack of two  $3 \times 3$  conv. layers has an effective receptive field of  $5 \times 5$ ; three such layers have a  $7 \times 7$  effective receptive field.*”



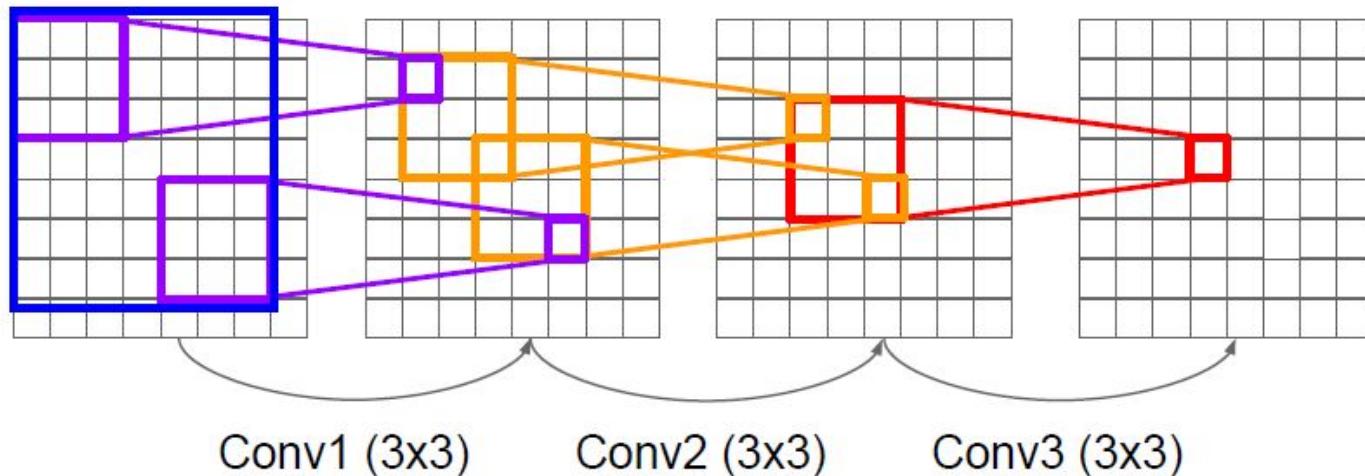
# VGG

- “*It is easy to see that a stack of two  $3 \times 3$  conv. layers has an effective receptive field of  $5 \times 5$ ; three such layers have a  $7 \times 7$  effective receptive field.*”



# VGG

- “*It is easy to see that a stack of two  $3 \times 3$  conv. layers has an effective receptive field of  $5 \times 5$ ; three such layers have a  $7 \times 7$  effective receptive field.*”



# VGG

- “*It is easy to see that a stack of two  $3 \times 3$  conv. layers has an effective receptive field of  $5 \times 5$ ; three such layers have a  $7 \times 7$  effective receptive field.*”
- What is the effect of this replacement in terms on the number of parameters?
- What about non-linearities?

# VGG

- “*It is easy to see that a stack of two  $3 \times 3$  conv. layers has an effective receptive field of  $5 \times 5$ ; three such layers have a  $7 \times 7$  effective receptive field.*”
- What is the effect of this replacement in terms on the number of parameters?

$$3 * (3^2 C) = 27C \text{ vs } 7^2 C = 49C$$

- What about non-linearities?

# VGG – key features

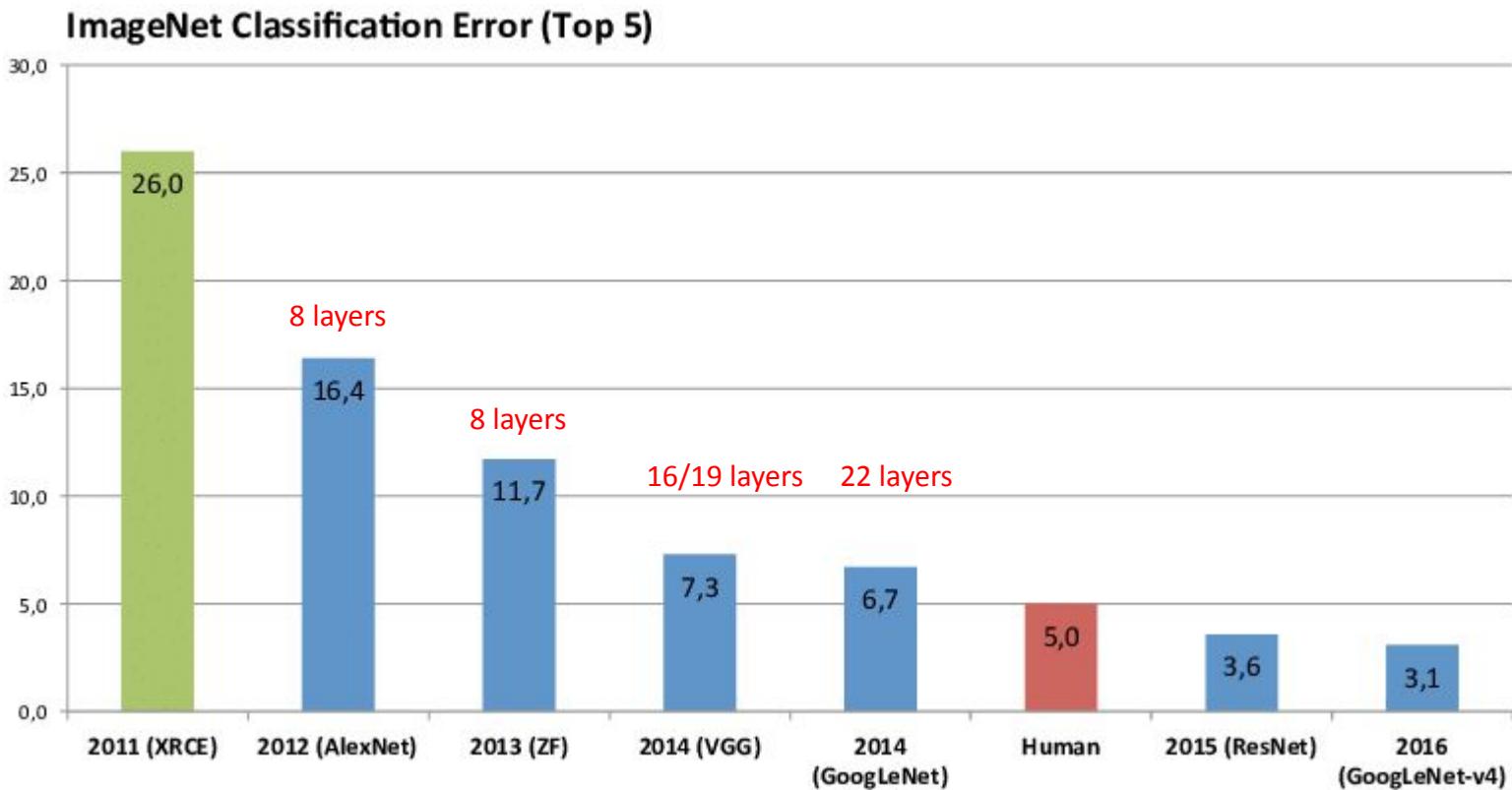
- Did not use local response normalization layers
- Use ensembles to boost performance
- Use VGG 16 or VGG 19 (VGG 19 brings only a small increase in performance, but it requires more memory)
- Similar training procedure as in AlexNet



**WE NEED TO GO**

**DEEPER**

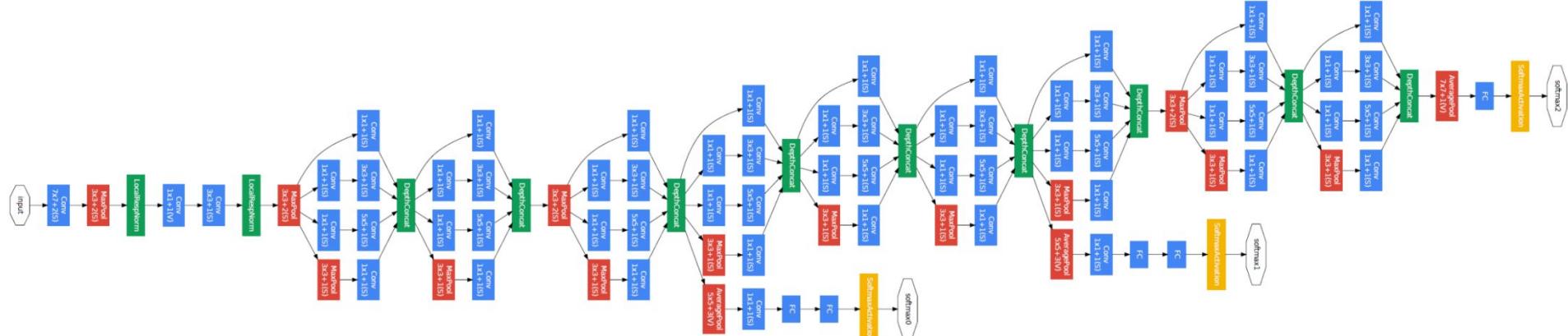
# GoogLeNet, 2014



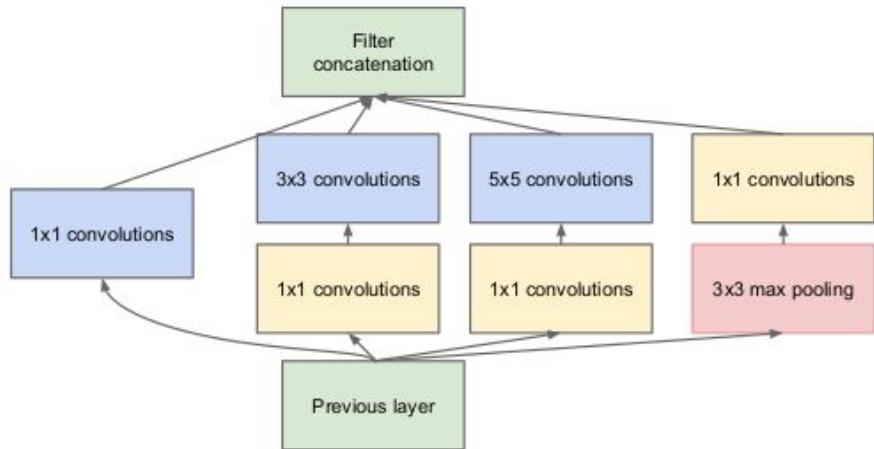
6,7 top-5 accuracy

# GoogLeNet

*This name is an homage to Yann LeCun's pioneering LeNet 5 network*

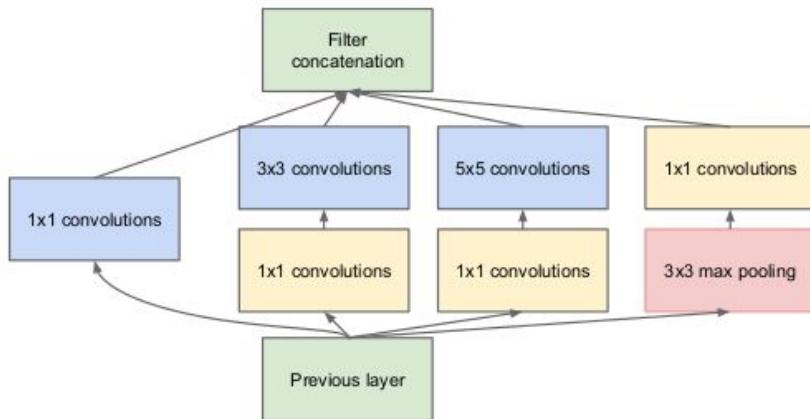


## Inception module

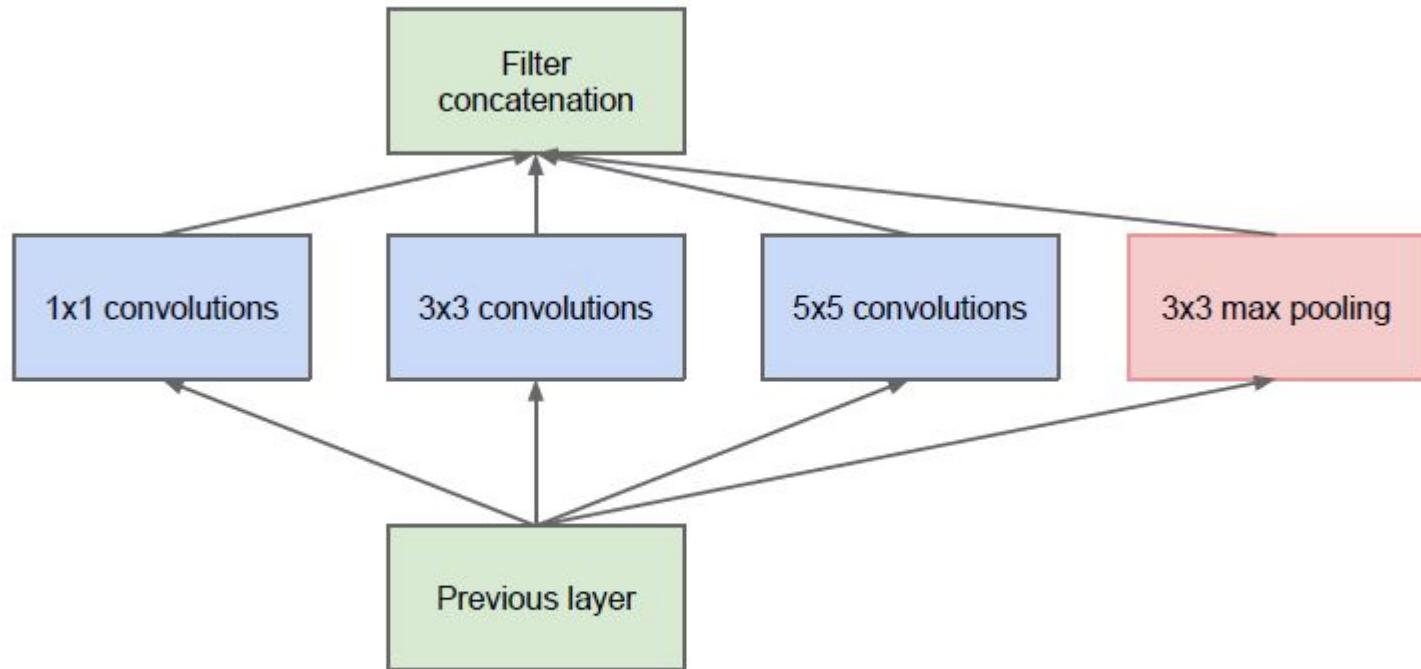


# Inception module

- You don't need to "pick" the filter sizes, instead you let the network "choose" between several values
  - Filters of different sizes
  - Pooling layer
  - Concatenate activations depth-wise



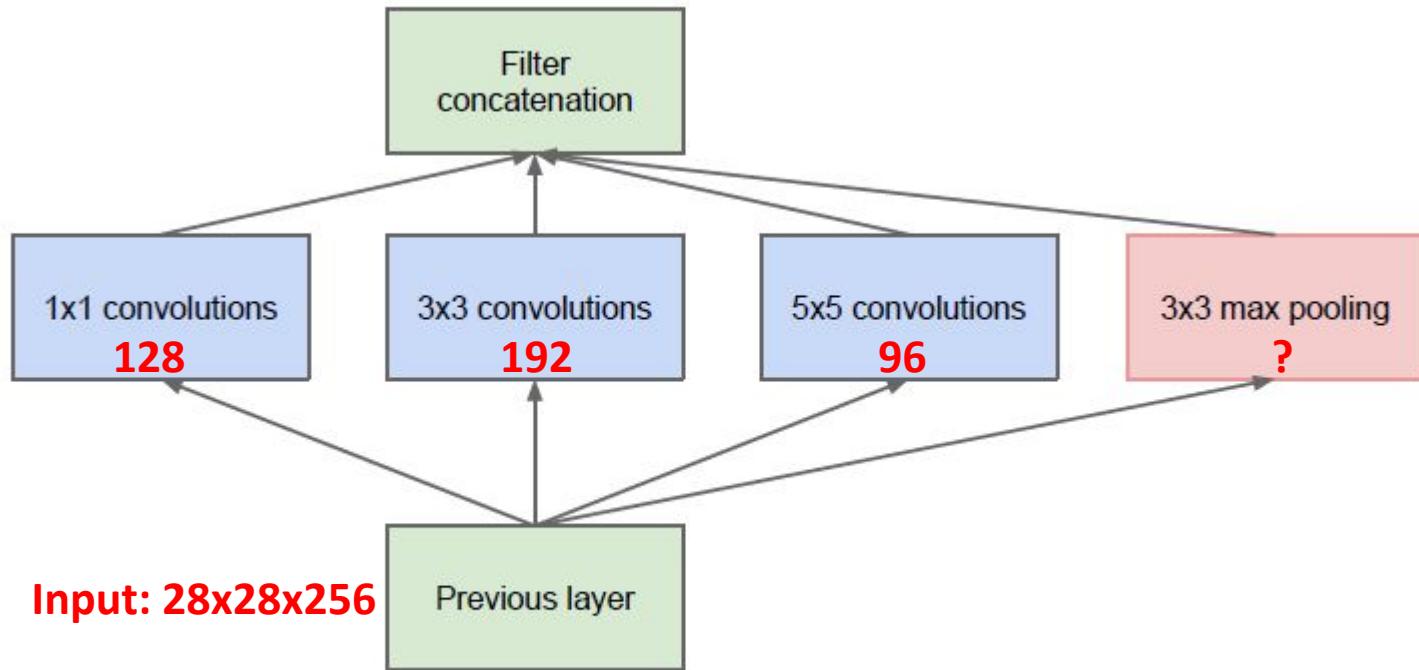
# Inception module



(a) Inception module, naïve version

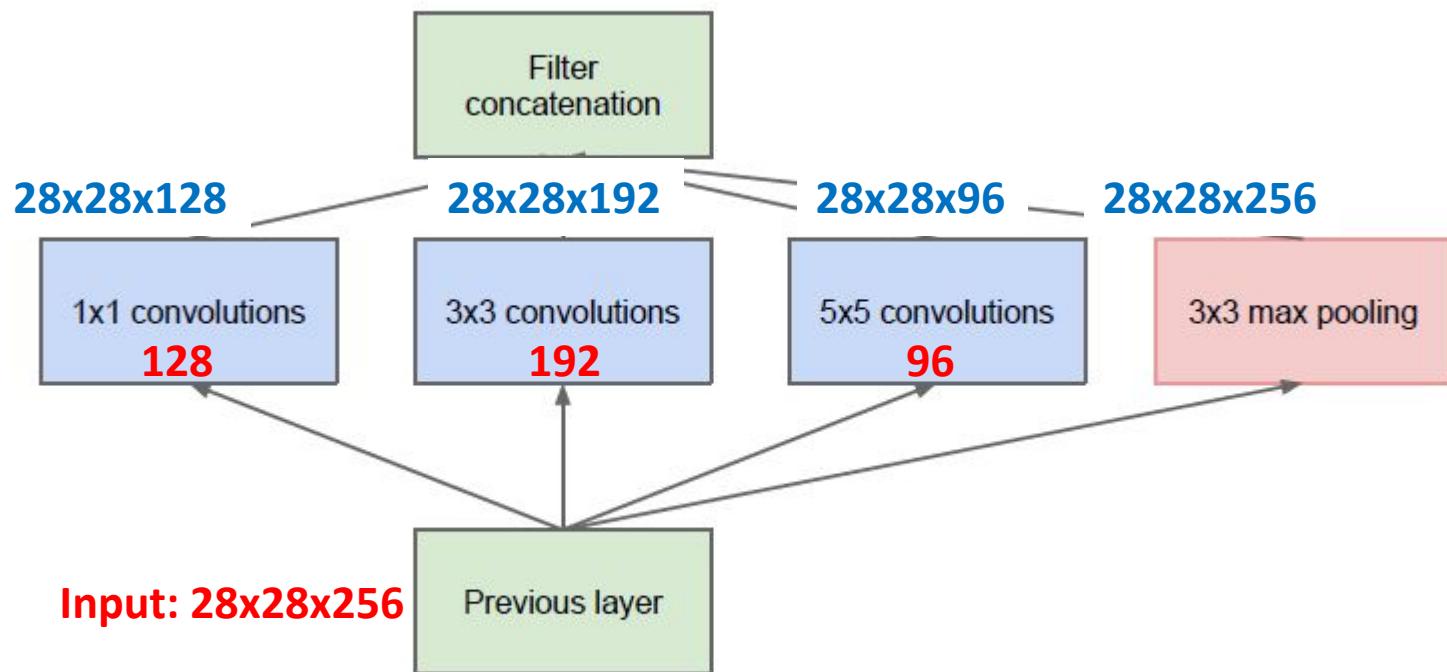
# Inception module

What are the output sizes of all these operations?



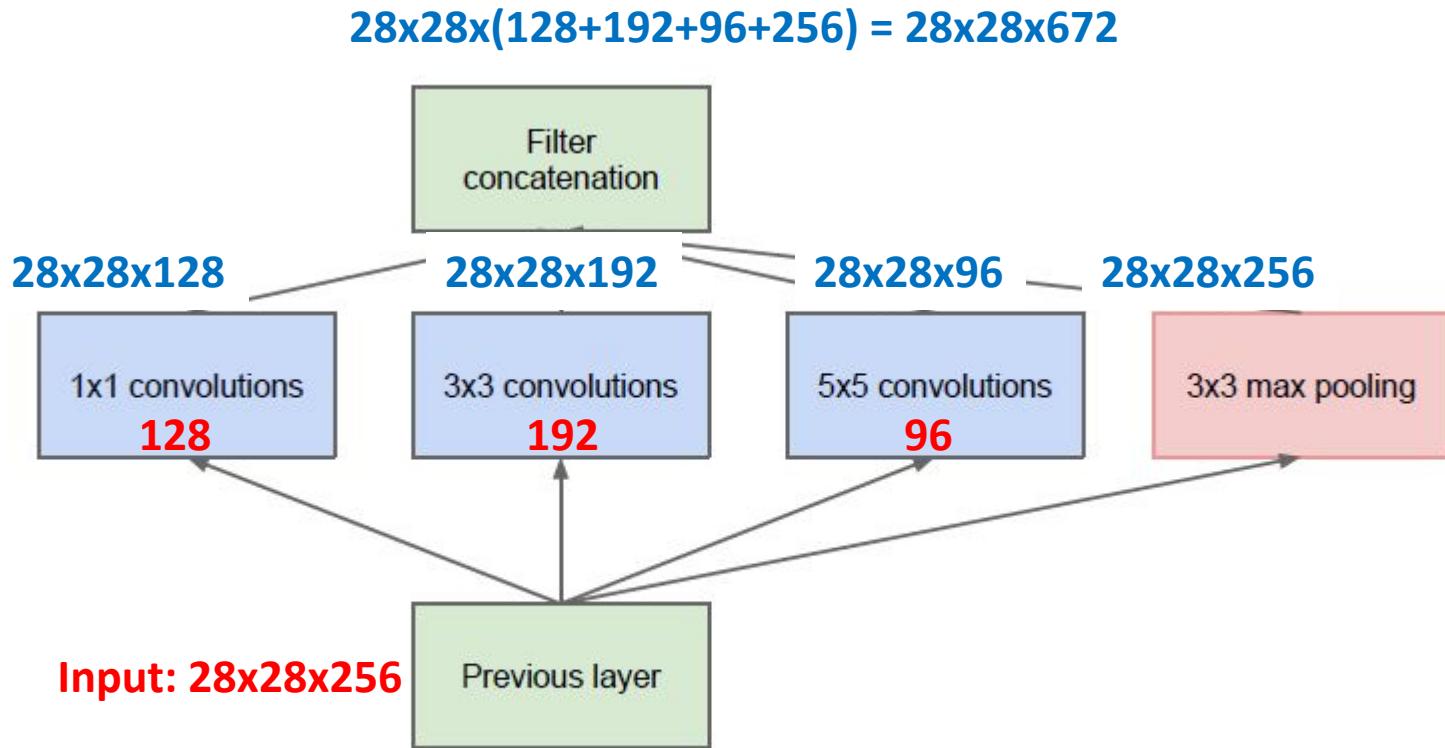
(a) Inception module, naïve version

# Inception module



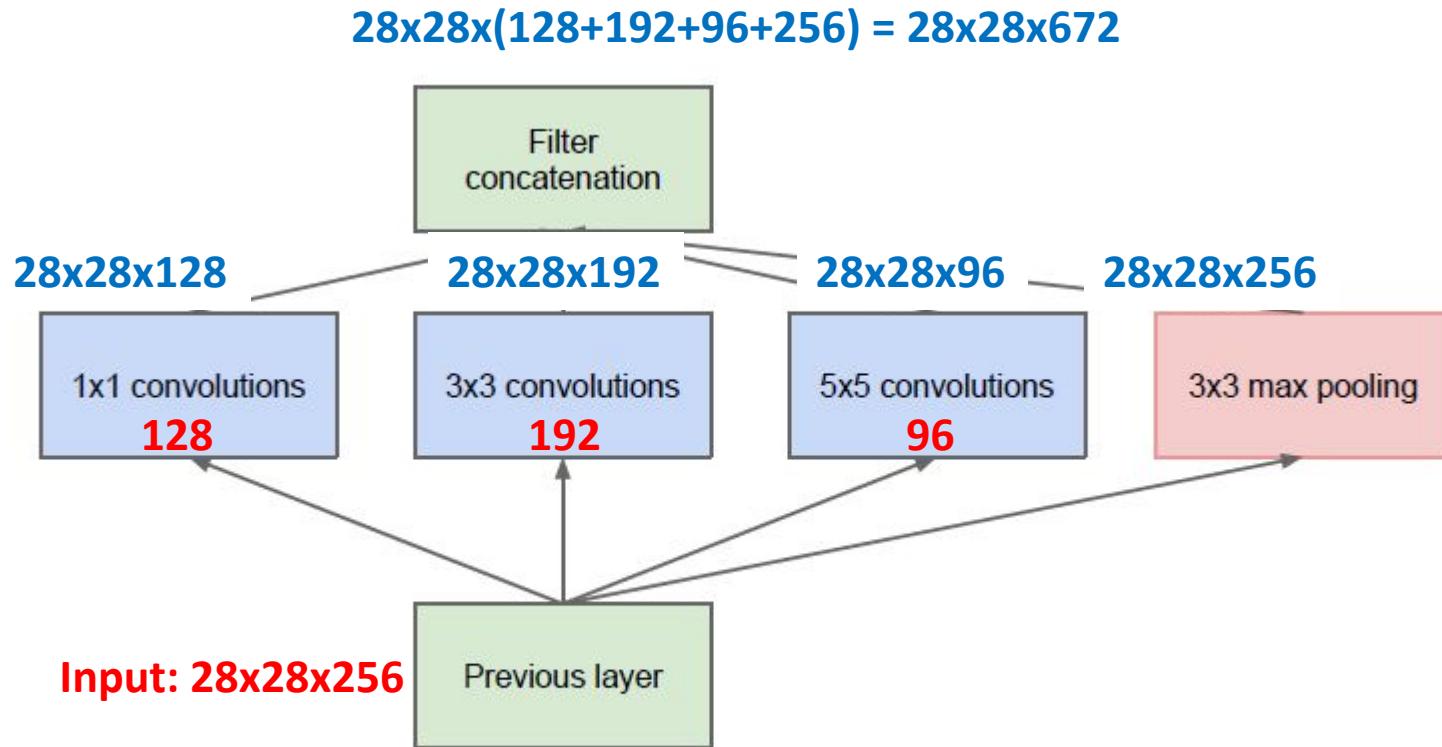
(a) Inception module, naïve version

# Inception module



(a) Inception module, naïve version

# Inception module

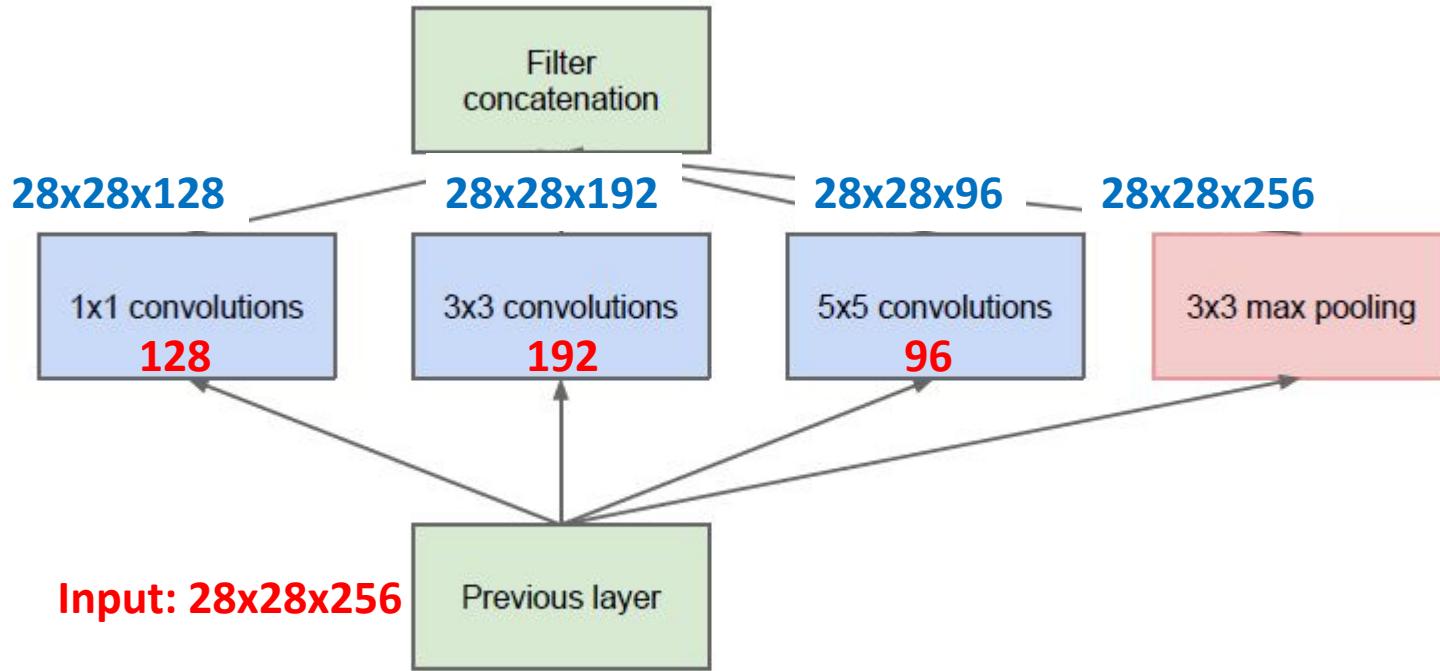


(a) Inception module, naïve version

[1x1 conv, 128 filters]: ? convolution operations  
[3x3 conv, 192 filters]: ? convolution operations  
[5x5 conv, 96 filters]: ? convolution operations

# Inception module

$$28 \times 28 \times (128 + 192 + 96 + 256) = 28 \times 28 \times 672 = 526848$$



(a) Inception module, naïve version

[1x1 conv, 128 filters]:  $28 \times 28 \times 128 \times 1 \times 1 \times 256$

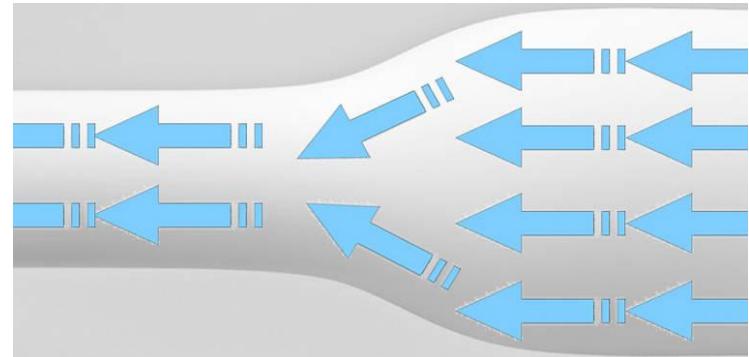
[3x3 conv, 192 filters]:  $28 \times 28 \times 192 \times 3 \times 3 \times 256$

[5x5 conv, 96 filters]:  $28 \times 28 \times 96 \times 5 \times 5 \times 256$

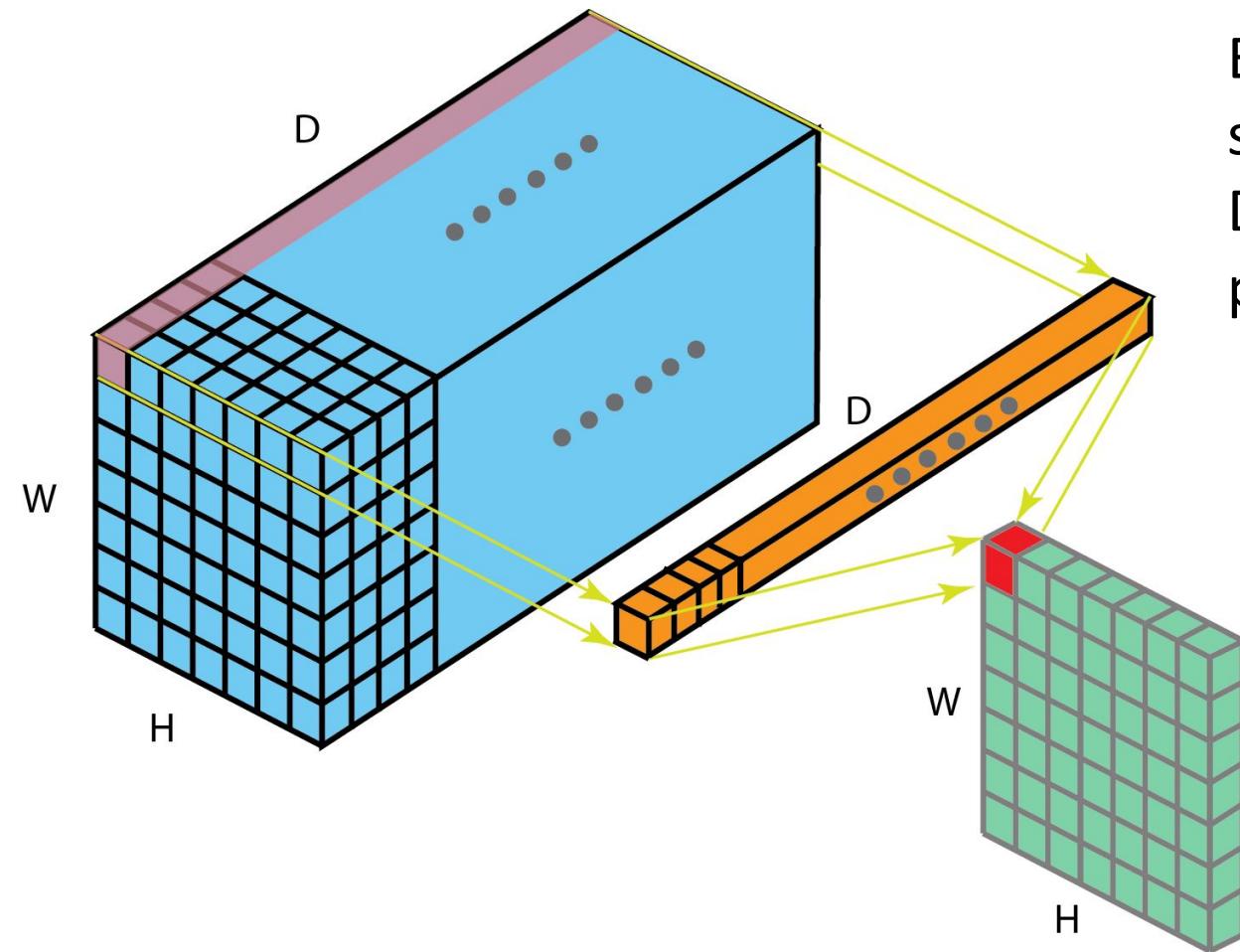
—————> **>850M**

# Bottleneck

- Use *bottleneck* layers to reduce the depth dimensions of the activations
  - use 1x1 convolutions

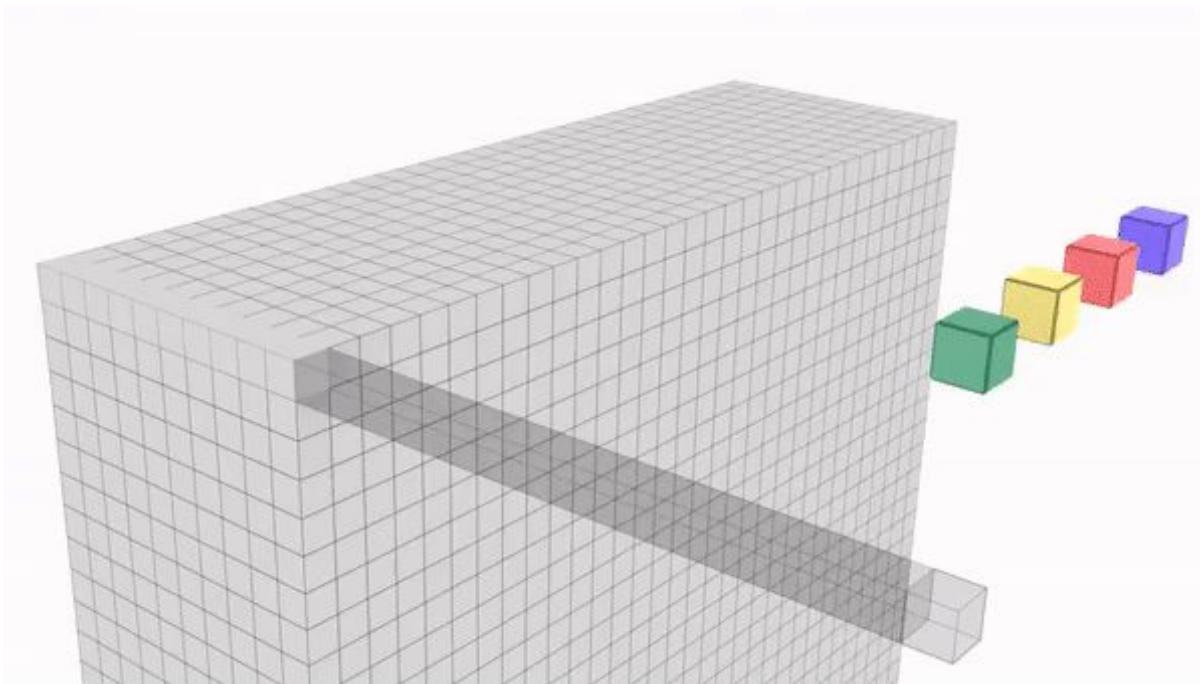


# $1 \times 1$ convolutions



Each filter has a  $1 \times 1 \times D$  size and performs a  $D$ -dimensional dot product

# 1x1 convolutions



# 1x1 convolution



**Yann LeCun**

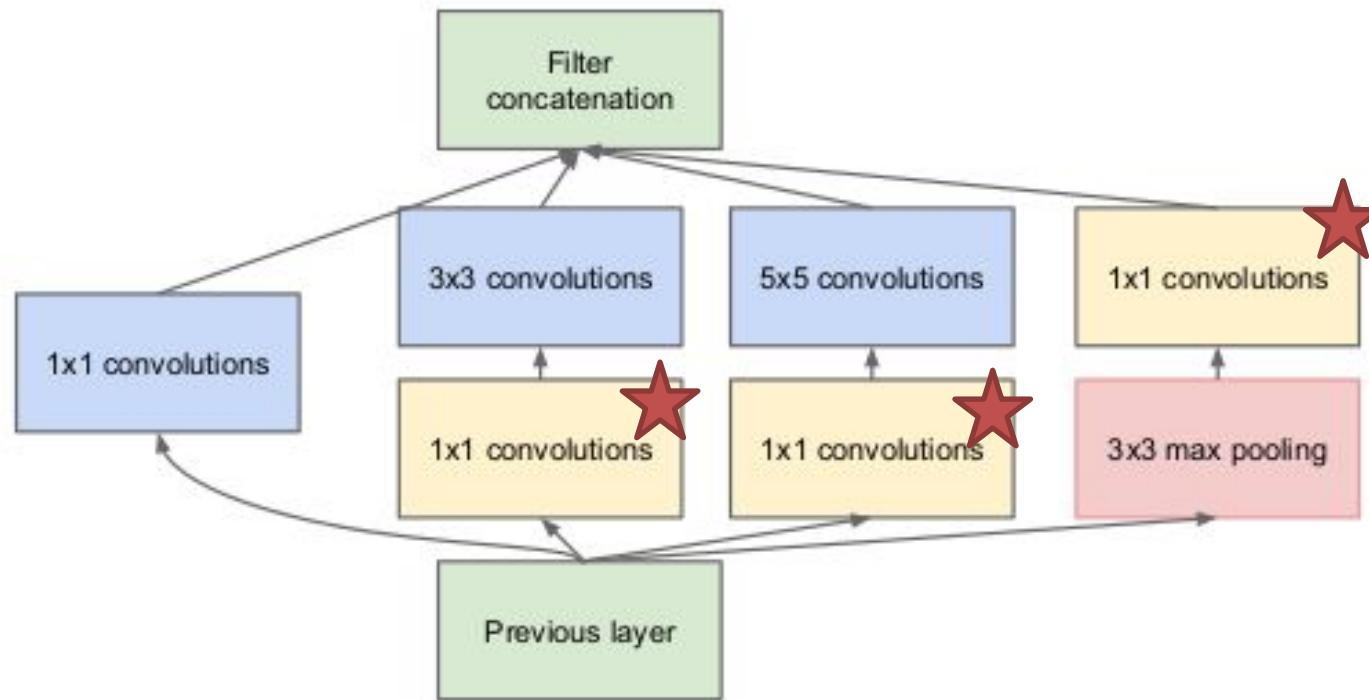
April 6, 2015 ·

In Convolutional Nets, there is no such thing as "fully-connected layers". There are only convolution layers with 1x1 convolution kernels and a full connection table.

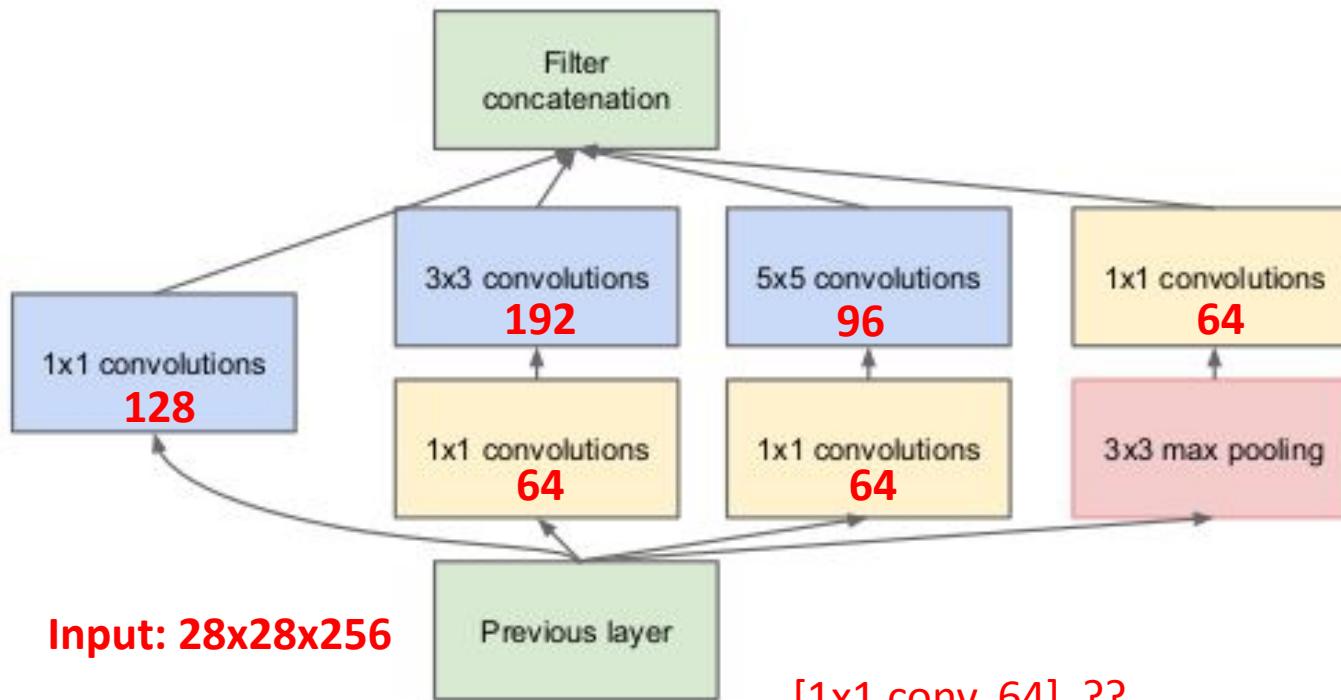
It's a too-rarely-understood fact that ConvNets don't need to have a fixed-size input. You can train them on inputs that happen to produce a single output vector (with no spatial extent), and then apply them to larger images. Instead of a single output vector, you then get a spatial map of output vectors. Each vector sees input windows at different locations on the input.

In that scenario, the "fully connected layers" really act as 1x1 convolutions.

# The real Inception module

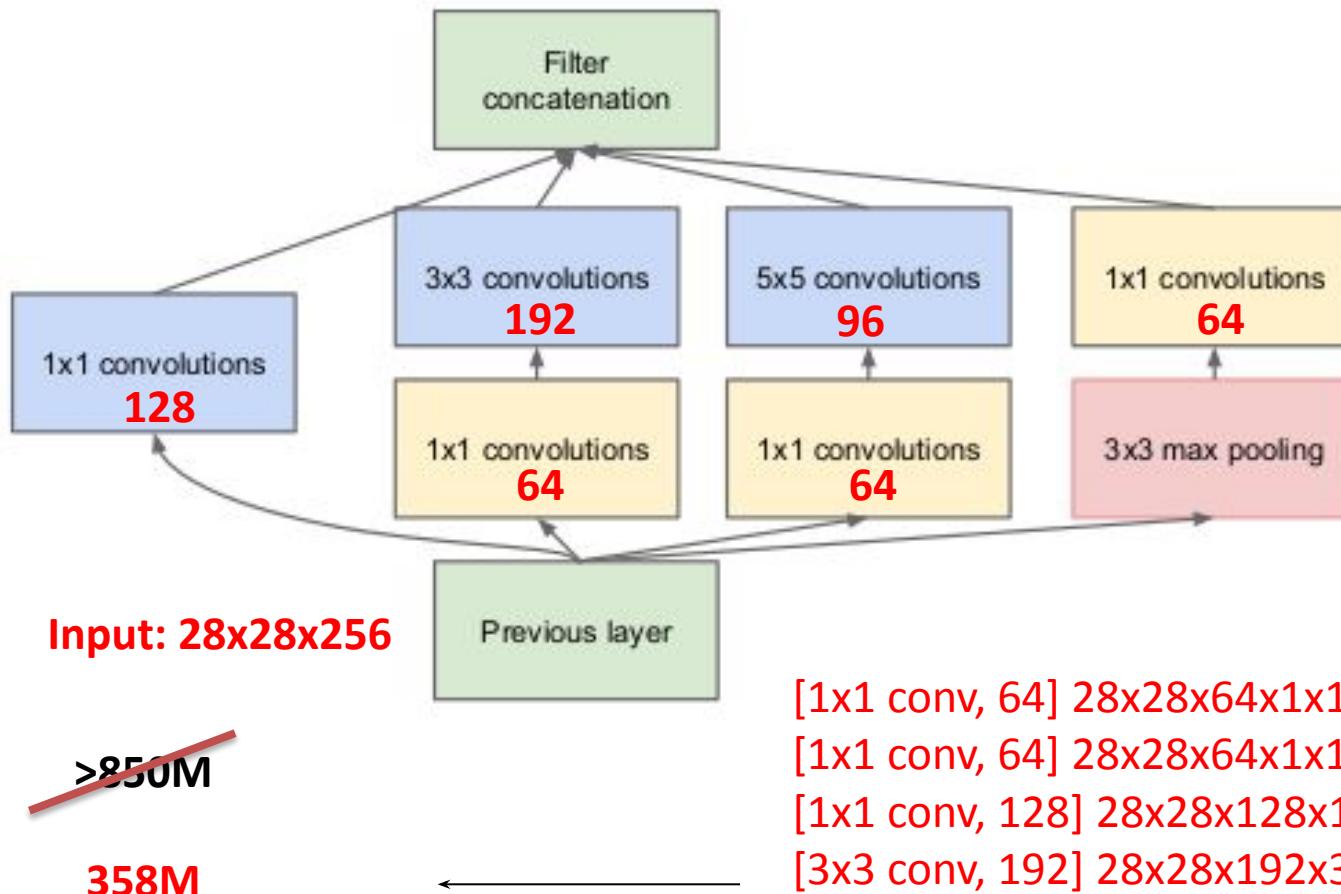


# The real Inception module



[1x1 conv, 64] ??  
[1x1 conv, 64] ??  
[1x1 conv, 128] ??  
[3x3 conv, 192] ??  
[5x5 conv, 96] ??  
[1x1 conv, 64] ??

# The real Inception module

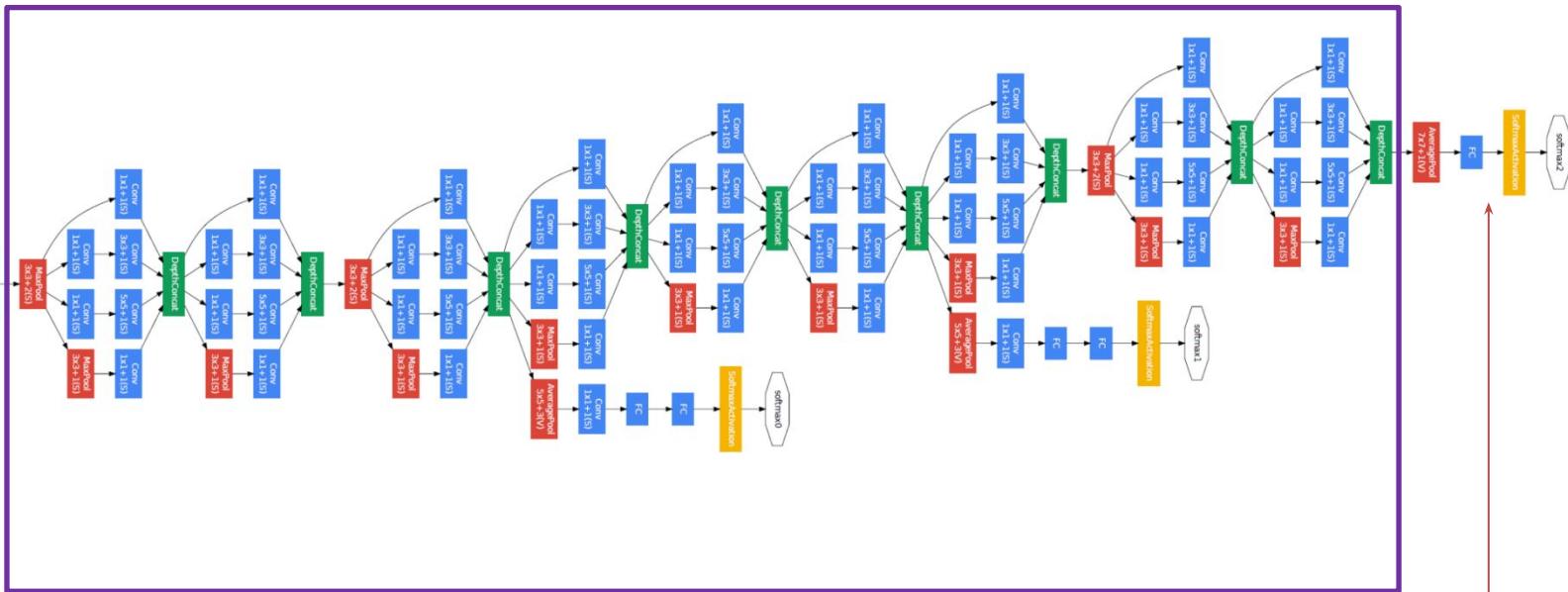


[ $1 \times 1$  conv, 64]  $28 \times 28 \times 64 \times 1 \times 1 \times 256$   
[ $1 \times 1$  conv, 64]  $28 \times 28 \times 64 \times 1 \times 1 \times 256$   
[ $1 \times 1$  conv, 128]  $28 \times 28 \times 128 \times 1 \times 1 \times 256$   
[ $3 \times 3$  conv, 192]  $28 \times 28 \times 192 \times 3 \times 3 \times 64$   
[ $5 \times 5$  conv, 96]  $28 \times 28 \times 96 \times 5 \times 5 \times 64$   
[ $1 \times 1$  conv, 64]  $28 \times 28 \times 64 \times 1 \times 1 \times 256$

# GoogLe Net

Stack Inception modules on top of each other

# GoogLe Net



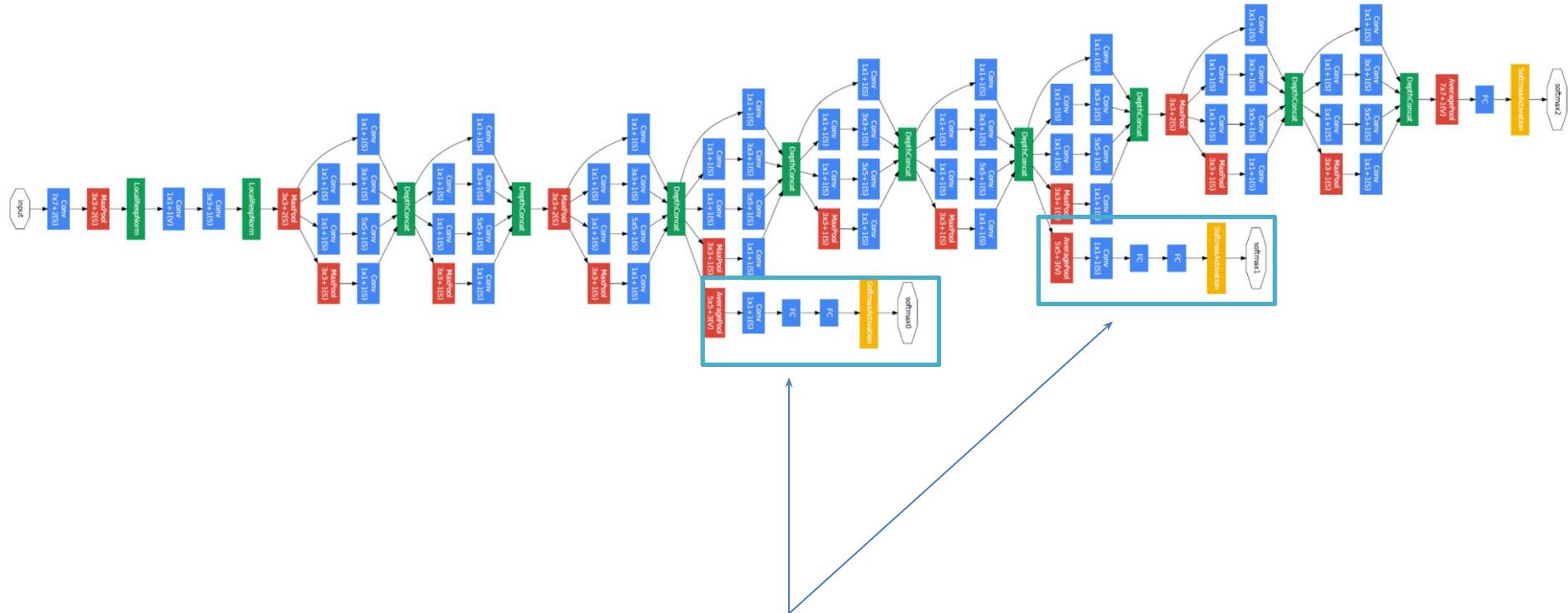
Inception modules

Stem network:

Conv – Pool – 2x Conv-Pool

Classifier output  
removes FC layers

# GoogLe Net



**Output layers to inject additional gradient at lower layers**  
**AvgPool – 1x1 Conv – FC – FC - Softmax**

# GoogLe Net

Stack Inception modules on top of each other

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								



# GoogLe Net

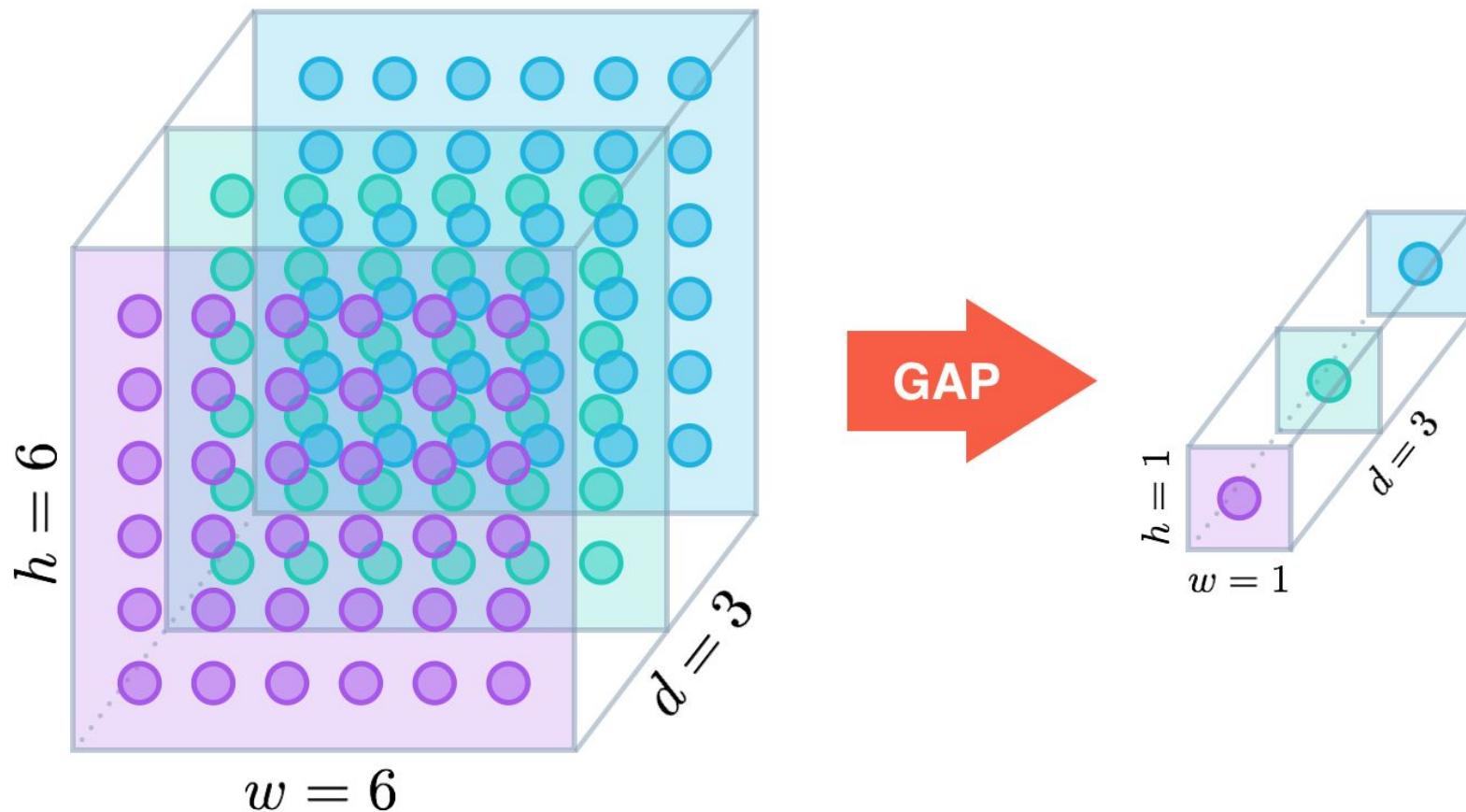
- Removes fully connected layers
- *It was found that a move from fully connected layers to average pooling improved the top-1 accuracy by about 0.6%.*

# Global average pooling

Network In Network, 2014

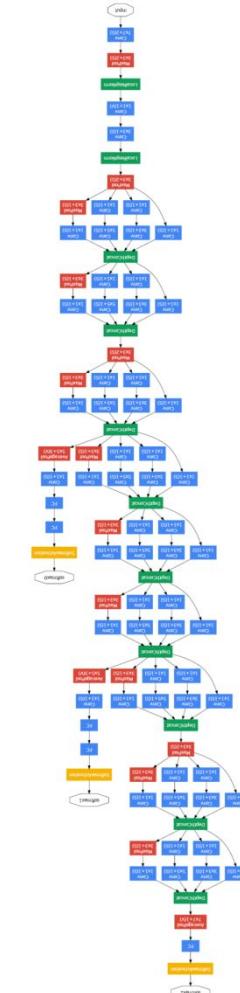
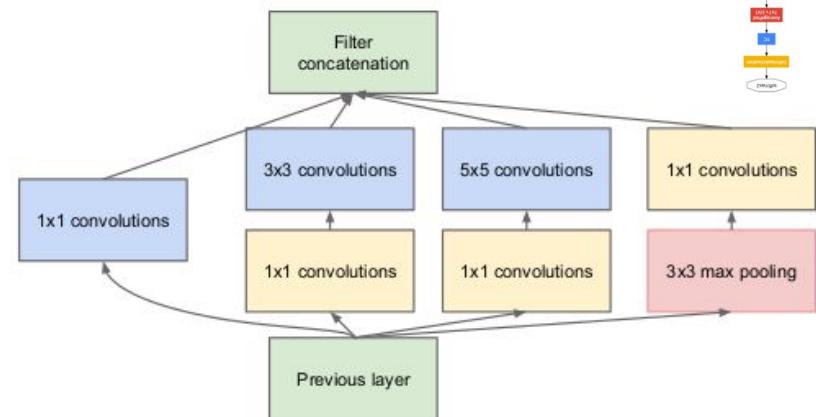
- Instead of adding fully connected layers on top of the feature maps, take **the average** of each feature map, and the resulting vector is fed directly into the softmax layer.
  - is more native to the convolution structure by enforcing correspondences between feature maps and categories (feature maps can be easily interpreted as categories confidence maps)
  - no parameter to optimize in the global average pooling thus overfitting is avoided at this layer
  - sums up spatial information -> more robust to spatial translations of the input

# Global average pooling (GAP)

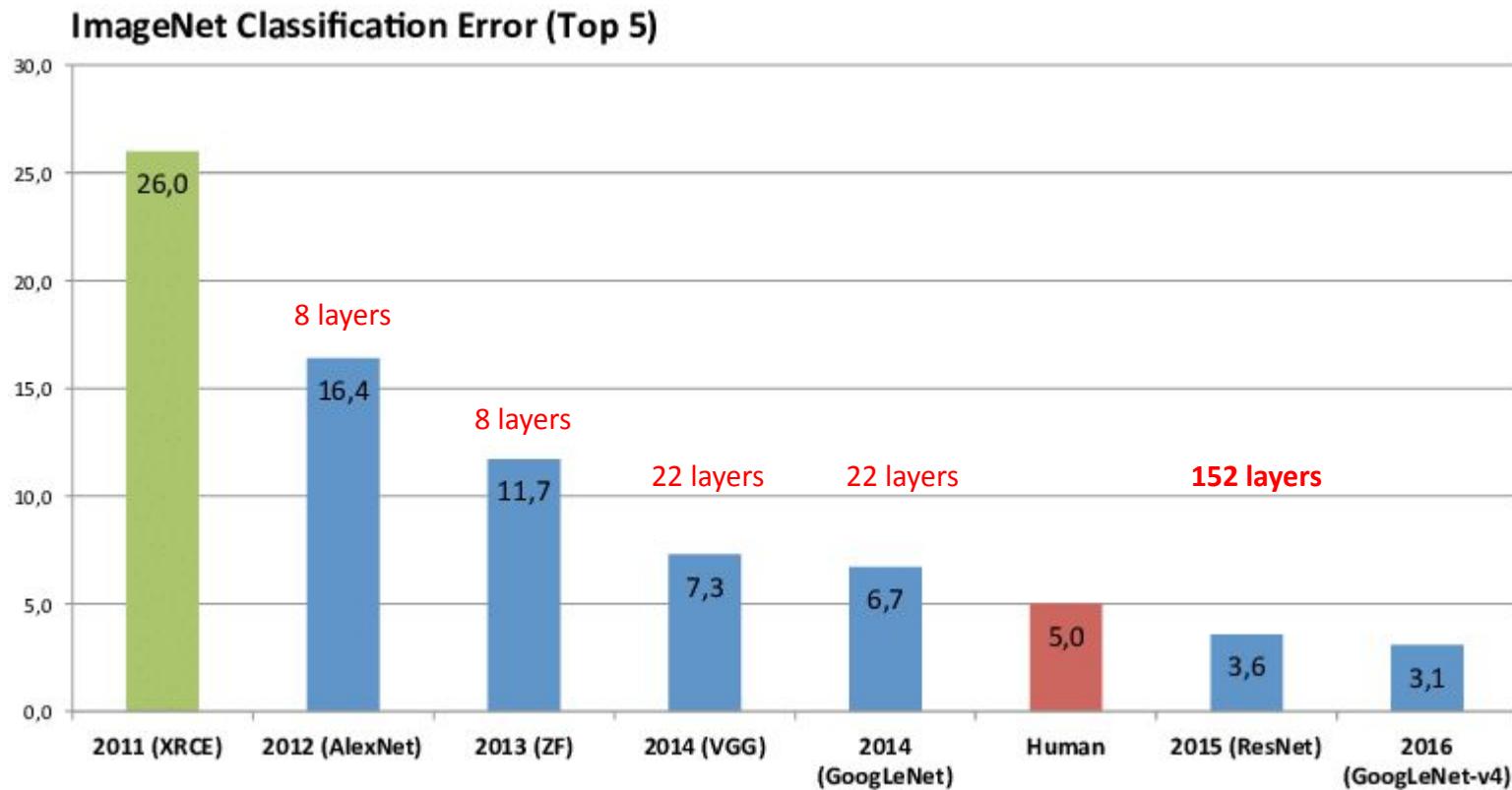


# GoogLe Net

- 22 layers
  - Inception module
  - 12x less parameters than AlexNet
  - **Top 5 accuracy: 6.7%**
  - Removes fully connected layers as in NIN
    - implementation differs in that we use an extra linear layer



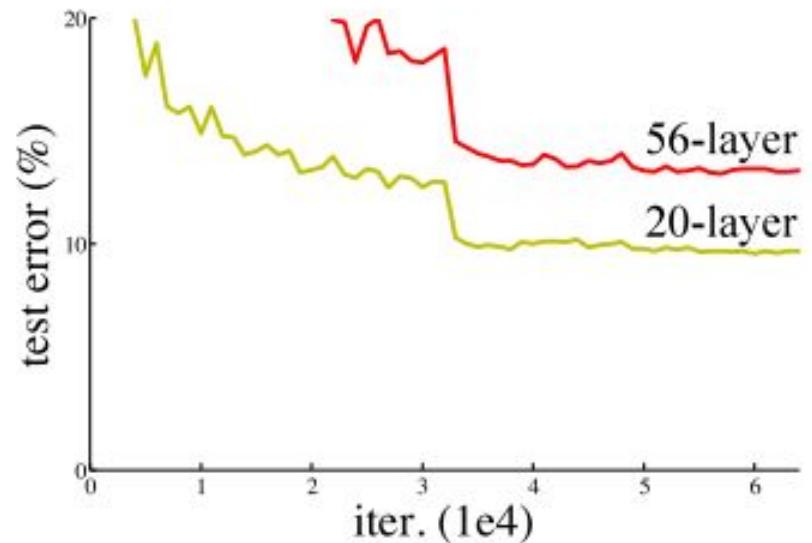
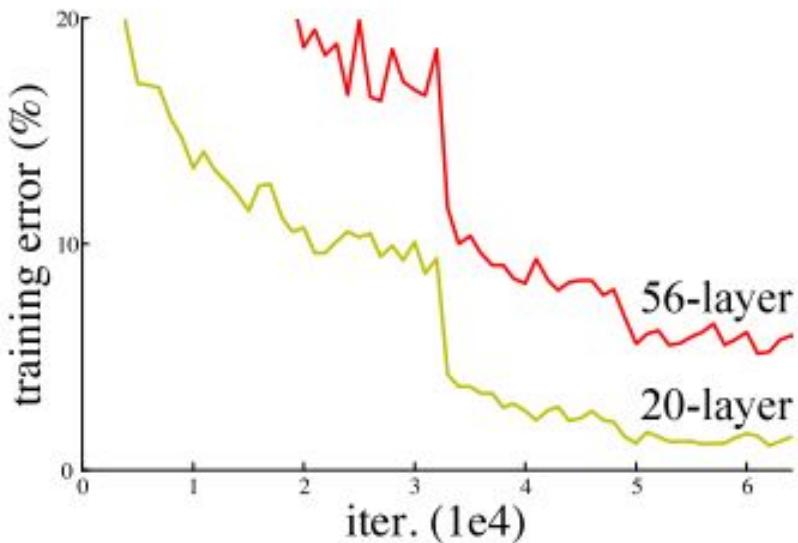
# ResNet, 2015



6.7 top-5 accuracy

# Res Net

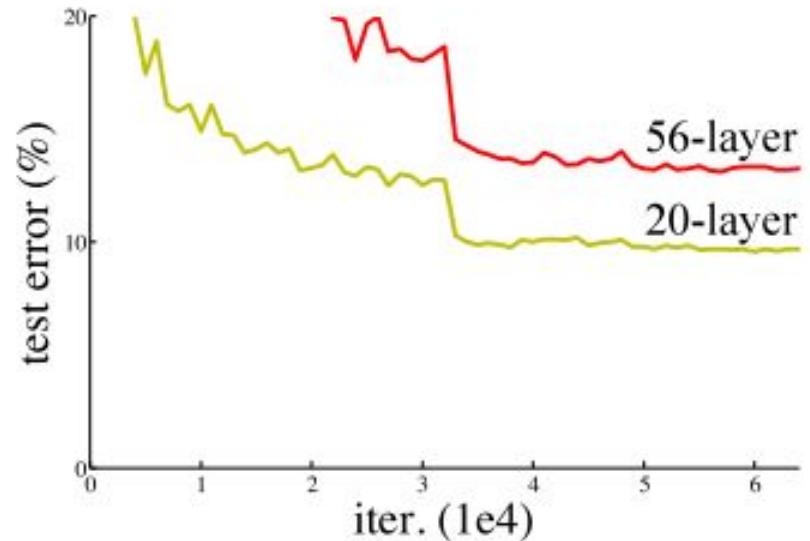
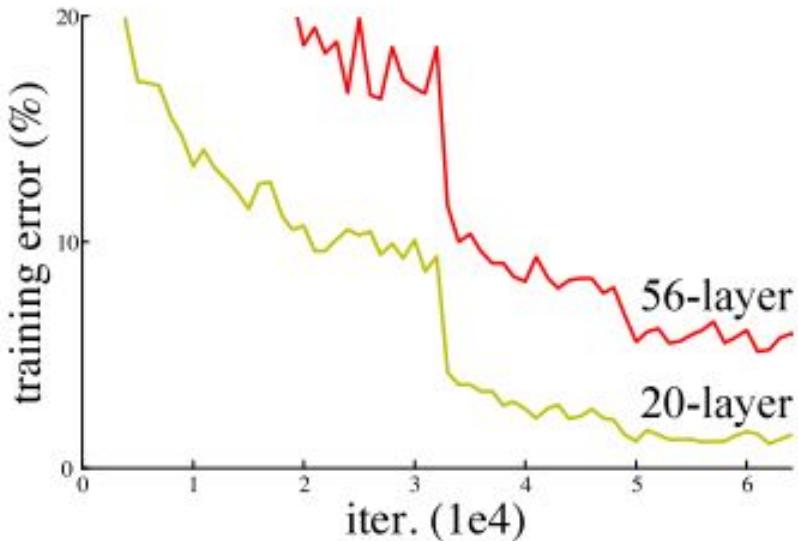
Increasing a network depth



Overfitting?

# Res Nets

Increasing a network depth



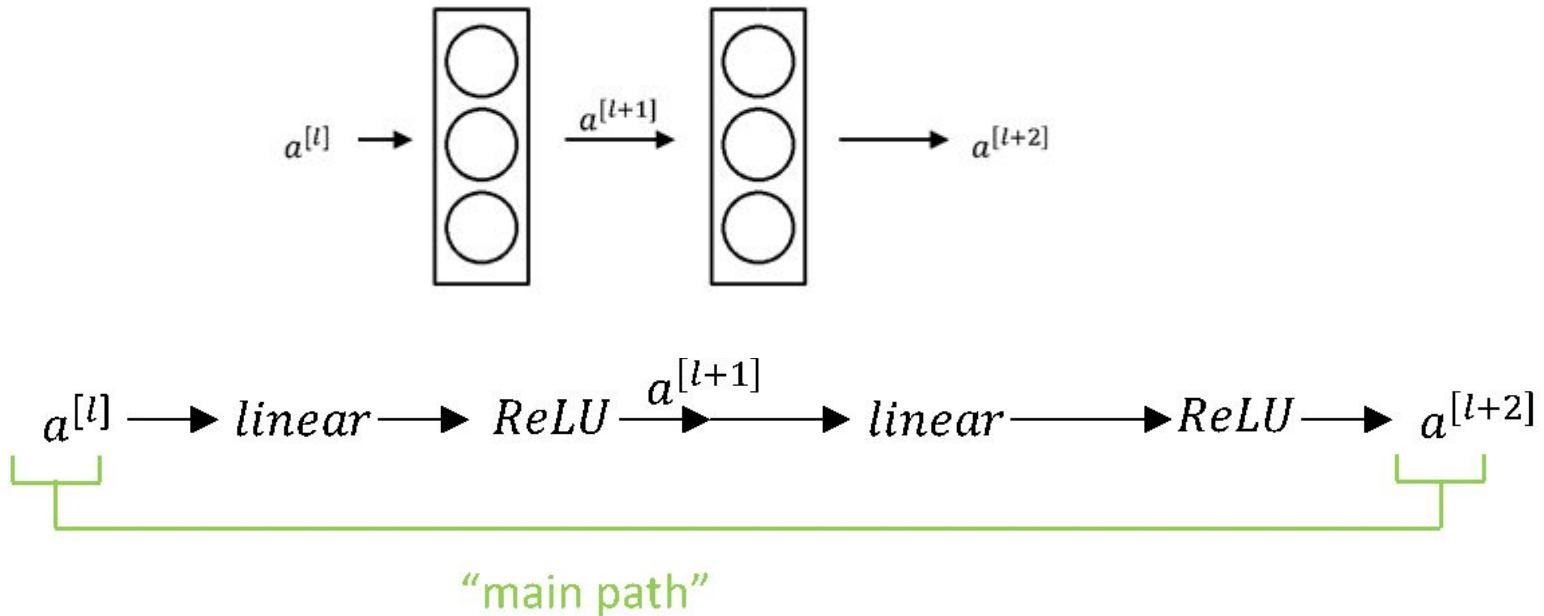
Overfitting? NO!

**Hypothesis: Deeper models are harder to optimize**

# Res Nets

- Deeper models shouldn't hurt performance, they should be at least as good as shallower ones
- Copy the learned layers from the shallower layer and set additional layers to identity mapping

# Classical neural network block



$$a^{[l]} \rightarrow linear \rightarrow ReLU \rightarrow a^{[l+1]} \rightarrow linear \rightarrow ReLU \rightarrow a^{[l+2]}$$

$$z^{[l+1]} = W^{[l+1]}a^{[l]} + b^{[l+1]}$$

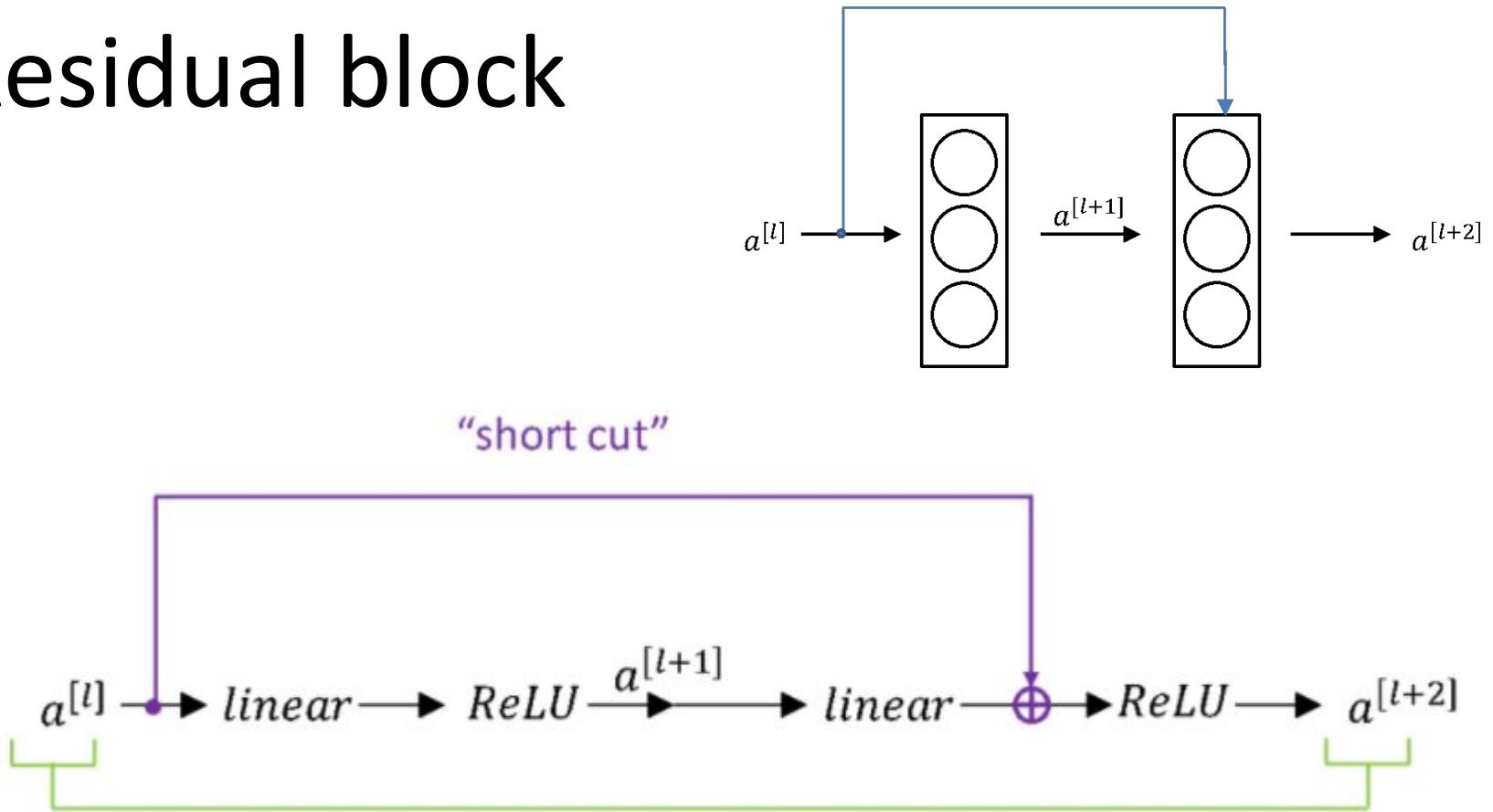
$$a^{[l+1]} = g(z^{[l+1]})$$

$$z^{[l+2]} = W^{[l+2]}a^{[l+1]} + b^{[l+2]}$$

$$a^{[l+2]} = g(z^{[l+2]})$$

Image source: <http://datahacker.rs/deep-learning-residual-networks/>  
Example credit: Andrew Ng

# Residual block



$$z^{[l+1]} = W^{[l+1]}a^{[l]} + b^{[l+1]}$$

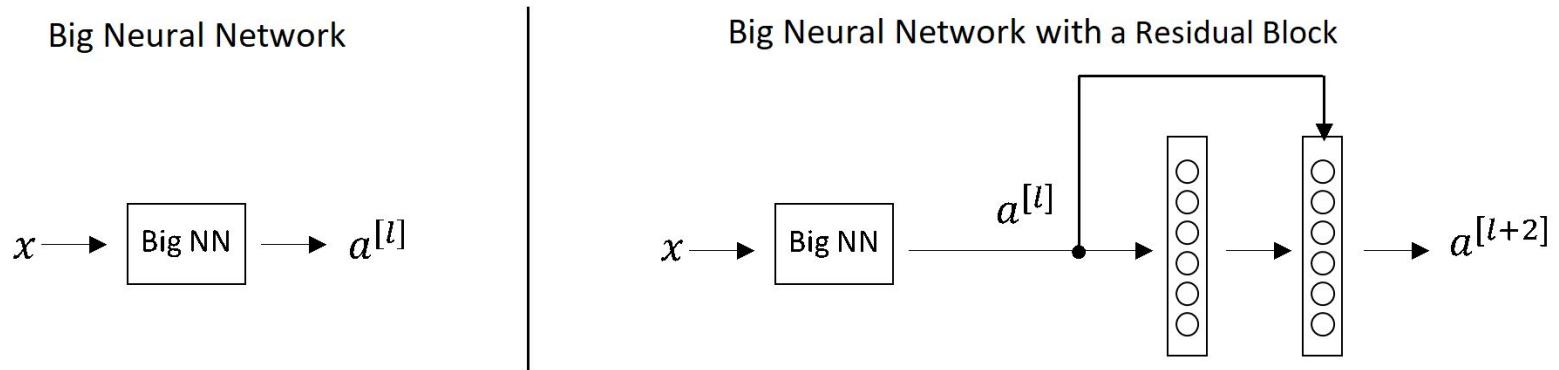
$$a^{[l+1]} = g(z^{[l+1]})$$

$$z^{[l+2]} = W^{[l+2]}a^{[l+1]} + b^{[l+2]}$$

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

Image source: <http://datahacker.rs/deep-learning-residual-networks/>

# Residual block



Equations for the neural network with a residual block are:

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

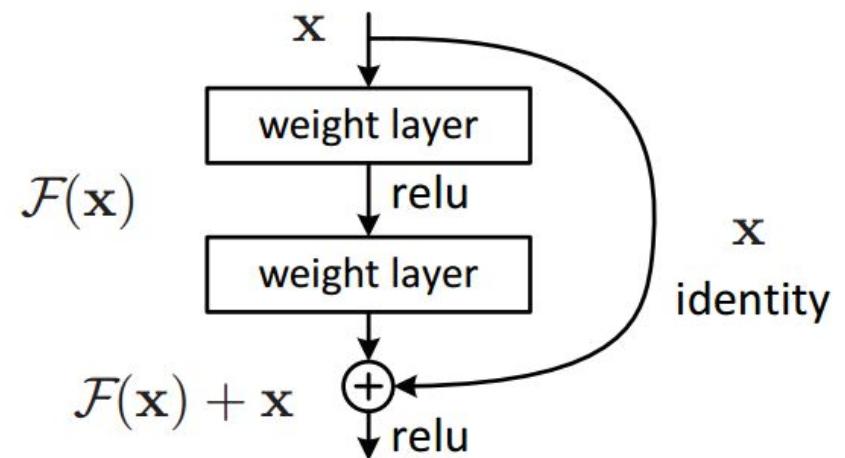
$$a^{[l+2]} = g(W^{[l+2]}a^{[l+1]} + b^{[l+2]} + a^{[l]})$$

If we have  $W^{[l+2]} = \mathbf{0}$  and  $b = \mathbf{0}$  then:

$$a^{[l+2]} = g(W^{[l+2]}a^{[l+1]} + b^{[l+2]} + a^{[l]}) = g(a^{[l]}) = a^{[l]}$$

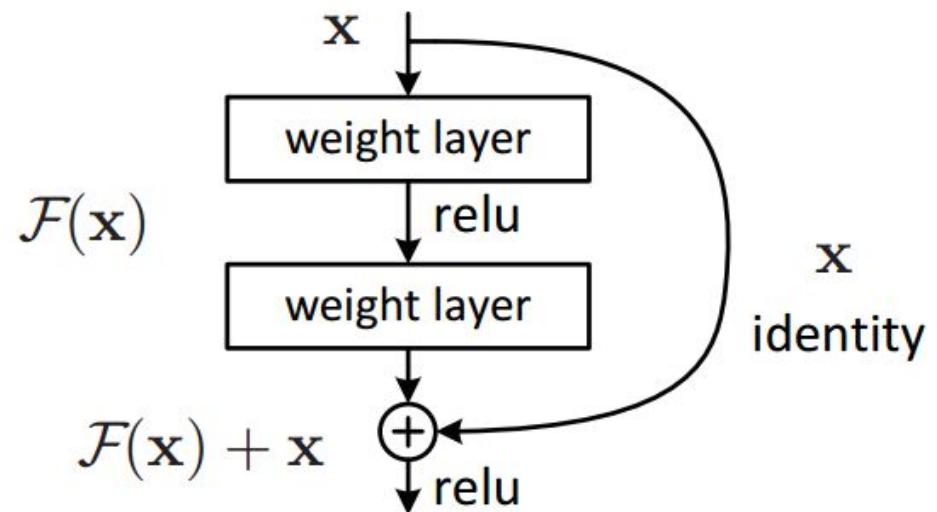
# Res Net

- Try to fit a residual mapping instead of directly trying to fit a desired underlying mapping



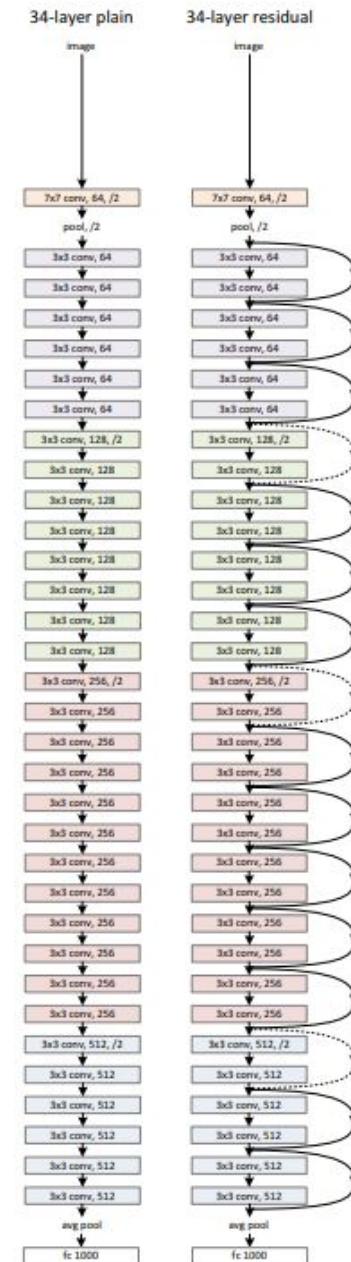
# Residual blocks

- Residual blocks
  - Main path
  - Shortcut



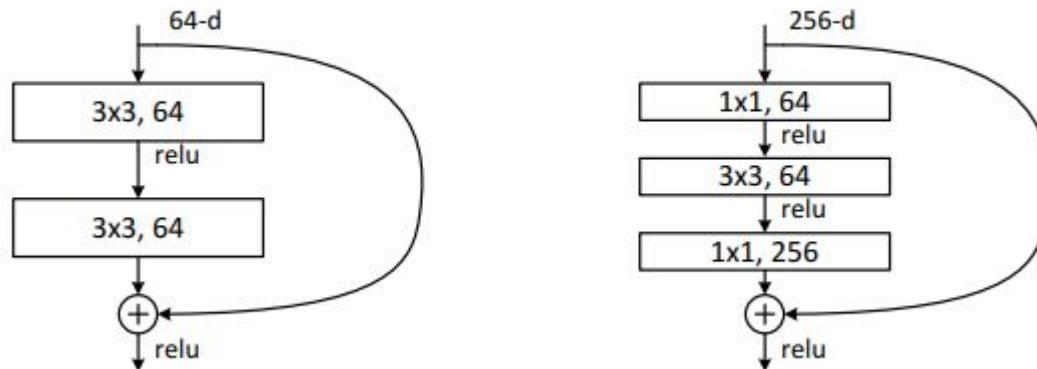
# Res Net

- Start with a CONV layer
- Stack multiple residual blocks on top of each other
  - Every residual block has two 3x3 conv layers
  - Periodically, double number of filters and downsample spatially using stride 2
- No fully connected layers



# Res Net > 50 layers

- Use “bottlenecks” just like in Inception networks



A deeper residual function  $F$  for ImageNet.  
Left: a building block (on  $56 \times 56$  feature maps)  
as in Fig. 3 for ResNet34. Right: a “bottleneck”  
building block for ResNet-50/101/152.

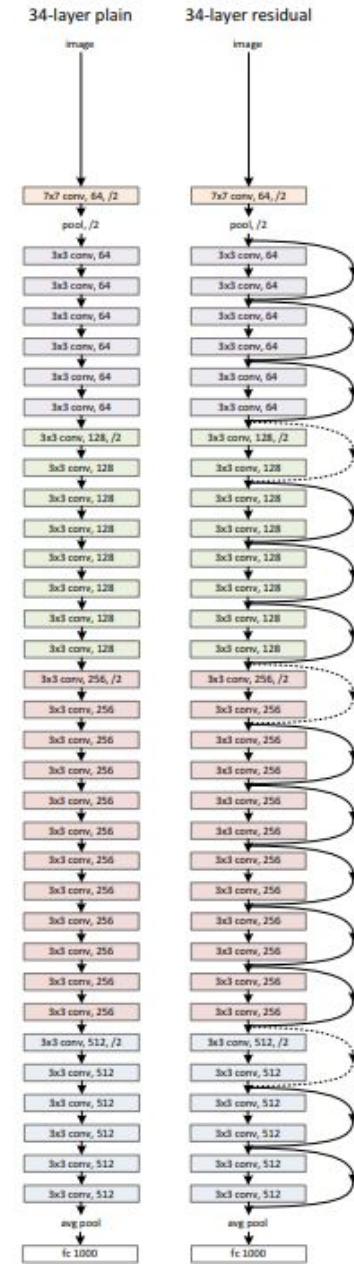
# Res Net

- Allows us to train really deep networks
  - “Identity Mappings in Deep Residual Networks”, 2016 – trained a **1001-layer deep** ResNet to outperform its shallower counterparts
- Helps us with vanishing and exploding gradients
- It is easy for ResNet to learn the identity function

# Res Net

1st places on:

- ImageNet detection
- ImageNet localization
- COCO detection
- COCO segmentation



# Res Net

- Batch Normalization after every CONV layer
- He initialization
- Momentum gradient descent (0.9)
  - Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

# Conv nets – the big picture

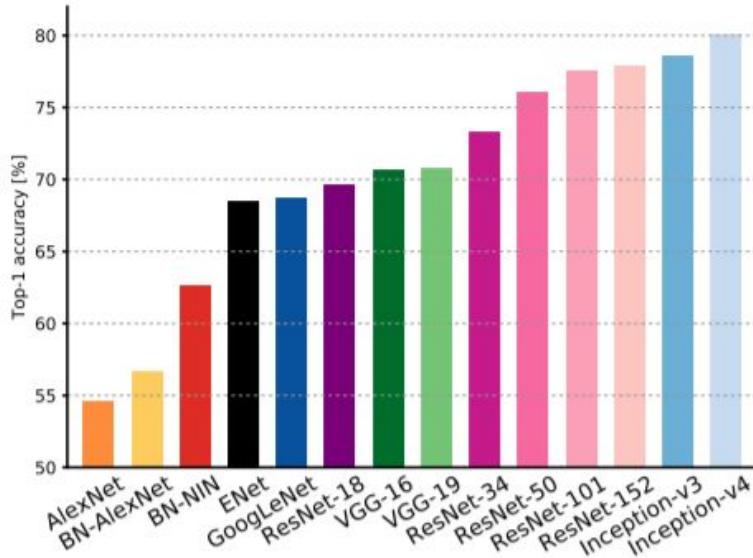


Figure 1: **Top1 vs. network.** Single-crop top-1 validation accuracies for top scoring single-model architectures. We introduce with this chart our choice of colour scheme, which will be used throughout this publication to distinguish effectively different architectures and their correspondent authors. Notice that networks of the same group share the same hue, for example ResNet are all variations of pink.

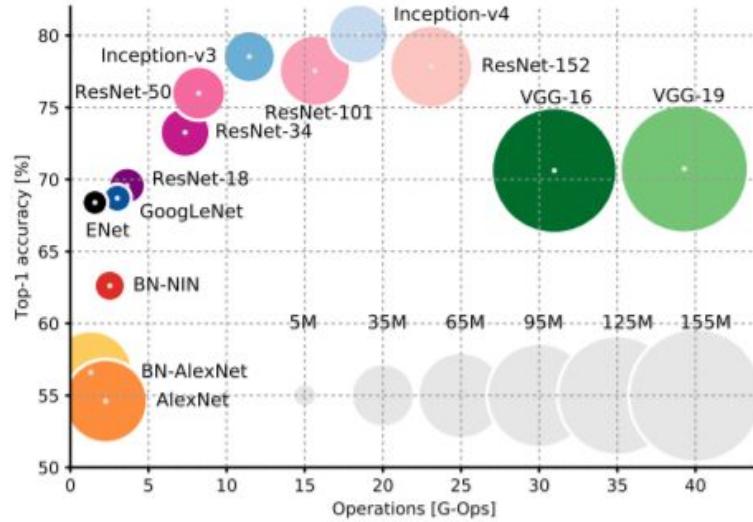
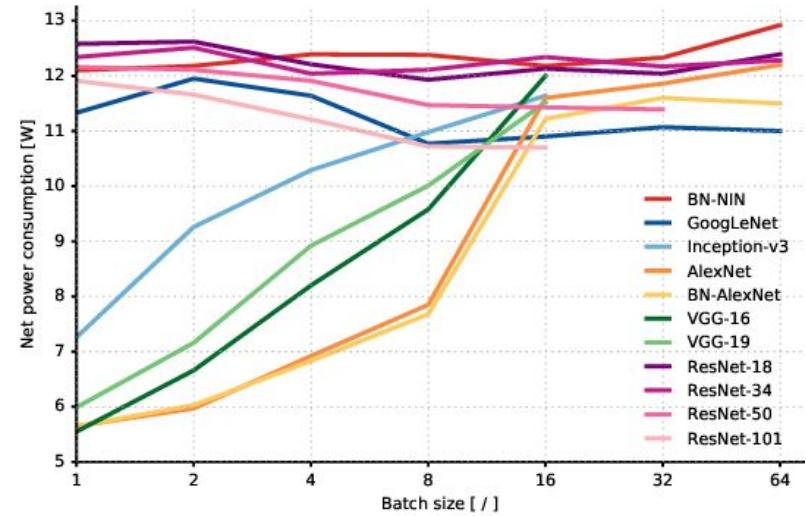
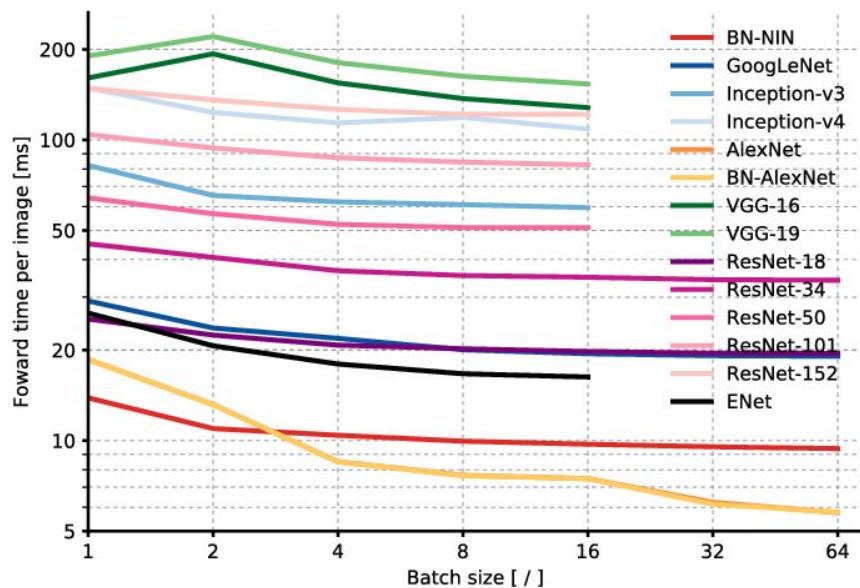


Figure 2: **Top1 vs. operations, size  $\propto$  parameters.** Top-1 one-crop accuracy versus amount of operations required for a single forward pass. The size of the blobs is proportional to the number of network parameters; a legend is reported in the bottom right corner, spanning from  $5 \times 10^6$  to  $155 \times 10^6$  params. Both these figures share the same y-axis, and the grey dots highlight the centre of the blobs.

# Conv nets – the big picture



# How to read a research paper?

- 1. Read the Title, the abstract and the figures**
  - get a general sense of the concepts in the paper
- 2. Read the introduction + conclusions + figures + skim the rest**
  - author(s) try to summarize their work carefully to clarify for the reviewer why their paper should be accepted for publication
- 3. Read the paper but skip the math**
- 4. Read the whole thing but skip the parts that don't make sense**
  - some of it doesn't make sense (it's not unusual), it's okay to skim it initially.

# Computer Vision and Deep Learning

Lecture 7

# Last time

- CNN architectures
  - Alexnet
  - ZFNet
  - VGG
  - started GoogLe Net

# Mobilenets

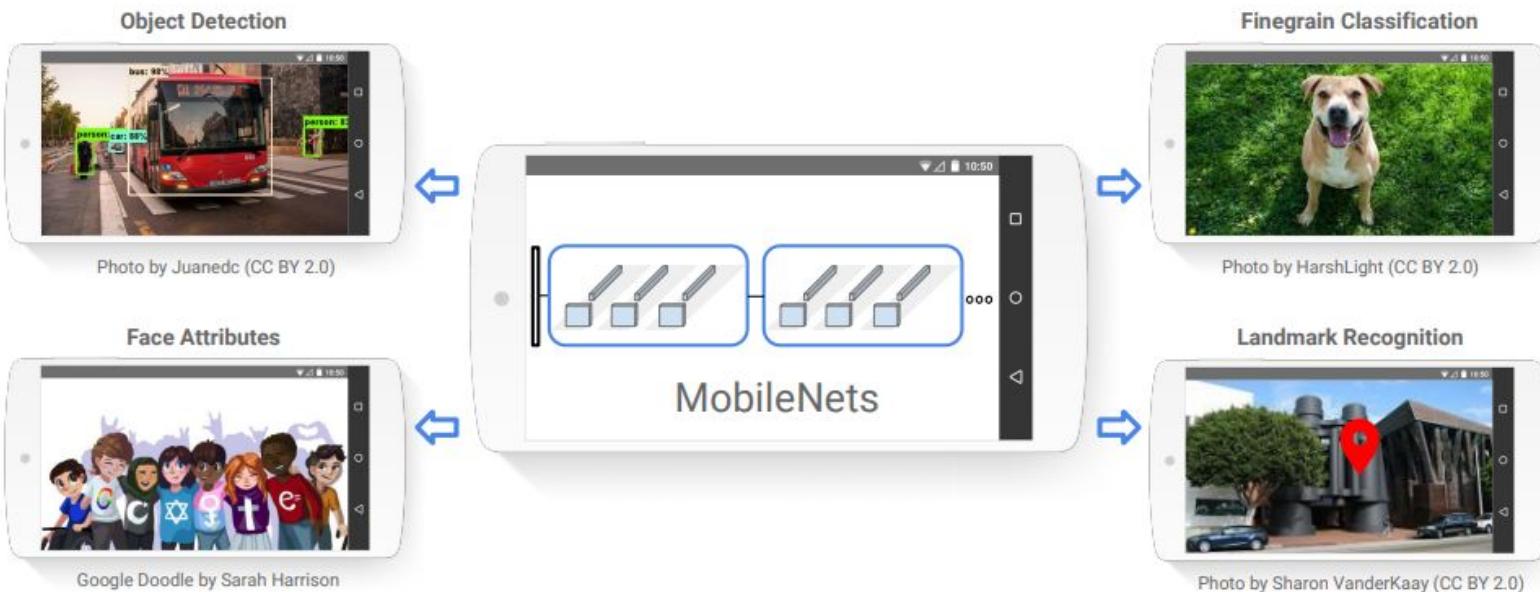
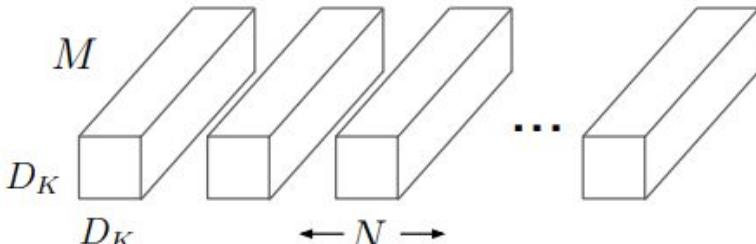
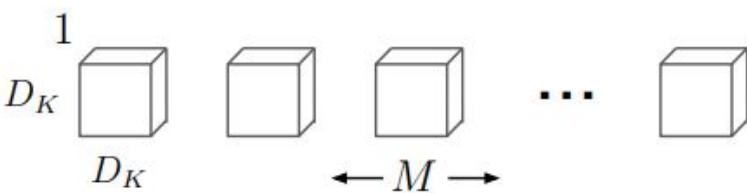


Figure 1. MobileNet models can be applied to various recognition tasks for efficient on device intelligence.

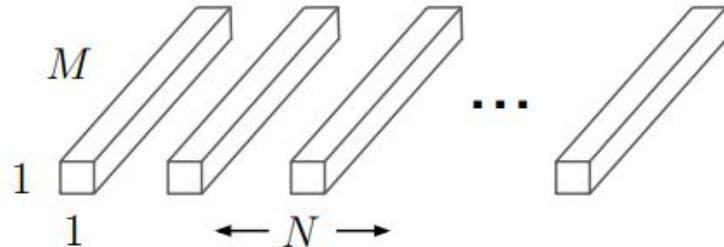
# Mobilenets



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



Standard convolutions have the computational cost of:

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F \quad (2)$$

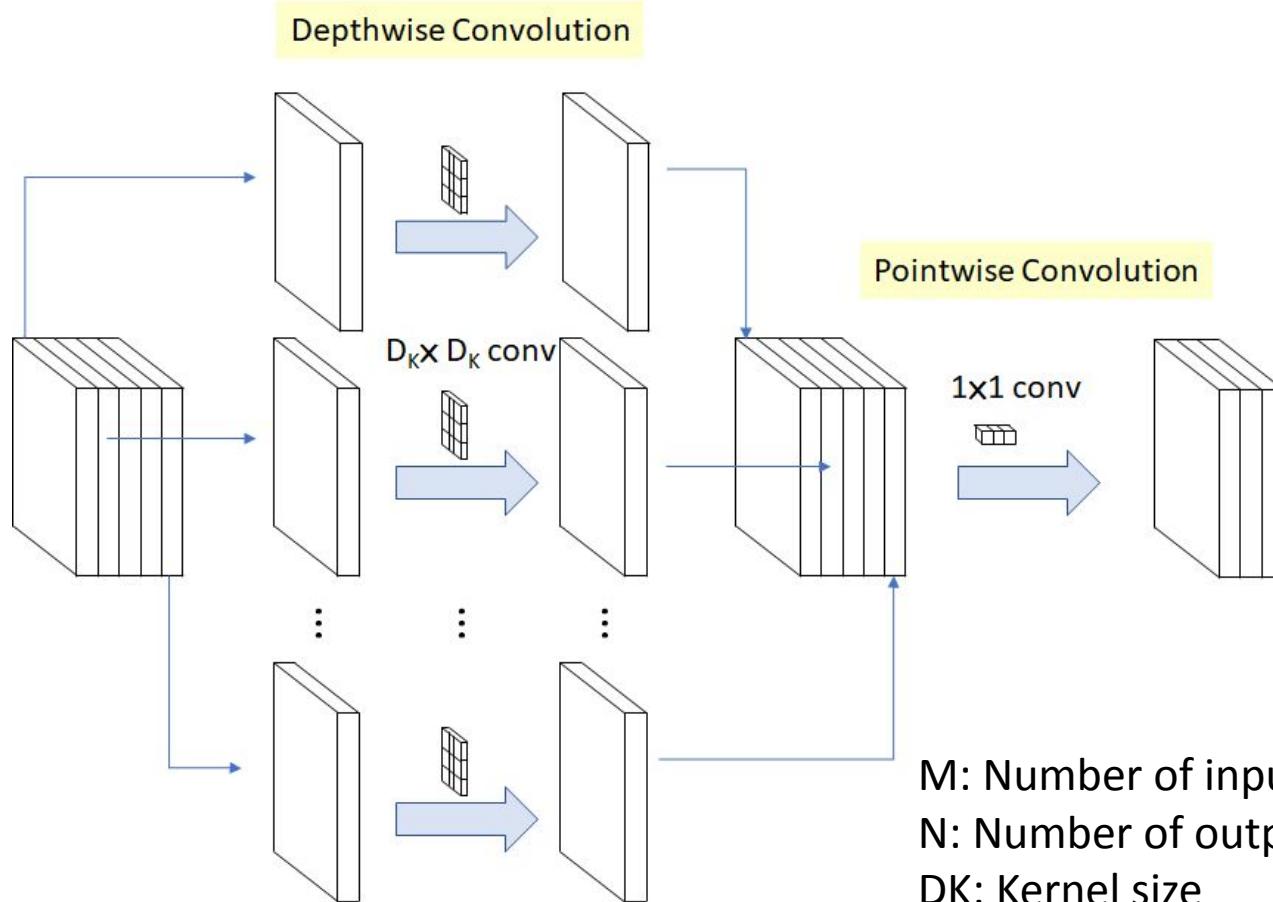
Depthwise convolution has a computational cost of:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F \quad (4)$$

By expressing convolution as a two step process of filtering and combining we get a reduction in computation of:

$$\begin{aligned}
 & \frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} \\
 = & \frac{1}{N} + \frac{1}{D_K^2}
 \end{aligned}$$

# Depth-wise separable convolutions



Operation cost:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F$$

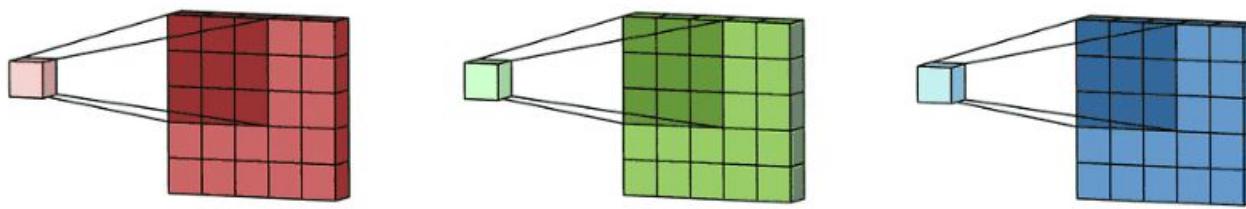
Depthwise convolution

vs.

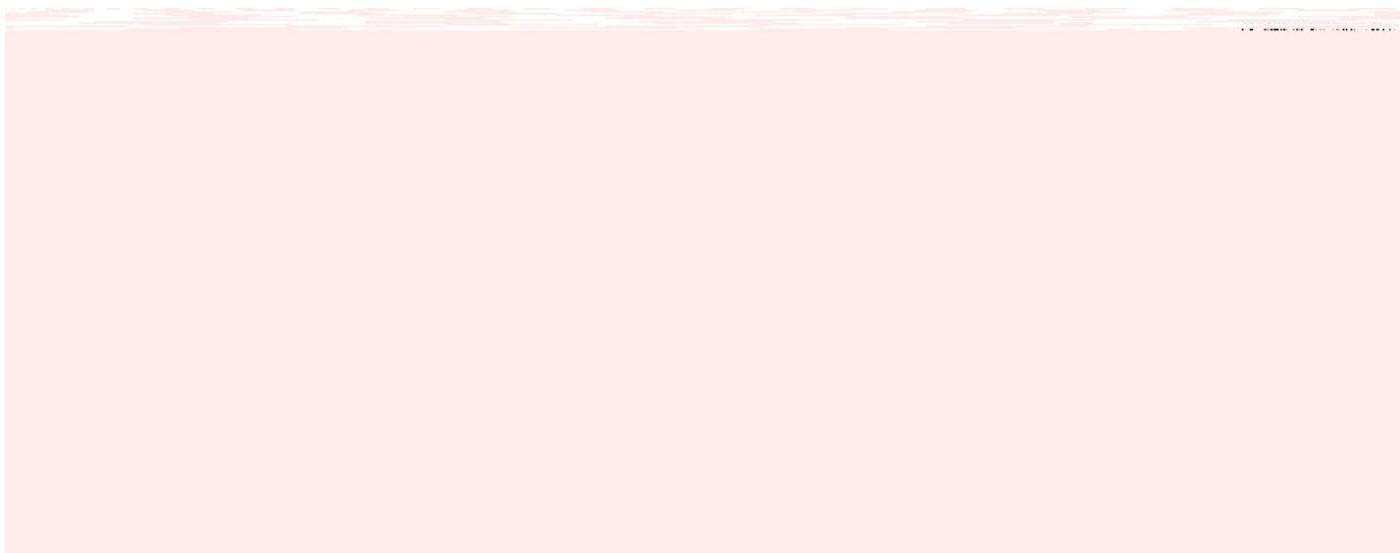
$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$$

Classical convolution

# Depth-wise separable convolutions



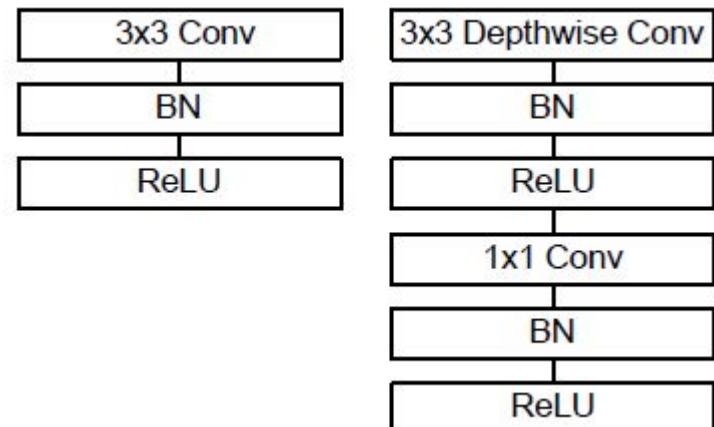
# Depth-wise separable convolutions



# Mobilenets

Table 1. MobileNet Body Architecture

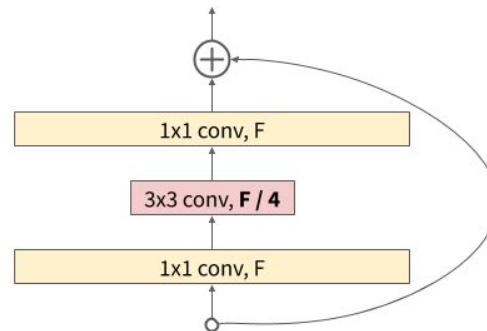
Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5 \times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool $7 \times 7$	$7 \times 7 \times 1024$
FC / s1	$1024 \times 1000$	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$



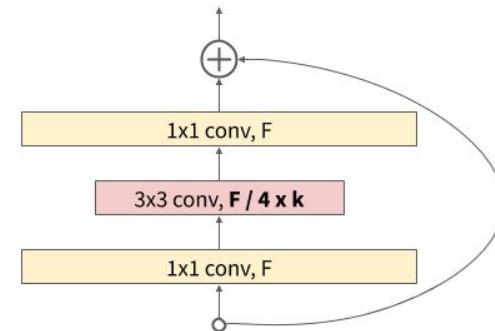
# Other architectures

# Wide RESNET

- Use wider residual blocks (larger number of filters)
- 50 layer wide resnet is better than 101 layer classical resnet



ResNet bottleneck



Wide ResNet bottleneck

# ResNext

- Again increase the width of the networks, but this time through parallel branches in a block

	setting	top-1 err (%)	top-5 err (%)
<i>1× complexity references:</i>			
ResNet-101	1 × 64d	22.0	6.0
ResNeXt-101	32 × 4d	21.2	5.6
<i>2× complexity models follow:</i>			
ResNet- <b>200</b> [15]	1 × 64d	21.7	5.8
ResNet-101, wider	1 × <b>100d</b>	21.3	5.7
ResNeXt-101	<b>2 × 64d</b>	20.7	5.5
ResNeXt-101	<b>64 × 4d</b>	<b>20.4</b>	<b>5.3</b>

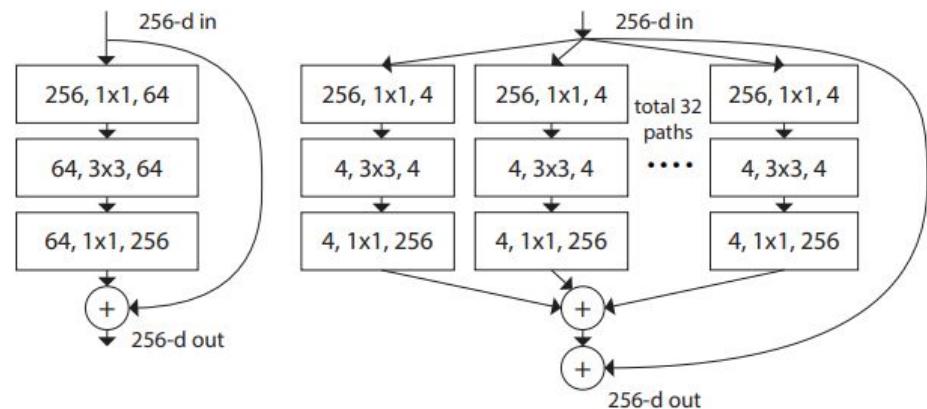
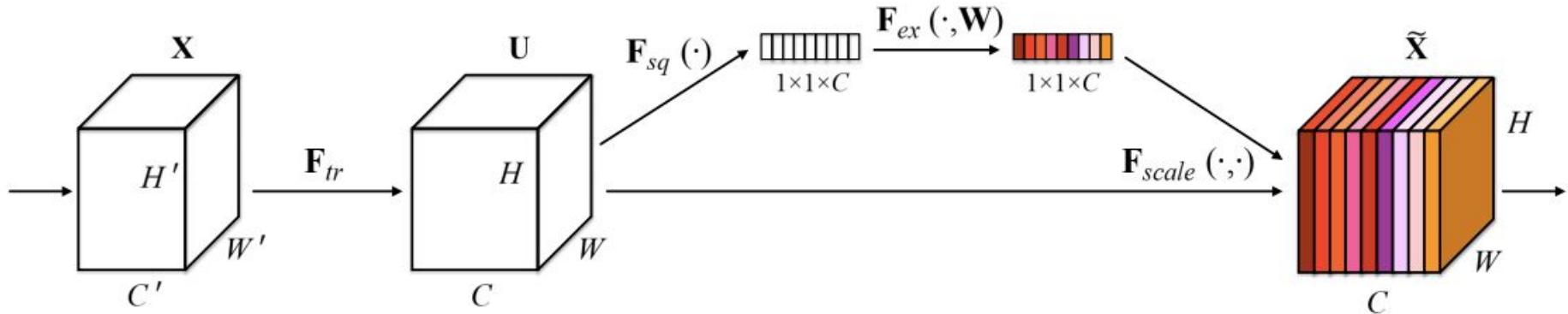


Figure 1. **Left:** A block of ResNet [14]. **Right:** A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

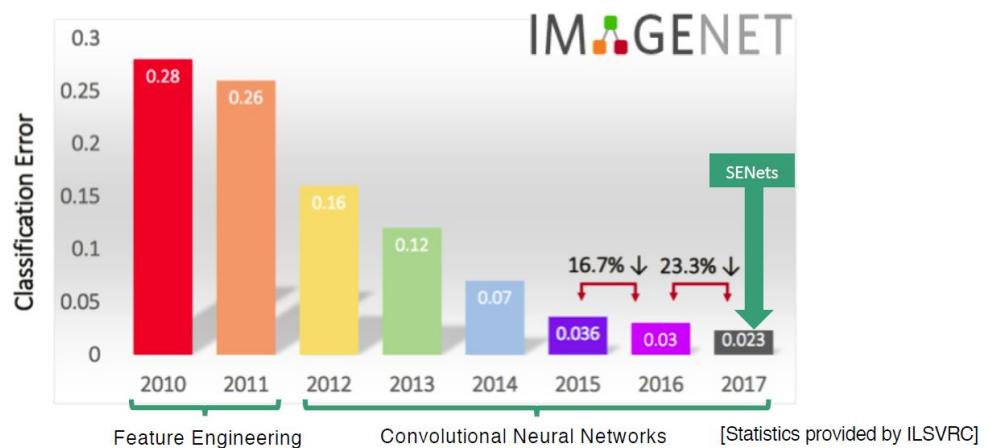
# SENet



ImageNet 2017 winner  
**2.3% top-5 accuracy**

Compute weight for each feature map:

- Global average pooling
- 2FC layers



# SENet

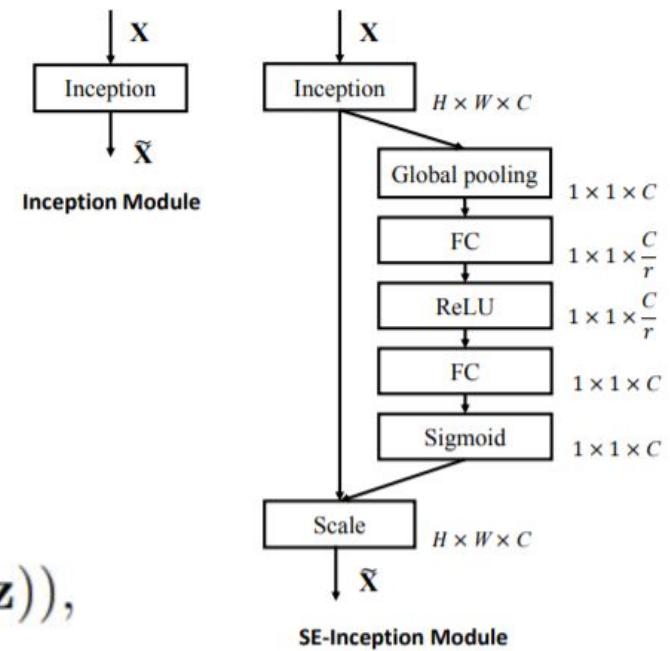
Squeeze (Global Information Embedding) squeeze global spatial information into a channel descriptor

$$z_c = \mathbf{F}_{sq}(\mathbf{u}_c) = \frac{1}{H \times W} \sum_{i=1}^H \sum_{j=1}^W u_c(i, j).$$

## Excitation (Adaptive Recalibration)

- capture channel-wise dependencies
- dynamics conditioned on the input

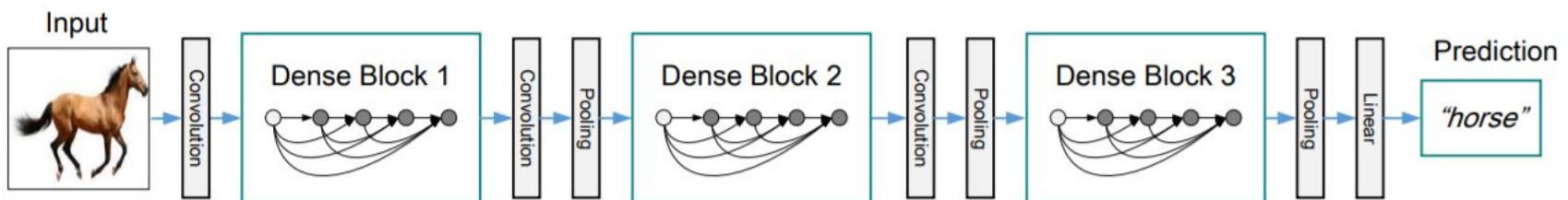
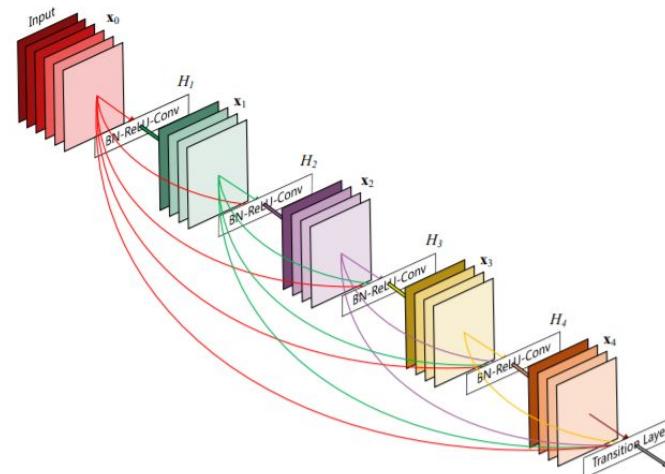
$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(g(\mathbf{z}, \mathbf{W})) = \sigma(\mathbf{W}_2 \delta(\mathbf{W}_1 \mathbf{z})),$$



# Densely connected networks

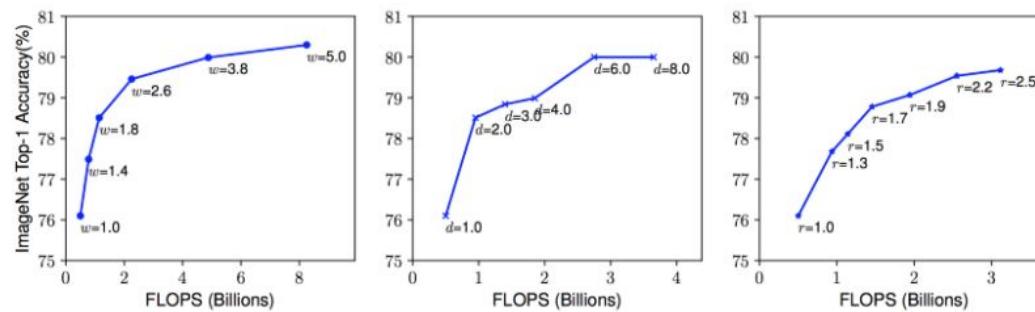
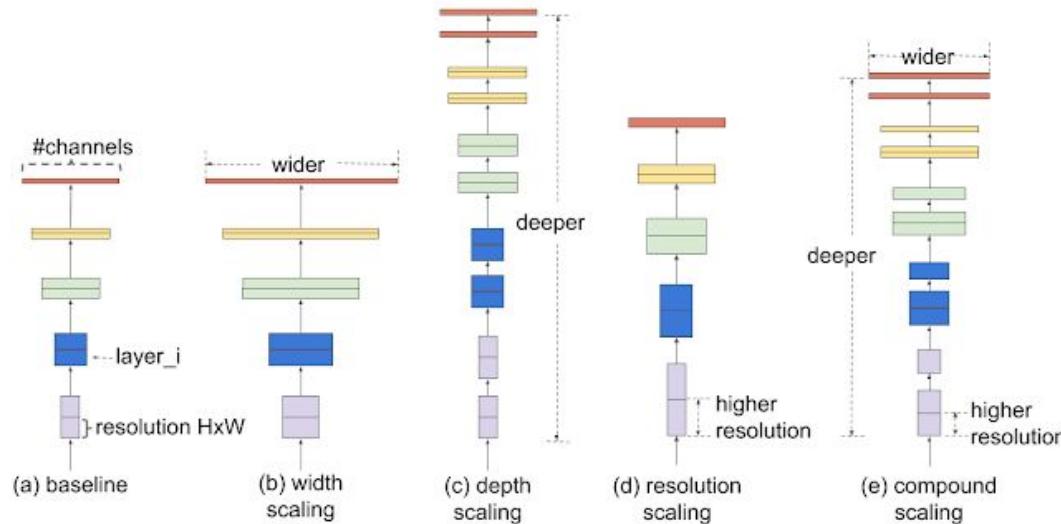
**Dense block:** each layer is connected to every other layer in feed-forward fashion

- strengthens feature propagation,
- encourages feature reuse
- helps with the vanishing gradients problem



**Figure 2:** A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

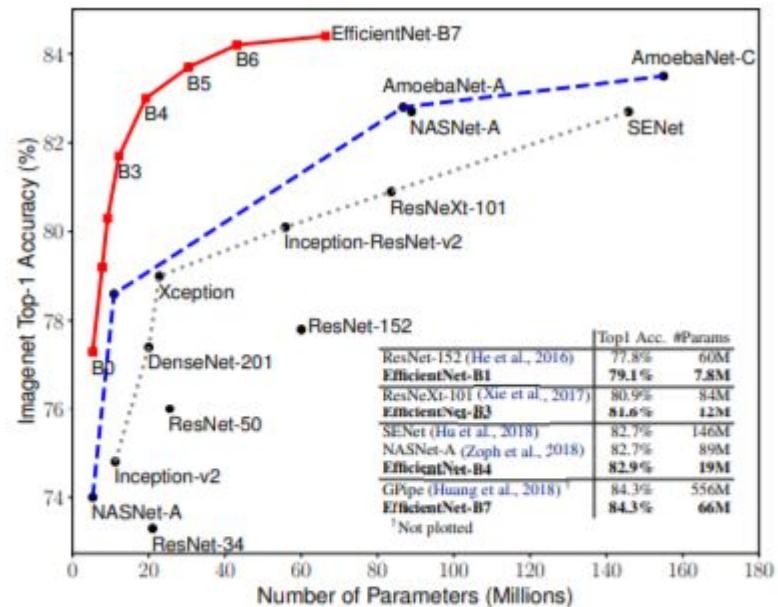
# Efficient nets



Scaling Up a Baseline Model with Different Network Width (w), Depth (d), and Resolution (r) Coefficients.  
Bigger networks with larger width, depth, or resolution tend to achieve higher accuracy, but the accuracy gain quickly saturates after reaching 80%, demonstrating the limitation of single dimension scaling.

# Efficient nets

- carefully balancing network depth, width, and resolution can lead to better performance
- design a new baseline network and scale it up to obtain a family of models, called *EfficientNets*



**Figure 1. Model Size vs. ImageNet Accuracy.** All numbers are for single-crop, single-model. Our EfficientNets significantly outperform other ConvNets. In particular, EfficientNet-B7 achieves new state-of-the-art 84.3% top-1 accuracy but being 8.4x smaller and 6.1x faster than GPipe. EfficientNet-B1 is 7.6x smaller and 5.7x faster than ResNet-152. Details are in Table 2 and 4.

# Efficient nets

- $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$  such that for any new  $\phi$ , the total FLOPS will approximately increase by  $2^\phi$
- if the input image is bigger, the network needs more layers to increase the receptive field and more channels to capture more fine-grained patterns.

Combined scaling:

$$\text{depth: } d = \alpha^\phi$$

$$\text{width: } w = \beta^\phi$$

$$\text{resolution: } r = \gamma^\phi$$

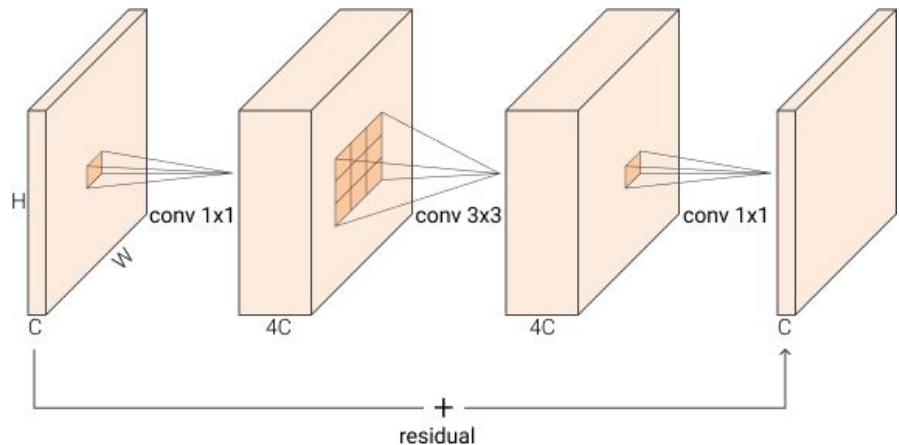
$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

- Random grid search to determine  $\alpha = 1.2$ ,  $\beta = 1.1$ ,  $\gamma = 1.1$
- Fix  $\alpha$ ,  $\beta$ ,  $\gamma$  as constants and scale up baseline network with different  $\varphi$  (EfficientNet-B1 to B7)

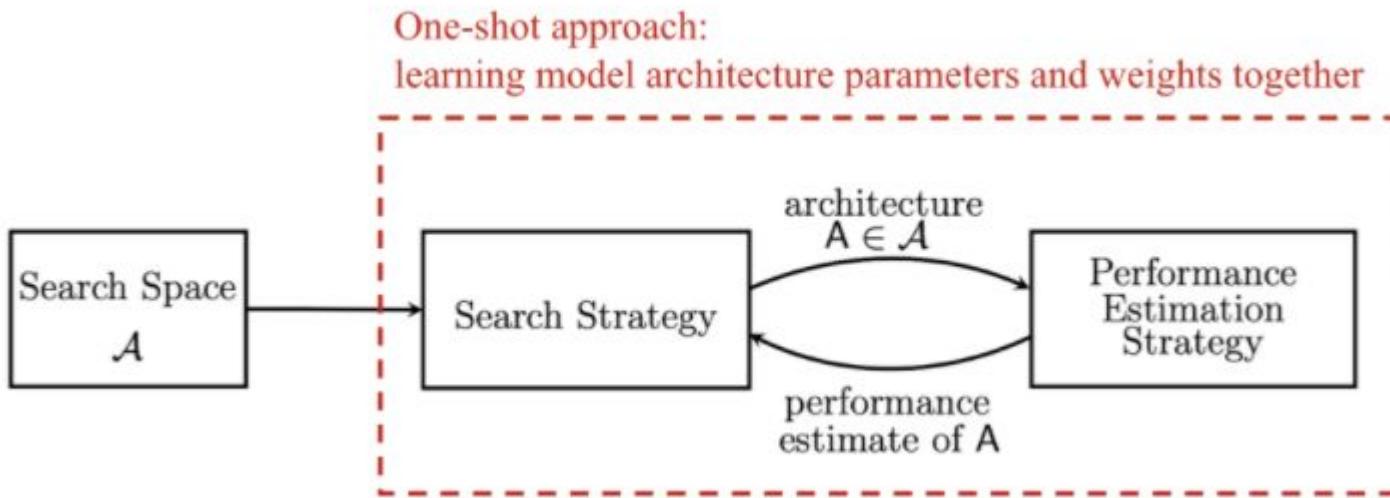
**Table 1. EfficientNet-B0 baseline network** – Each row describes a stage  $i$  with  $\hat{L}_i$  layers, with input resolution  $\langle \hat{H}_i, \hat{W}_i \rangle$  and output channels  $\hat{C}_i$ . Notations are adopted from equation 2.

Stage $i$	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	$224 \times 224$	32	1
2	MBCConv1, k3x3	$112 \times 112$	16	1
3	MBCConv6, k3x3	$112 \times 112$	24	2
4	MBCConv6, k5x5	$56 \times 56$	40	2
5	MBCConv6, k3x3	$28 \times 28$	80	3
6	MBCConv6, k5x5	$14 \times 14$	112	3
7	MBCConv6, k5x5	$14 \times 14$	192	4
8	MBCConv6, k3x3	$7 \times 7$	320	1
9	Conv1x1 & Pooling & FC	$7 \times 7$	1280	1



MBConv

# Neural network architectural search



**Search space:** define a set of operations and how they can be connected.

**Search strategy:** samples a population of network architecture candidates: child model performance metrics as rewards and optimizes to generate high-performance architecture candidates.

**Performance estimation strategy:** measure, estimate, or predict the performance of the proposed child models in order to obtain feedback for the search algorithm to learn

# Summary – CNN architectures

- Networks are getting more and more deep
- The trend is to not use fully connected layers anymore
- Pre-trained models on large datasets are available
- ResNet and SENet currently good defaults to use
- Convolutional networks design is still a flourishing research area
  - aspects of network architectures are continuously investigated and improved

# Summary – CNN architectures

**Alexnet**

SeNet

**VGG**

Wide Resnet

**Inception**

Densely

**ResNet**

connected nets

Mobile nets

Efficient nets

# Transfer learning

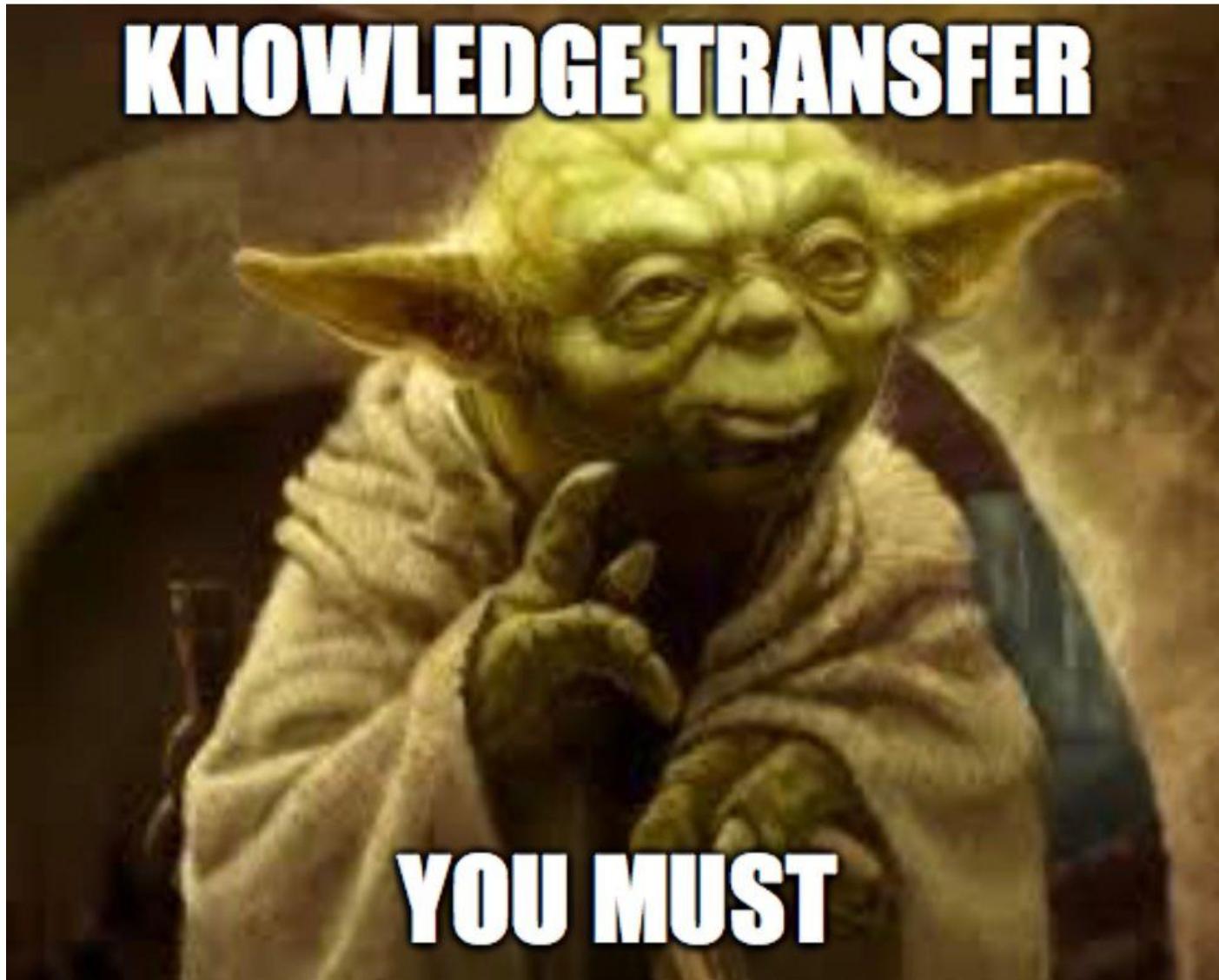
# Transfer learning

Have some dataset for a problem you want to solve but it has < ~1M images?

1. Find a very large dataset that has similar data, train a big ConvNet there
2. Transfer learn to your dataset

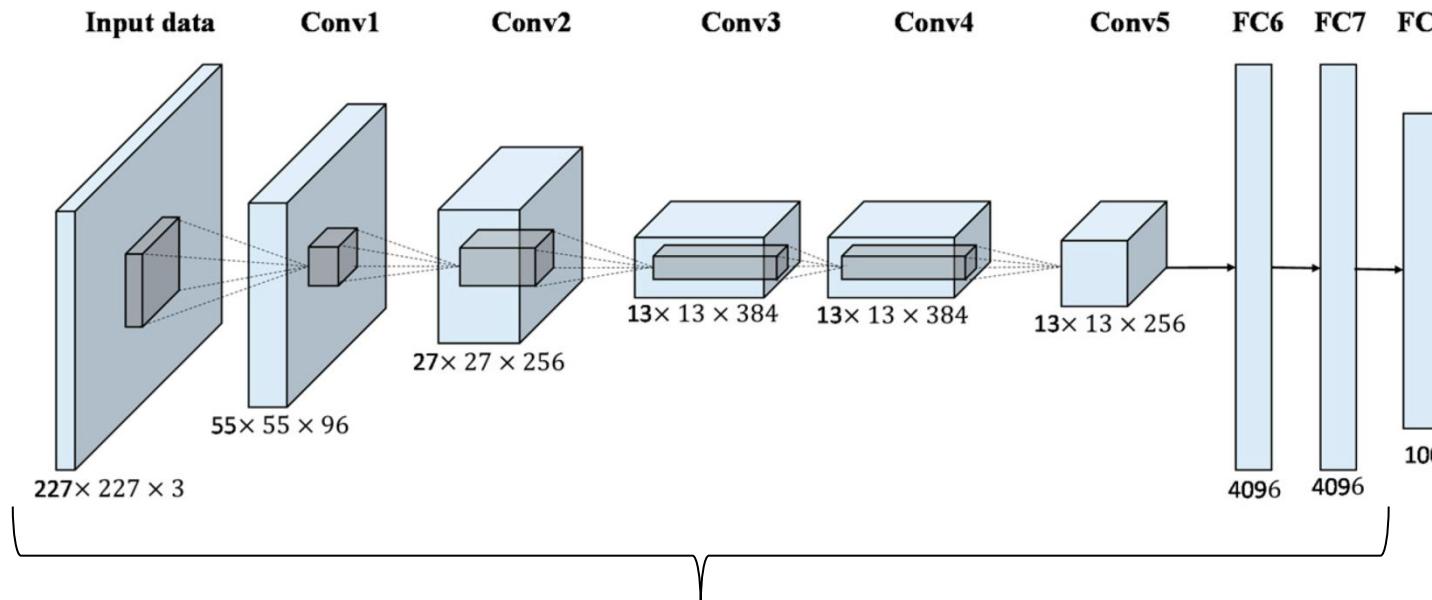
**It's the norm, not the exception!**

# Transfer learning



# Transfer learning – small database

Train on a similar large dataset: ImageNet, COCO etc.

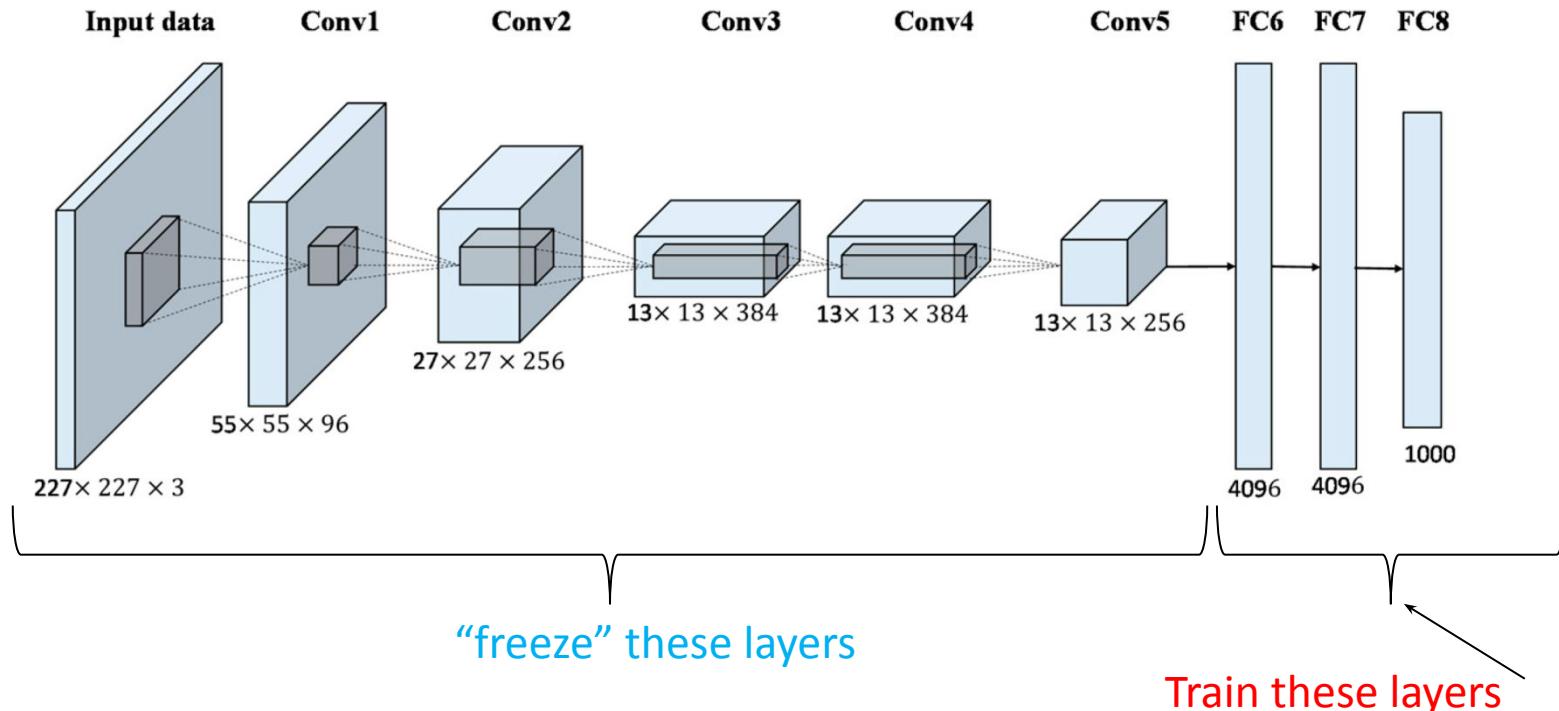


"freeze" these layers

Change the last dense layer to C classes, reinitialize and train it

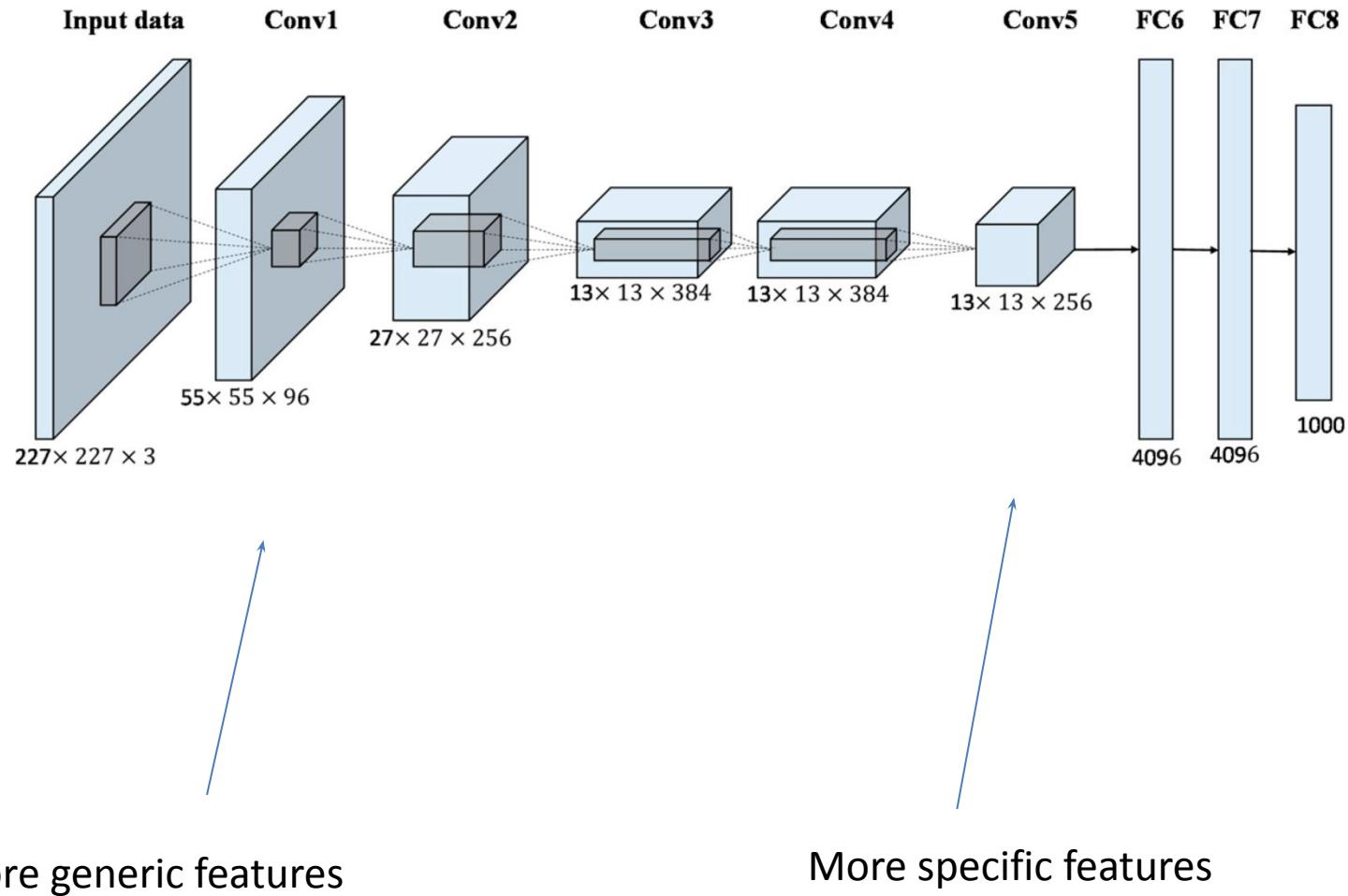
# Transfer learning – larger database

Train on a similar large dataset: ImageNet, COCO etc.



Use a lower learning rate when finetuning! 1/10 of the original learning rate is a good starting point

# Transfer learning



# Transfer learning

	<b>Similar large dataset</b>	<b>Different large dataset</b>
<b>Few training data</b>	Change and train the last layer	 linear classifier from different layers in the net ?
<b>Quite a lot of training data</b>	Fine-tune a few layers	Fine-tune a larger number of layers

# Transfer learning

- TensorFlow:  
<https://github.com/tensorflow/models>
- PyTorch: <https://github.com/pytorch/vision>



<https://modelzoo.co/>

# Multi-task learning

Case study

# An All-In-One Convolutional Neural Network for Face Analysis (2017)

- <https://arxiv.org/pdf/1611.00851.pdf>
- Key ideas that we should take from this paper:
  - End to end deep learning
  - Multitask learning
  - Designing new loss functions



Fig. 1: The proposed method can simultaneously detect faces, predict their landmarks locations, pose angles, smile expression, gender, age as well as the identity from any unconstrained face image.

# Multi-task learning

- When?
  - Training on a set of tasks that could benefit from having shared lower-level features
  - Amount of data you have for each task is quite similar
  - Can train a big enough network to do well on all tasks

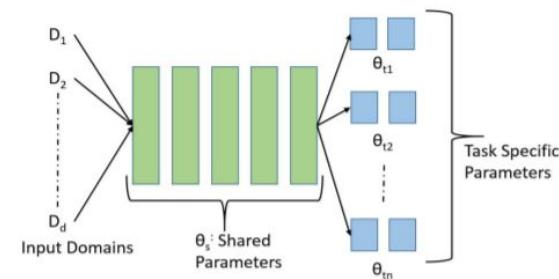


Fig. 2: A general multitask learning framework for deep CNN architecture. The lower layers are shared among all the tasks and input domains.

# An All-In-One Convolutional Neural Network for Face Analysis (2017)

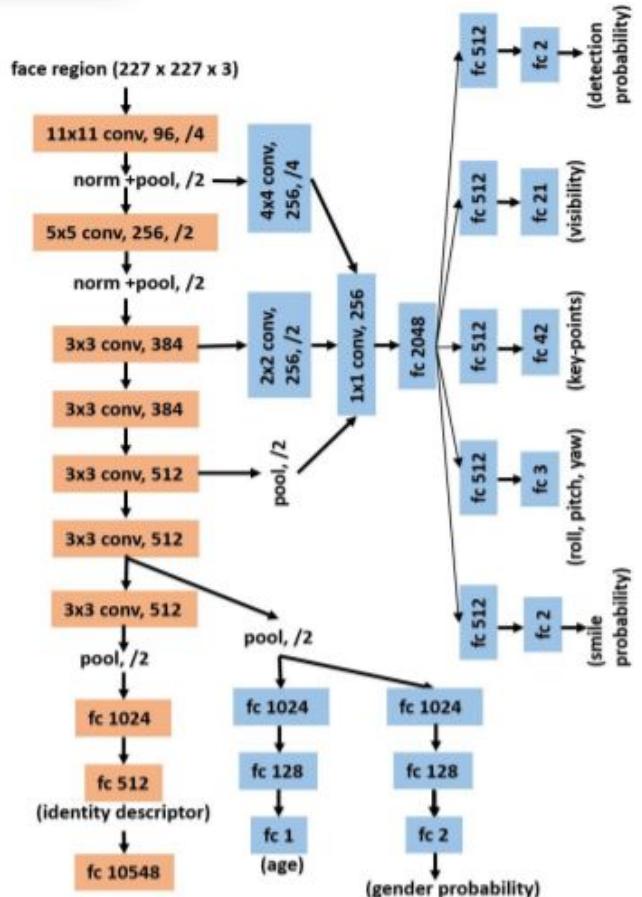


Fig. 3: CNN Architecture for the proposed method. Each layer is represented by filter kernel size, type of layer, number of feature maps and the filter stride. Orange represents the pre-trained network from Sankaranarayanan et al. [41], while blue represents added layers for MTL.

$$L_G = -(1 - g) \cdot \log(1 - p_g) - g \cdot \log(p_g),$$

$$L_S = -(1 - s) \cdot \log(1 - p_s) - s \cdot \log(p_s),$$

$$L_A = (1 - \lambda) \frac{1}{2} (y - a)^2 + \lambda \left( 1 - \exp\left(-\frac{(y - a)^2}{2\sigma^2}\right) \right),$$

$$L_R = \sum_{c=0}^{10547} -y_c \cdot \log(p_c),$$

Dataset	Face Analysis Tasks	# training samples
CASIA [51]	Identification, Gender	490,356
MORPH [39]	Age, Gender	55,608
IMDB+WIKI [40]	Age, Gender	224,840
Adience [27]	Age	19,370
CelebA [31]	Smile, Gender	182,637
AFLW [24]	Detection, Pose, Fiducials	20,342
Total		<b>993,153</b>

# Face recognition

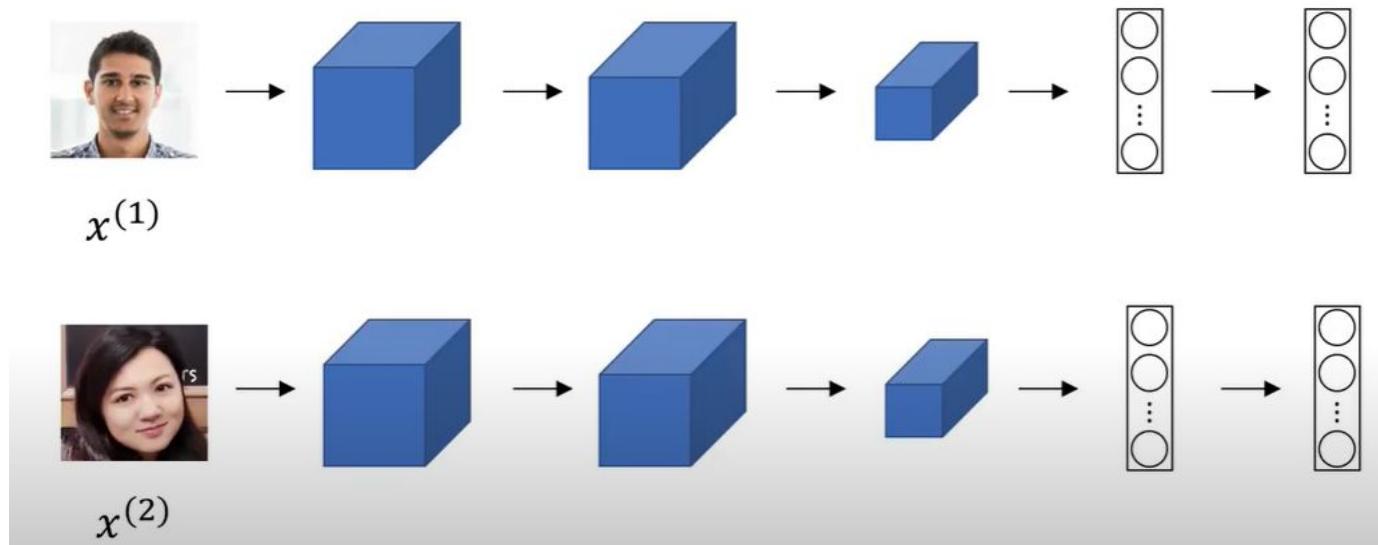
Case study

# Face recognition – case study

- Face verification: validating a claimed identity based on the image of a face, and either accepting or rejecting the identity claim (*one-to-one matching*)
- Face recognition: is to identify a person based on the image of a face: the face image has to be compared with all the registered persons (*one-to-many matching*)
- One shot learning

# Siamese networks

Two or more inputs are encoded and the output features are compared

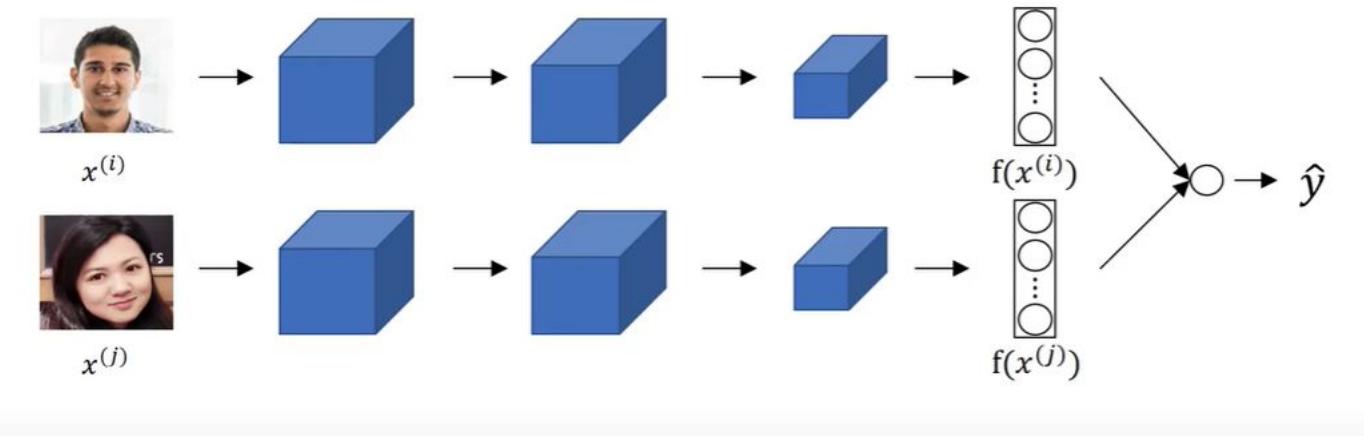


If  $x^{(i)}, x^{(j)}$  are the same person,  $\|f(x^{(i)}) - f(x^{(j)})\|^2$  is small.

If  $x^{(i)}, x^{(j)}$  are different persons,  $\|f(x^{(i)}) - f(x^{(j)})\|^2$  is large.

[https://github.com/pytorch/examples/tree/main/siamese\\_network](https://github.com/pytorch/examples/tree/main/siamese_network)

# Face verification as a binary classification problem



- $$\sigma\left(\sum_k \omega_i |f(x^{(i)})_k - f(x^{(j)})_k| + b\right)$$

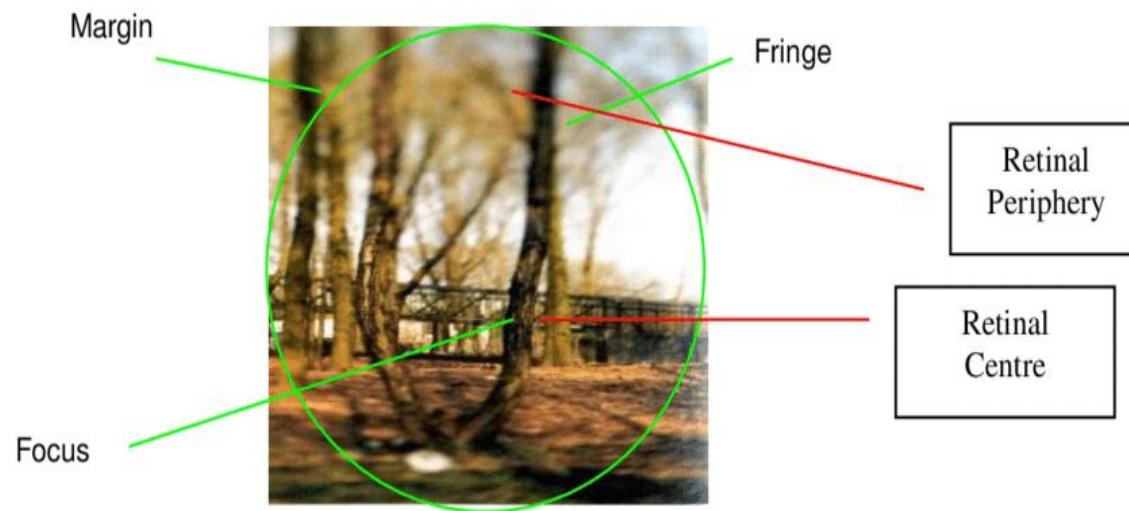
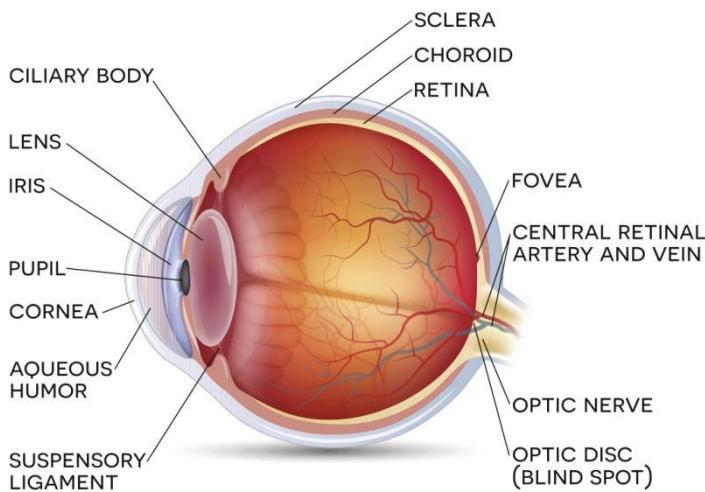
# Computer Vision and Deep Learning

Lecture 8

# How to read a research paper

<https://www.scientifica.uk.com/neurowire/gradhacks-a-guide-to-reading-research-papers>

# Attention



humans exploit a sequence of partial glimpses  
and selectively focus on salient parts in order  
to capture visual structure better

# CBAM – Convolutional Block Attention Module

## Convolutional Block Attention Module (CBAM):

- a simple yet effective attention module for feed-forward convolutional neural networks

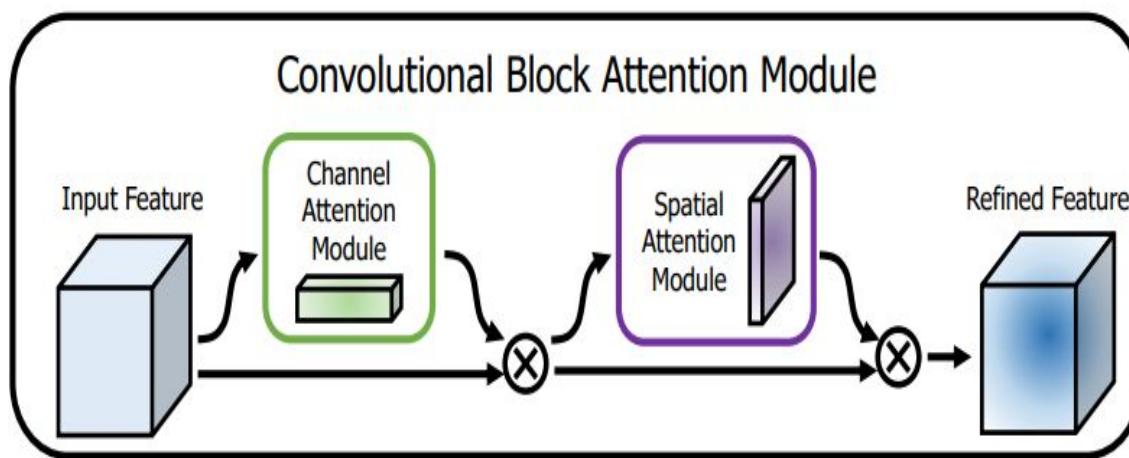
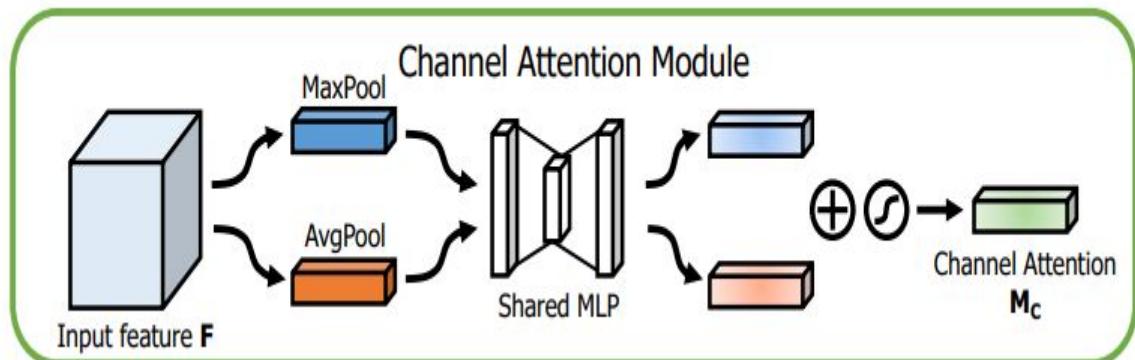


Fig. 1: The overview of CBAM. The module has two sequential sub-modules: *channel* and *spatial*. The intermediate feature map is adaptively refined through our module (CBAM) at every convolutional block of deep networks.

# Channel Attention

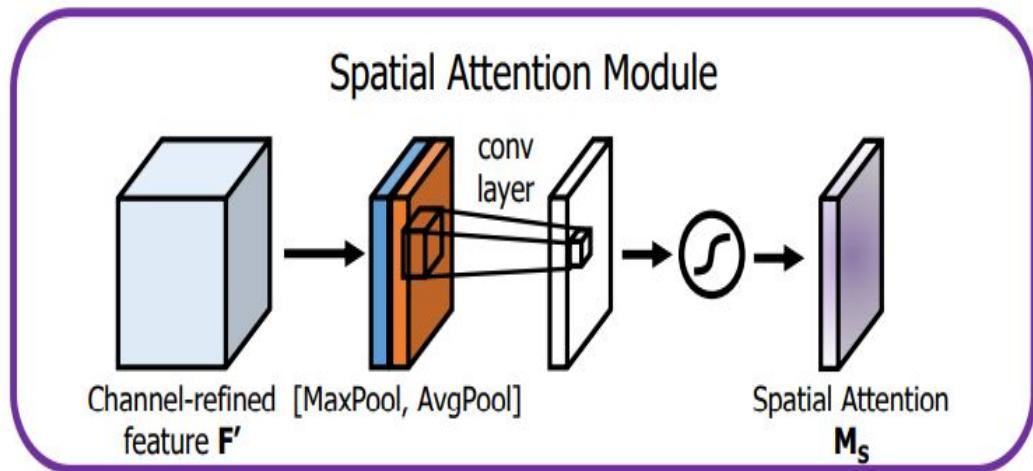
- GAP(Global Average Pooling)
  - Aggregate spatial information
- GMP
  - Preserve richer context information
- Multilayer perceptron (MLP)
- Sigmoid activation
  - Gives the weights for each channel

$$\begin{aligned} M_c(F) &= \sigma(MLP(AvgPool(F)) + MLP(MaxPool(F))) \\ &= \sigma(W_1(W_0(F_{avg}^c)) + W_1(W_0(F_{max}^c))), \end{aligned}$$



# Spatial Attention

- Two pooling operations
- 1x1 Conv
- Sigmoid activation
  - Can be applied element-wise to all the positions in the input feature map



$$\begin{aligned}\mathbf{M}_s(\mathbf{F}) &= \sigma(f^{7 \times 7}([AvgPool(\mathbf{F}); MaxPool(\mathbf{F})])) \\ &= \sigma(f^{7 \times 7}([\mathbf{F}_{avg}^s; \mathbf{F}_{max}^s])),\end{aligned}$$

```
class ChannelAttention(nn.Module):
    def __init__(self, in_planes, ratio=16):
        super(ChannelAttention, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.max_pool = nn.AdaptiveMaxPool2d(1)

        self.fc = nn.Sequential(nn.Conv2d(in_planes, in_planes // 16, 1, bias=False),
                               nn.ReLU(),
                               nn.Conv2d(in_planes // 16, in_planes, 1, bias=False))
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        avg_out = self.fc(self.avg_pool(x))
        max_out = self.fc(self.max_pool(x))
        out = avg_out + max_out
        return self.sigmoid(out)
```

```
class SpatialAttention(nn.Module):
    def __init__(self, kernel_size=7):
        super(SpatialAttention, self).__init__()

        self.conv1 = nn.Conv2d(2, 1, kernel_size, padding=kernel_size//2, bias=False)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        avg_out = torch.mean(x, dim=1, keepdim=True)
        max_out, _ = torch.max(x, dim=1, keepdim=True)
        x = torch.cat([avg_out, max_out], dim=1)
        x = self.conv1(x)
        return self.sigmoid(x)
```

# CBAM – Convolutional Block Attention Module

Architecture	Param.	GFLOPs	Top-1 Error (%)	Top-5 Error (%)
ResNet18 [5]	11.69M	1.814	29.60	10.55
ResNet18 [5] + SE [28]	11.78M	1.814	29.41	10.22
ResNet18 [5] + CBAM	11.78M	1.815	<b>29.27</b>	<b>10.09</b>
ResNet34 [5]	21.80M	3.664	26.69	8.60
ResNet34 [5] + SE [28]	21.96M	3.664	26.13	8.35
ResNet34 [5] + CBAM	21.96M	3.665	<b>25.99</b>	<b>8.24</b>
ResNet50 [5]	25.56M	3.858	24.56	7.50
ResNet50 [5] + SE [28]	28.09M	3.860	23.14	6.70
ResNet50 [5] + CBAM	28.09M	3.864	<b>22.66</b>	<b>6.31</b>
ResNet101 [5]	44.55M	7.570	23.38	6.88
ResNet101 [5] + SE [28]	49.33M	7.575	22.35	6.19
ResNet101 [5] + CBAM	49.33M	7.581	<b>21.51</b>	<b>5.69</b>
WideResNet18 [6] (widen=1.5)	25.88M	3.866	26.85	8.88
WideResNet18 [6] (widen=1.5) + SE [28]	26.07M	3.867	26.21	8.47
WideResNet18 [6] (widen=1.5) + CBAM	26.08M	3.868	<b>26.10</b>	<b>8.43</b>
WideResNet18 [6] (widen=2.0)	45.62M	6.696	25.63	8.20
WideResNet18 [6] (widen=2.0) + SE [28]	45.97M	6.696	24.93	7.65
WideResNet18 [6] (widen=2.0) + CBAM	45.97M	6.697	<b>24.84</b>	<b>7.63</b>
ResNeXt50 [7] (32x4d)	25.03M	3.768	22.85	6.48
ResNeXt50 [7] (32x4d) + SE [28]	27.56M	3.771	<b>21.91</b>	6.04
ResNeXt50 [7] (32x4d) + CBAM	27.56M	3.774	21.92	<b>5.91</b>
ResNeXt101 [7] (32x4d)	44.18M	7.508	21.54	5.75
ResNeXt101 [7] (32x4d) + SE [28]	48.96M	7.512	21.17	5.66
ResNeXt101 [7] (32x4d) + CBAM	48.96M	7.519	<b>21.07</b>	<b>5.59</b>

# Recurrent neural networks

## Other resources:

RNNs:

[\(HIGHLY RECOMMENDED\)](http://karpathy.github.io/2015/05/21/rnn-effectiveness/)

<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

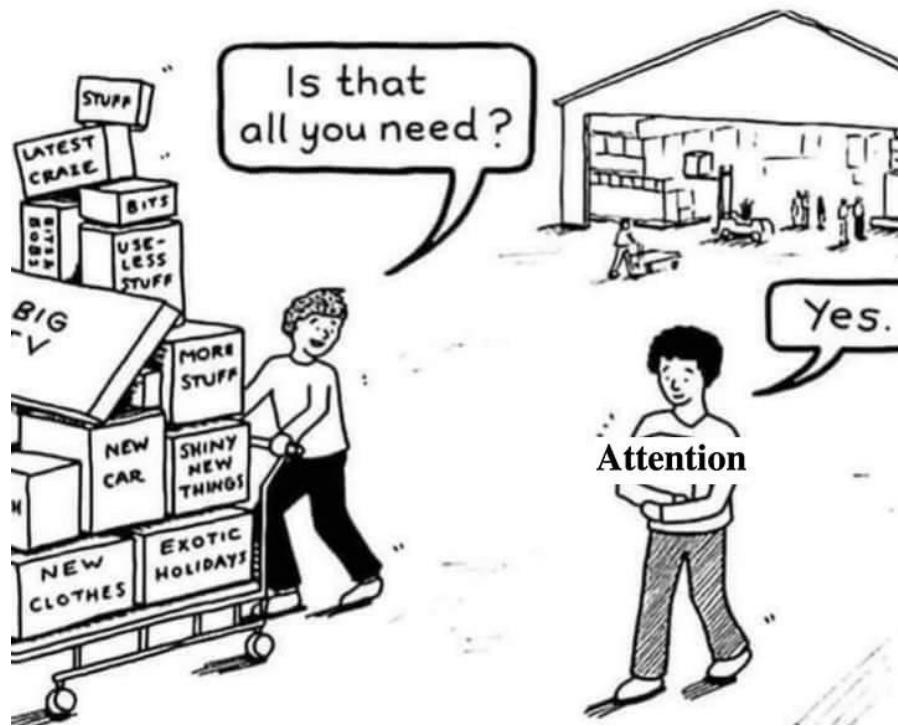
Self attention and transformers (Stanford lecture 2021):

<https://www.youtube.com/watch?v=ptuGIIU5SQQ>

Jay Alammar series on Transformers & BERT:

- <https://jalammar.github.io/illustrated-transformer/>
- <https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/>
- <https://jalammar.github.io/illustrated-bert/>

# Attention is all you need



<https://arxiv.org/abs/1706.03762>

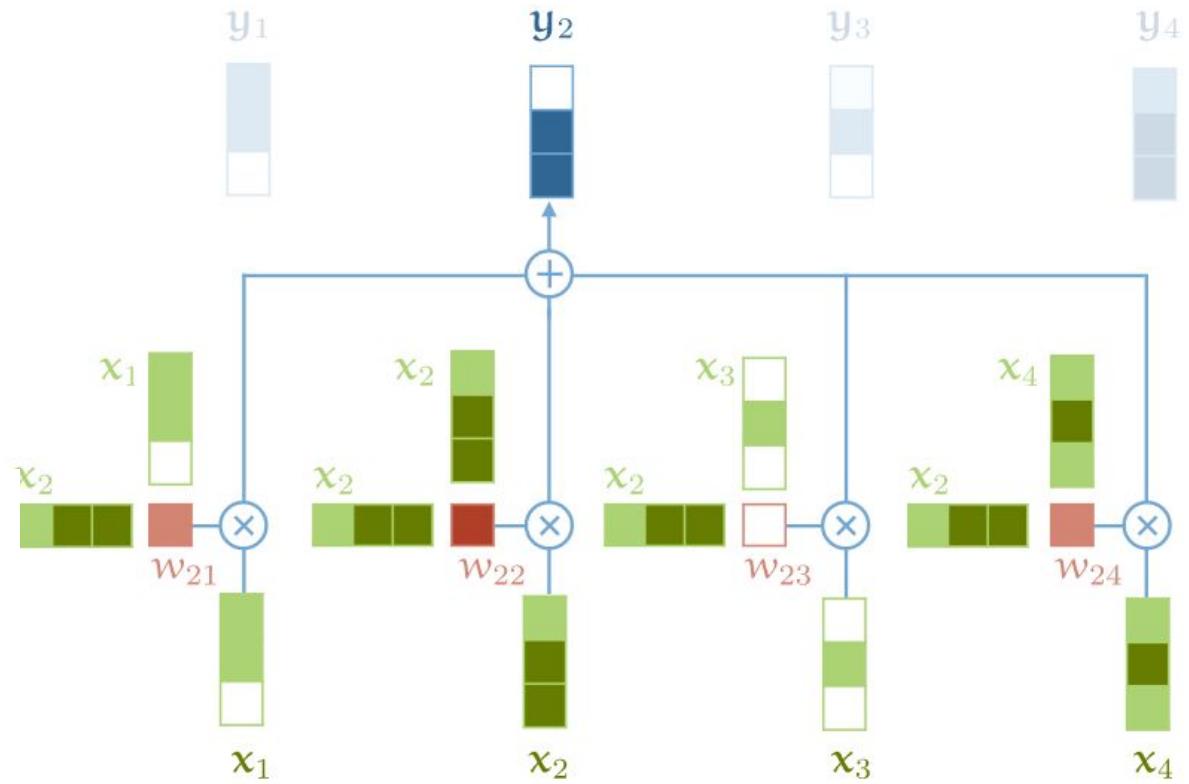
# Self attention

Self-attention, sometimes called intra-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations [4, 27, 28, 22].

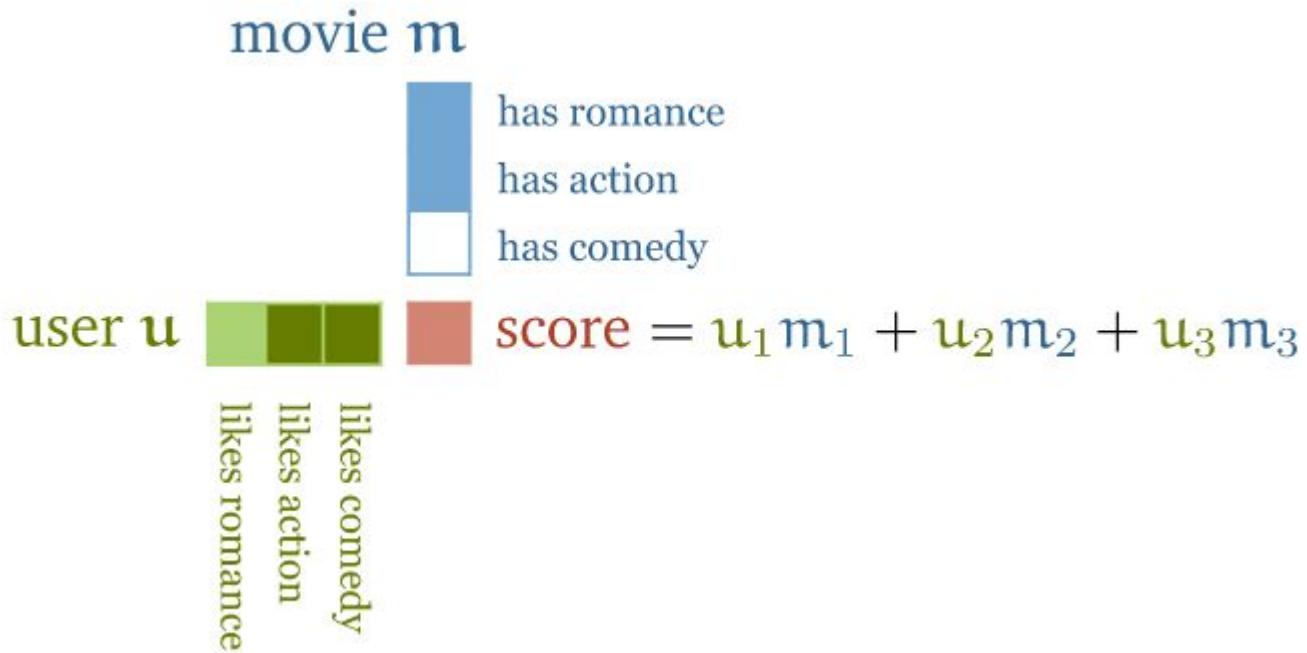
$$y_i = \sum_j w_{ij} x_j .$$

$$w'_{ij} = x_i^T x_j .$$

$$w_{ij} = \frac{\exp w'_{ij}}{\sum_j \exp w'_{ij}} .$$



# Self attention



# Attention: query, keys and values

“An *attention function* can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors.”

$$\mathbf{q}_i = \mathbf{W}_q \mathbf{x}_i \quad \mathbf{k}_i = \mathbf{W}_k \mathbf{x}_i \quad \mathbf{v}_i = \mathbf{W}_v \mathbf{x}_i$$

$$w'_{ij} = \mathbf{q}_i^T \mathbf{k}_j$$

$$w_{ij} = \text{softmax}(w'_{ij})$$

$$\mathbf{y}_i = \sum_j w_{ij} \mathbf{v}_j .$$

# Attention: query, keys and values

$$q_i = W_q x_i \quad k_i = W_k x_i \quad v_i = W_v x_i$$

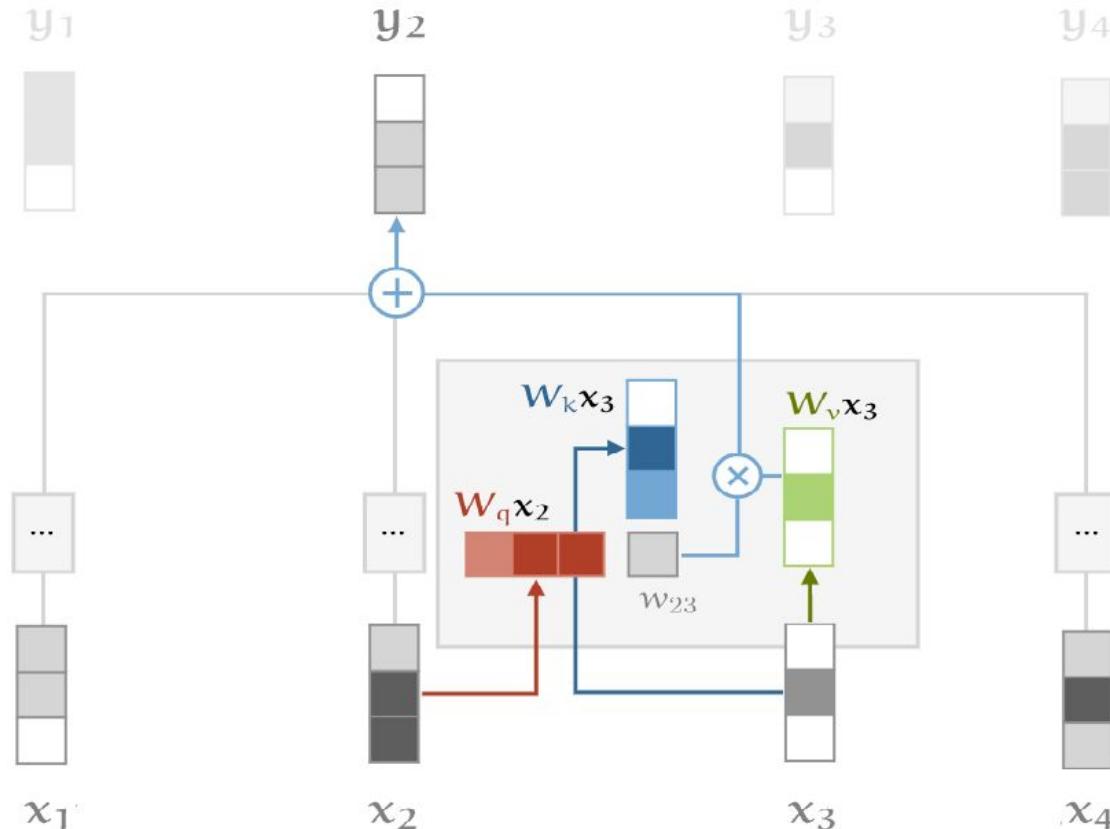


Illustration of the self-attention with **key**, **query** and **value**

# Scaled Dot-Product Attention

We call our particular attention "Scaled Dot-Product Attention" (Figure 2). The input consists of queries and keys of dimension  $d_k$ , and values of dimension  $d_v$ . We compute the dot products of the query with all keys, divide each by  $\sqrt{d_k}$ , and apply a softmax function to obtain the weights on the values.

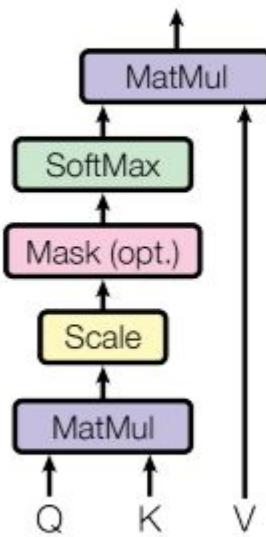
While for small values of  $d_k$  the two mechanisms perform similarly, additive attention outperforms dot product attention without scaling for larger values of  $d_k$  [3]. We suspect that for large values of  $d_k$ , the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients<sup>4</sup>. To counteract this effect, we scale the dot products by  $\frac{1}{\sqrt{d_k}}$ .

$k(d_k$  in the paper) – dimension of the embedding

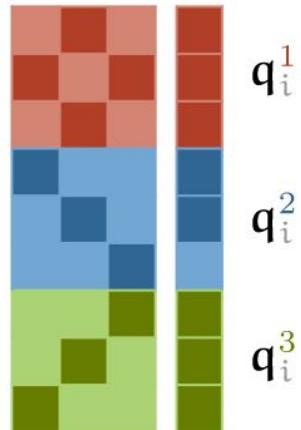
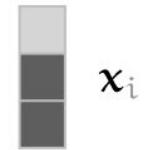
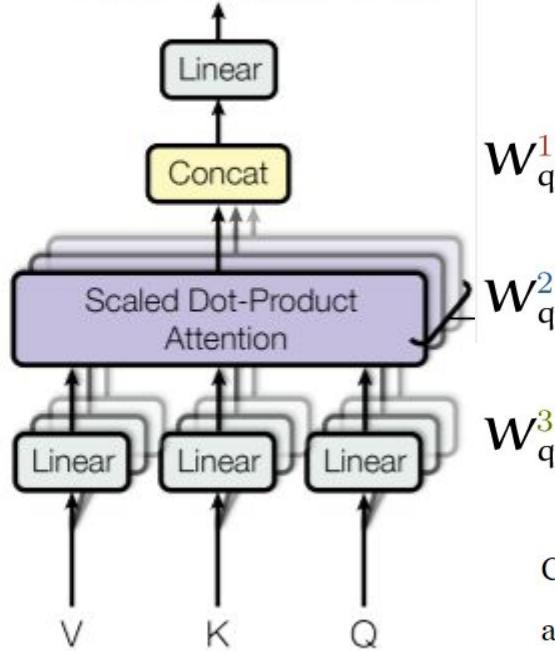
$$w'_{ij} = \frac{\mathbf{q}_i^T \mathbf{k}_j}{\sqrt{k}}$$

# Multi-head attention

Scaled Dot-Product Attention



Multi-Head Attention



Combining three  
attention heads into one  
matrix multiplication  
(for the queries).

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

$$\text{where } \text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

# torch code for self-attention

<https://github.com/pbloem/former>

# Transformer architecture

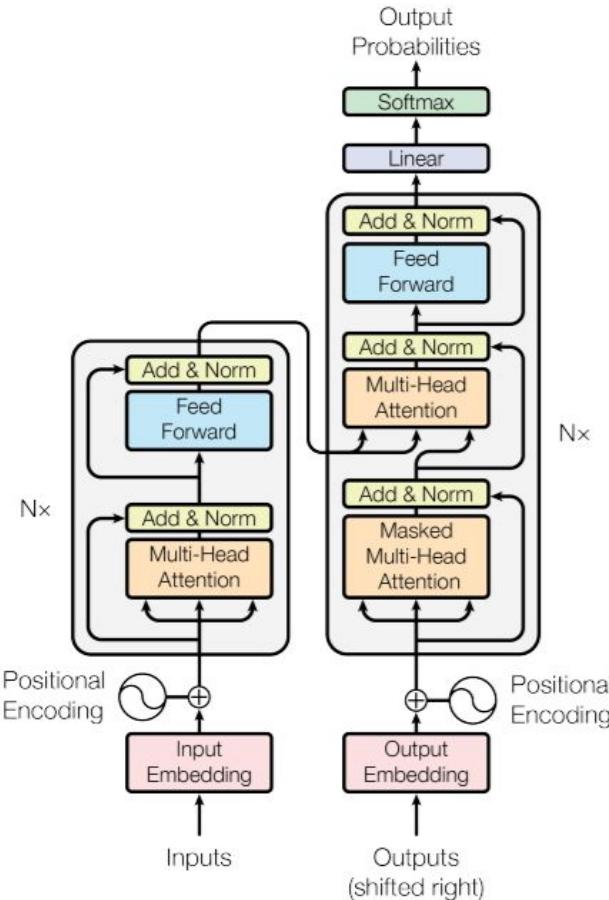


Figure 1: The Transformer - model architecture.

Most competitive neural sequence transduction models have an encoder-decoder structure [5, 2, 35]. Here, the encoder maps an input sequence of symbol representations  $(x_1, \dots, x_n)$  to a sequence of continuous representations  $\mathbf{z} = (z_1, \dots, z_n)$ . Given  $\mathbf{z}$ , the decoder then generates an output sequence  $(y_1, \dots, y_m)$  of symbols one element at a time. At each step the model is auto-regressive [10], consuming the previously generated symbols as additional input when generating the next.

# Positional Encoding

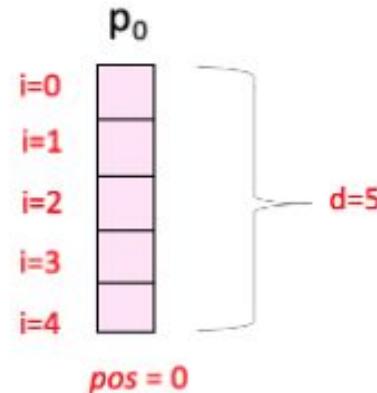
$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

where  $pos$  is the position and  $i$  is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from  $2\pi$  to  $10000 \cdot 2\pi$ . We

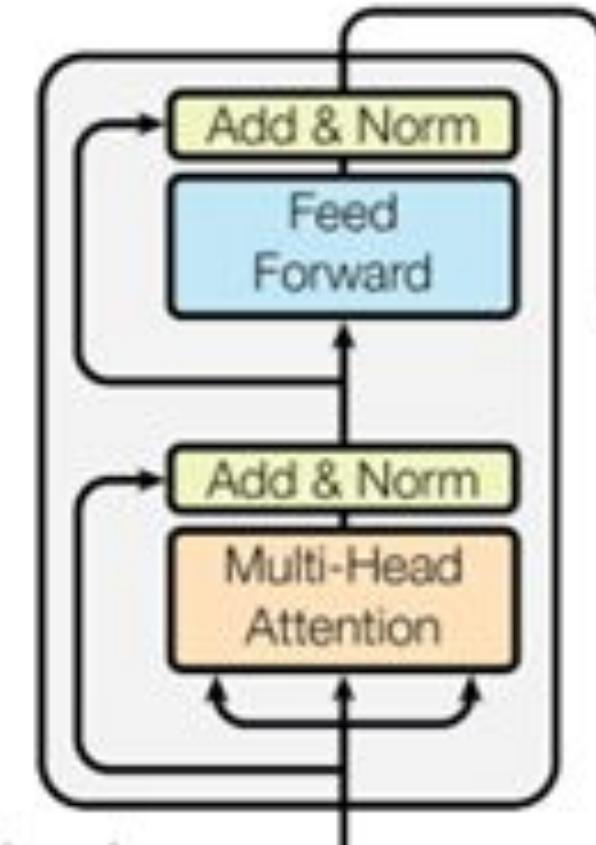
---

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

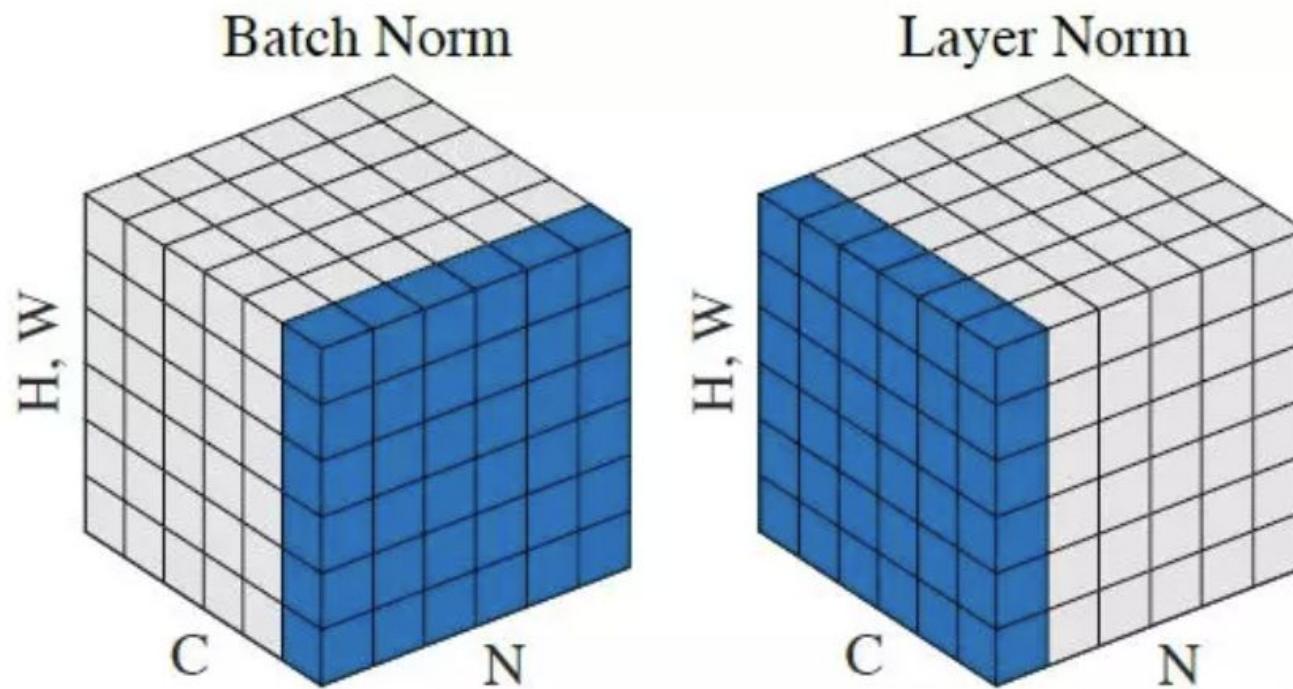


# Transformer: encoder block

- Multi head attention layer
- Layer normalization
- Feed forward layer (applied independently to each vector)
- Layer normalization + Residual connections (before the normalizations)

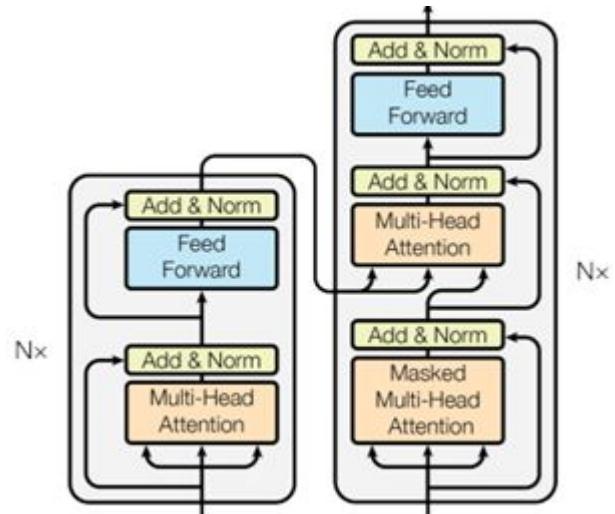


# Batch norm vs Layer norm

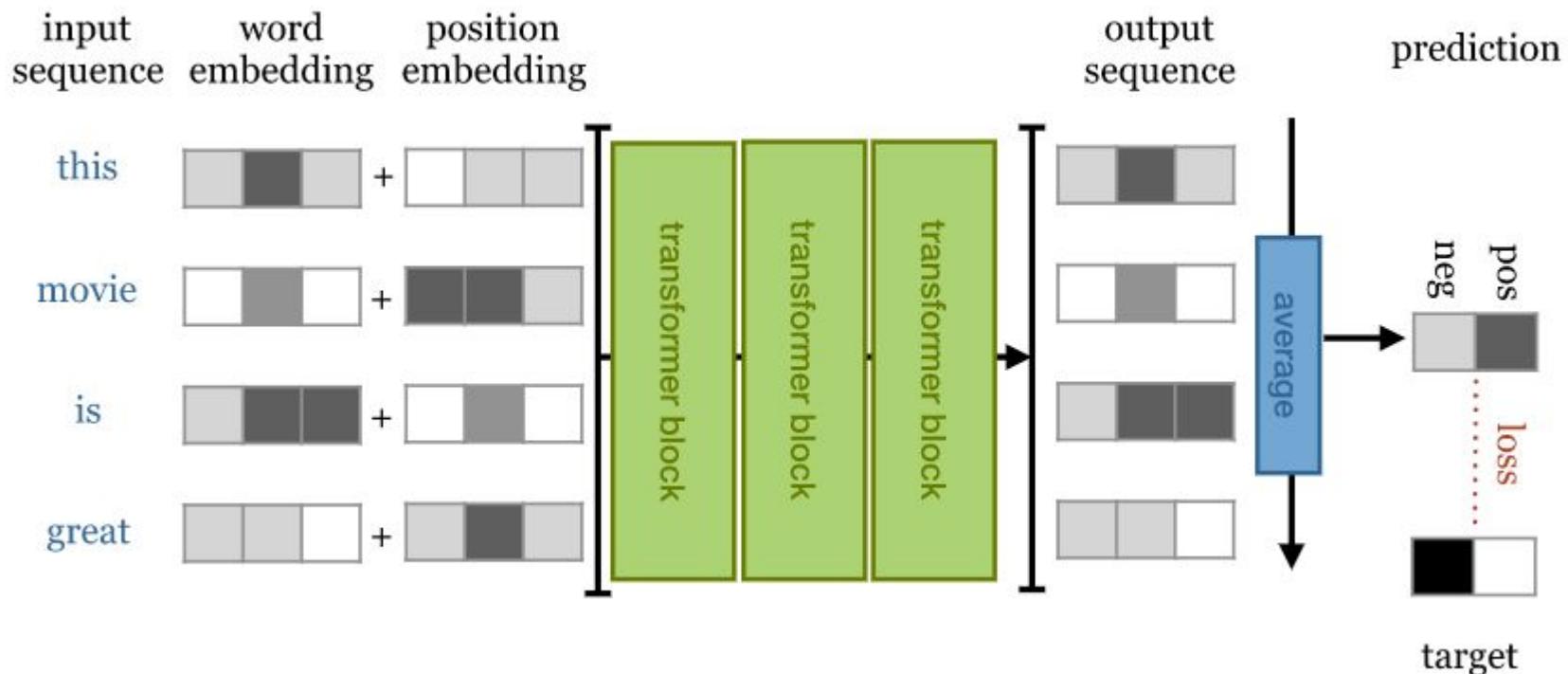


# Transformer: decoder block

- Input: block on the same level in the encoder and the previous level in the decoder
- Masked multi head attention



# Simple transformer for sentiment analysis



# BERT

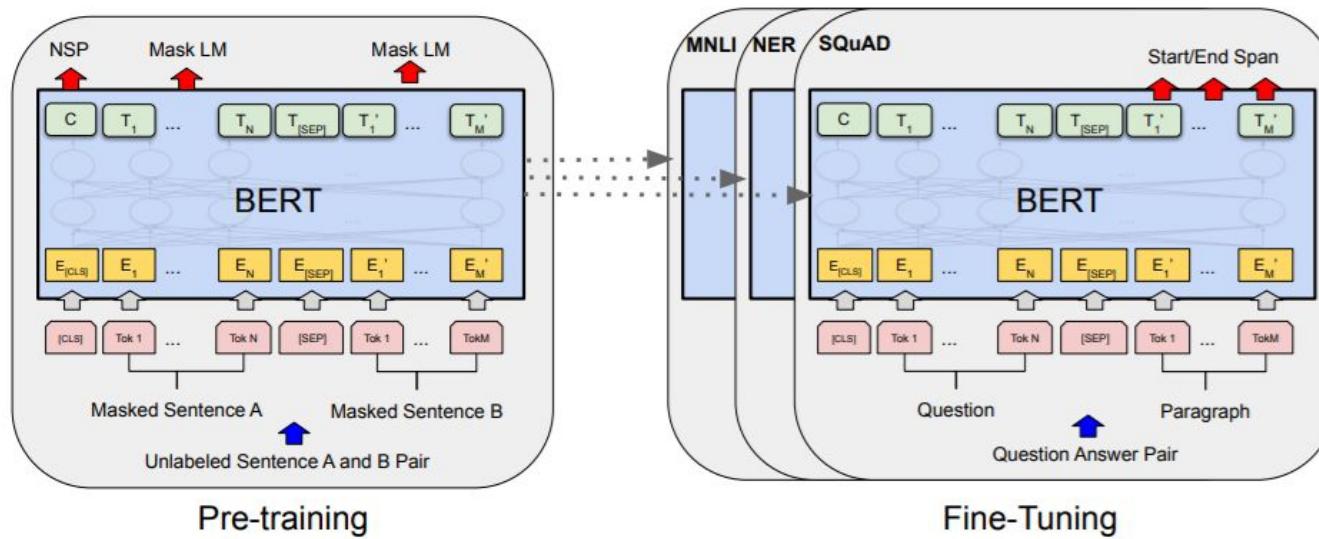


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned.  $[CLS]$  is a special symbol added in front of every input example, and  $[SEP]$  is a special separator token (e.g. separating questions/answers).

# BERT

- ***PRE-TRAINED*** on a large general-domain corpus consisting of 800M words from English books 2.5B words of text from English Wikipedia articles.
  - **Masking:** words are masked out, replaced with a random word or kept as they are. The model is then asked to predict, for **only** these words, what the original words were.

# BERT

- ***PRETRAINING***
  - Masking
  - **Next Sentence Prediction:** Two sequences of about 256 words are sampled that either (a) follow each other directly in the corpus, or (b) are both taken from random places. The model must then predict whether a or b is the case.

# BERT

- Input is prepended with a special <CLS> token. The output vector corresponding to this token is used as a sentence representation in sequence classification tasks like the next sentence classification.

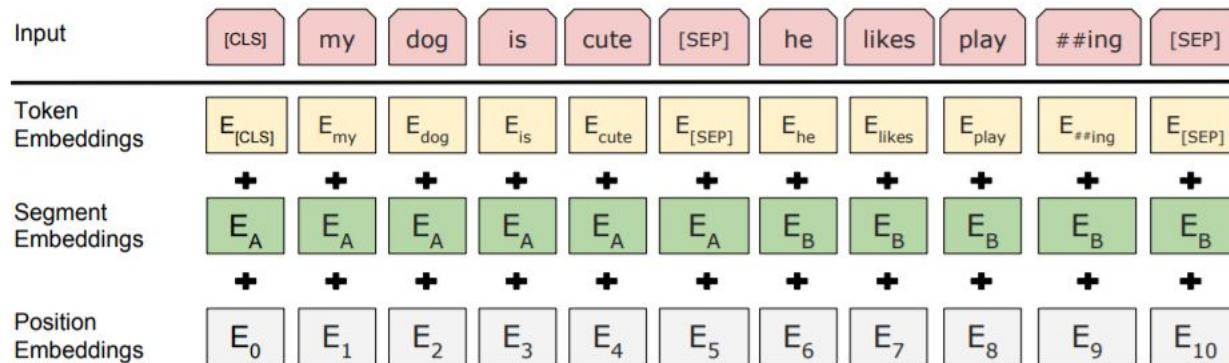


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

# BERT

- **FINE-TUNING**
  - a single task-specific layer is placed after the transformer blocks, which maps the general purpose representation to a task specific output.
- **Classification:** maps the first output token (corresponding to <CLS>) to *softmax* probabilities over the classes

# **An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale**

Vision transformer

Official implementation in JAX:

[https://github.com/google-research/vision\\_transformer#vision-transformer](https://github.com/google-research/vision_transformer#vision-transformer)

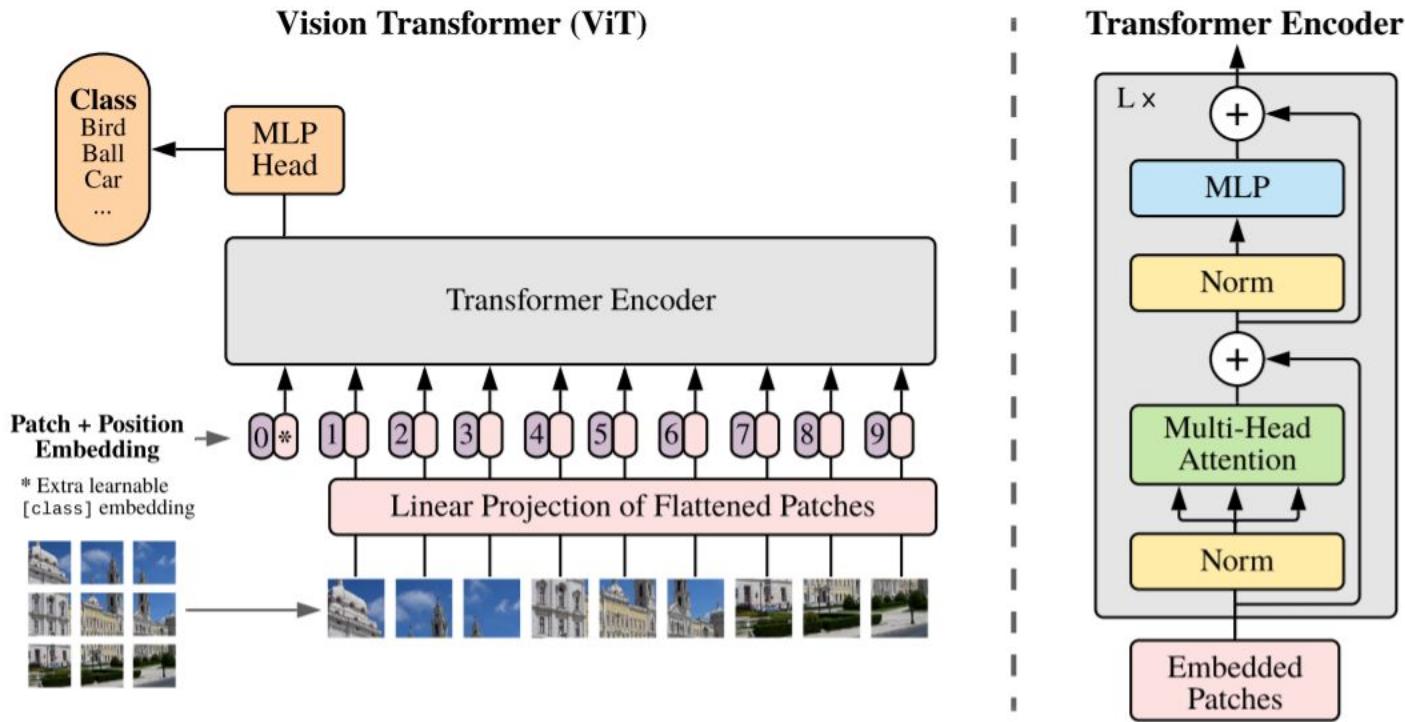
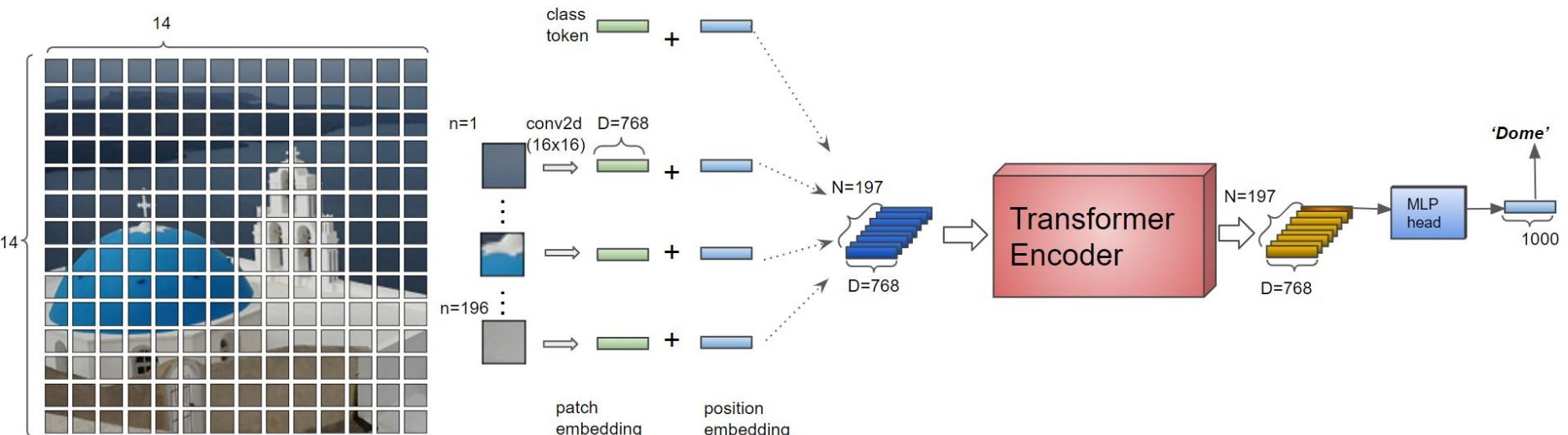
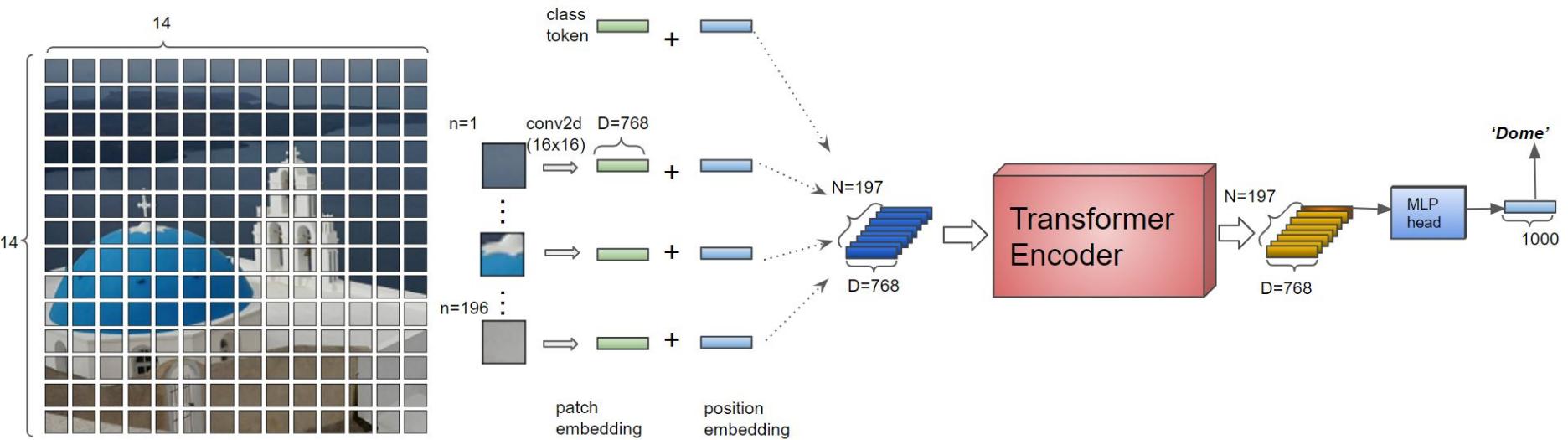


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable “classification token” to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).



# 1. Split the image into patches and encode each patch

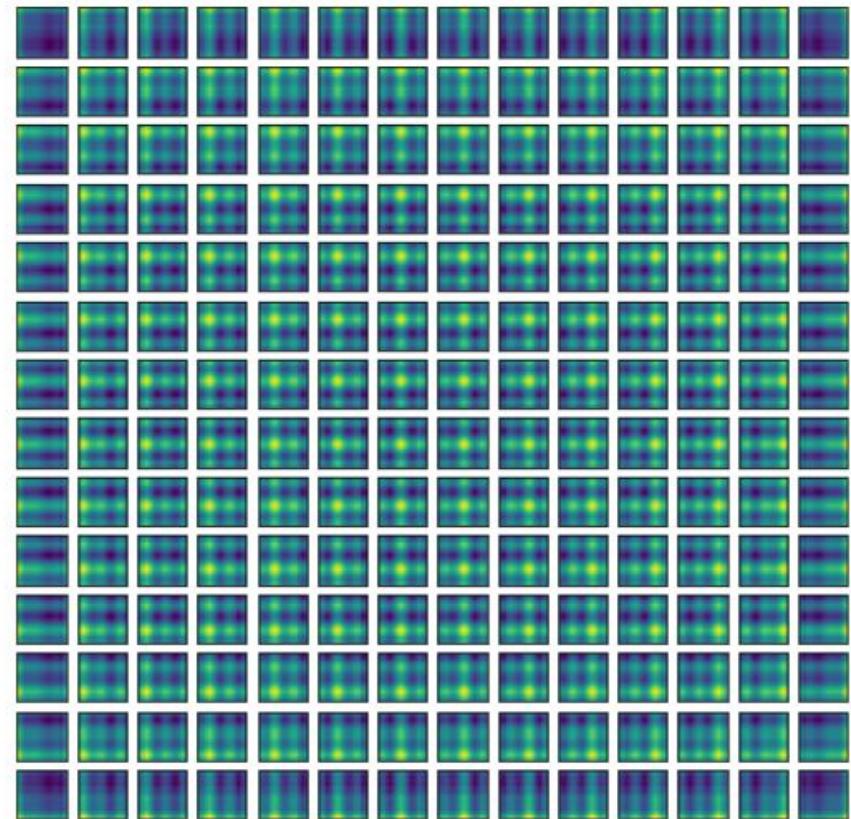
```
x = nn.Conv(
    features=self.hidden_size,
    kernel_size=self.patches.size,
    strides=self.patches.size,
    padding='VALID',
    name='embedding')(
    x)
```



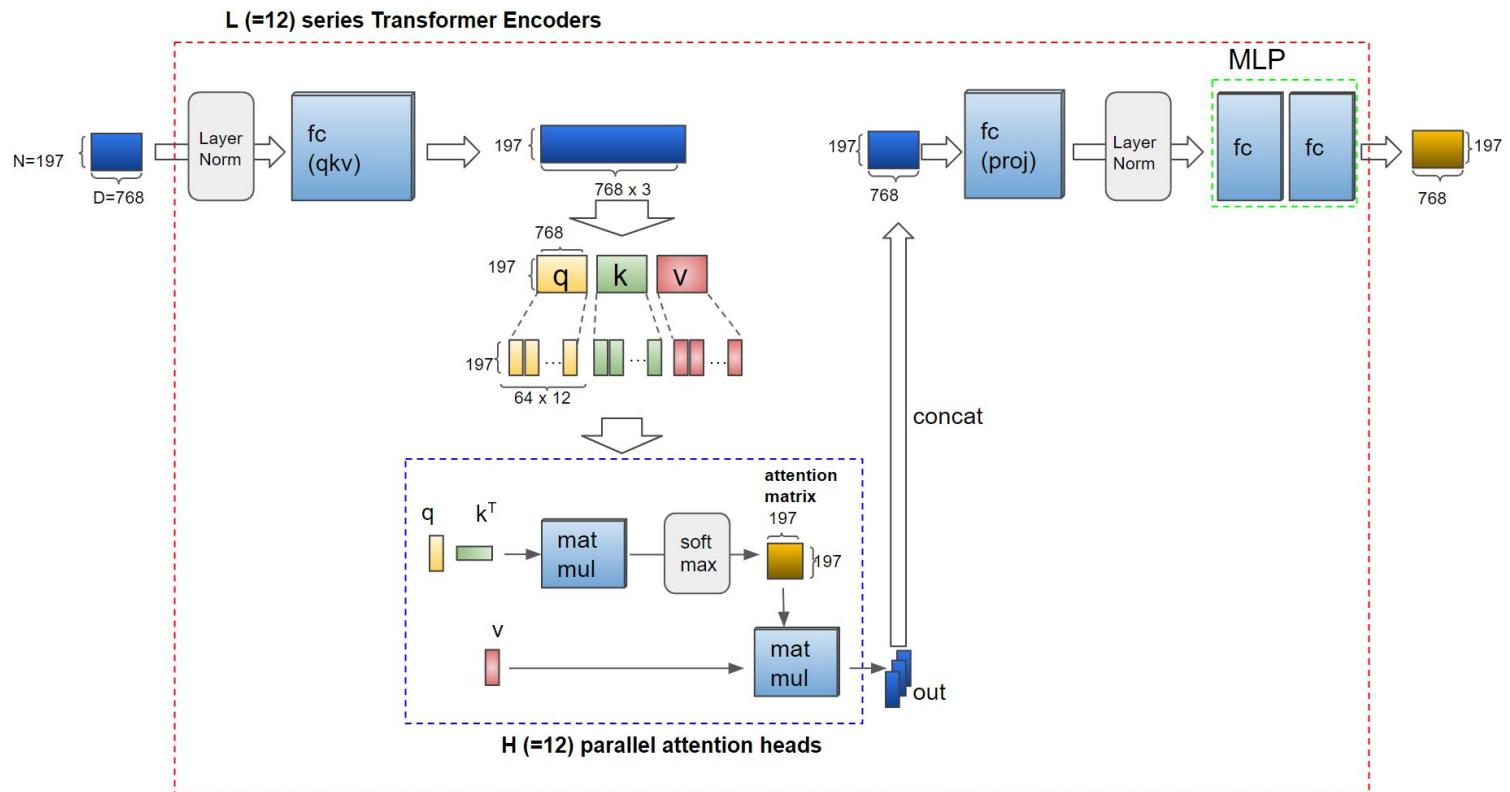
1. Split the image into patches and encode each patch
2. Add Position Embeddings

# Position Embeddings Visualisation

```
# Visualize position embedding similarities.  
# One cell shows cos similarity between an embedding and all the other embeddin  
cos = torch.nn.CosineSimilarity(dim=1, eps=1e-6)  
fig = plt.figure(figsize=(8, 8))  
fig.suptitle("Visualization of position embedding similarities", fontsize=24)  
for i in range(1, pos_embed.shape[1]):  
    sim = F.cosine_similarity(pos_embed[0, i:i+1], pos_embed[0, 1:], dim=1)  
    sim = sim.reshape((14, 14)).detach().cpu().numpy()  
    ax = fig.add_subplot(14, 14, i)  
    ax.axes.get_xaxis().set_visible(False)  
    ax.axes.get_yaxis().set_visible(False)  
    ax.imshow(sim)
```



1. Split the image into patches and encode each patch
2. Add Position Embeddings
3. Transformer Encoder



# VISION TRANSFORMER

1. Split the image into patches and encode each patch
2. Add Position Embeddings
3. Transformer Encoder
4. 4. MLP (Classification) Head

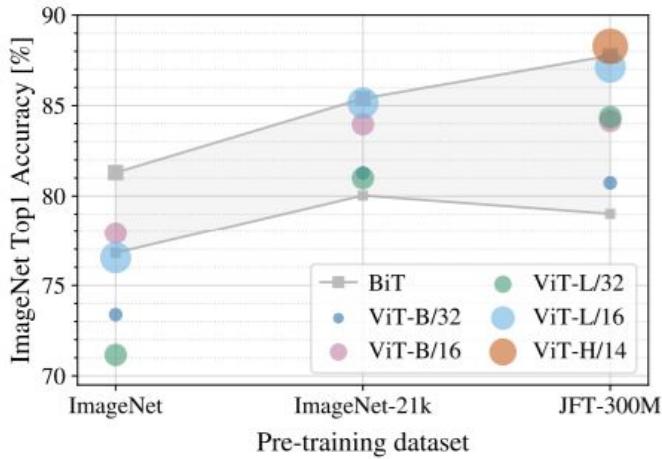


Figure 3: Transfer to ImageNet. While large ViT models perform worse than BiT ResNets (shaded area) when pre-trained on small datasets, they shine when pre-trained on larger datasets. Similarly, larger ViT variants overtake smaller ones as the dataset grows.

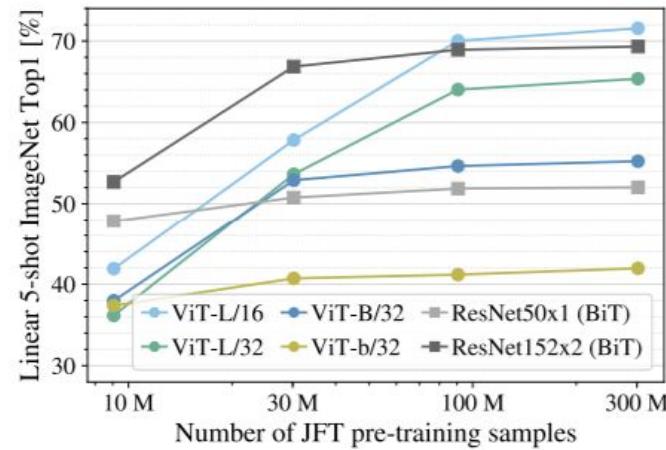


Figure 4: Linear few-shot evaluation on ImageNet versus pre-training size. ResNets perform better with smaller pre-training datasets but plateau sooner than ViT, which performs better with larger pre-training. ViT-b is ViT-B with all hidden dimensions halved.

# Computer Vision and Deep Learning

Lecture 9

# Computer vision tasks

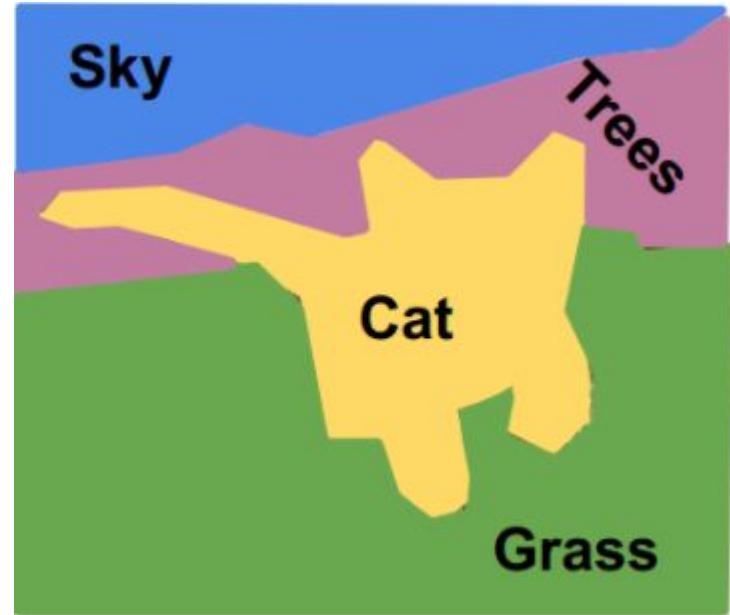
# Computer vision tasks



## Classification

What object is in this image?

CAT



## Semantic segmentation

What label has each pixel?

Pixel level, we are not interested in objects

# Computer vision tasks

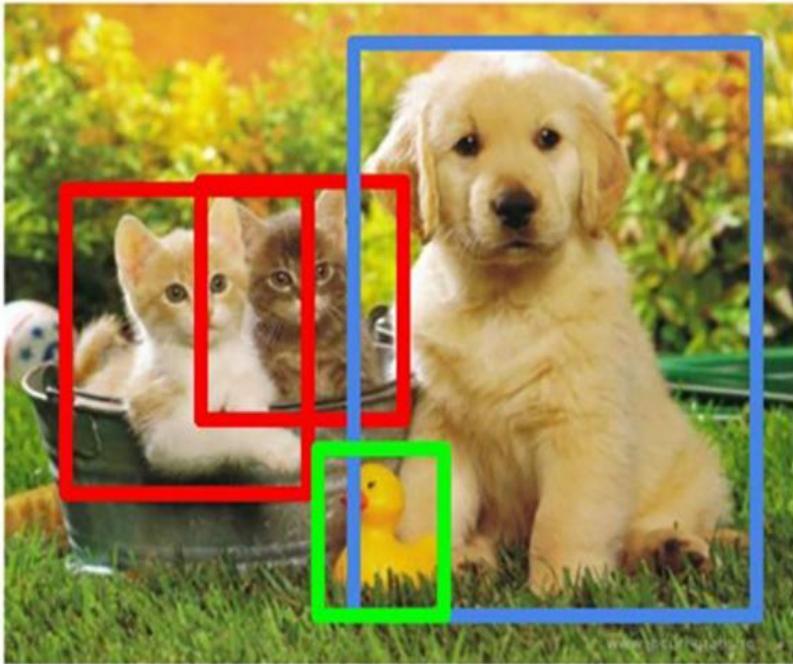


## Object localization

What object is in this image and  
where is it located?

CAT

# Computer vision tasks



**CAT, DOG, DUCK**

**Object detection:** multiple objects  
(the label and position - bounding box  
- of each object)

Image source: <https://static.artfido.com/2018/05/cover-12.jpg>



**CAT, DOG, DUCK**

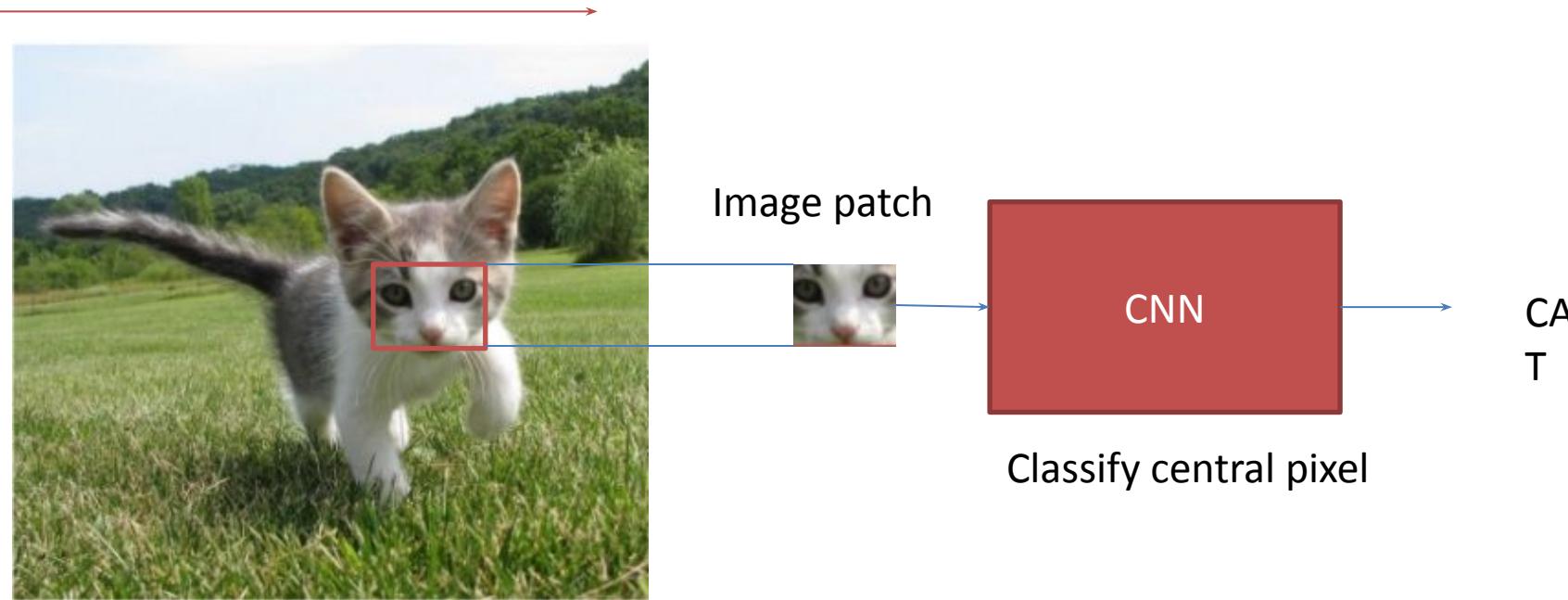
**Instance segmentation:** multiple objects  
(the label and position of each object)

# Semantic segmentation

- Understanding the image at **pixel level**: each pixel is labelled individually with a class
- Groups both semantics and location
  - Global information: WHAT?
  - Local information: WHERE?

# Semantic segmentation

Naïve approach – sliding window



Slide window over the input image and classify each image patch using a CNN

# Semantic segmentation

Naïve approach – sliding window

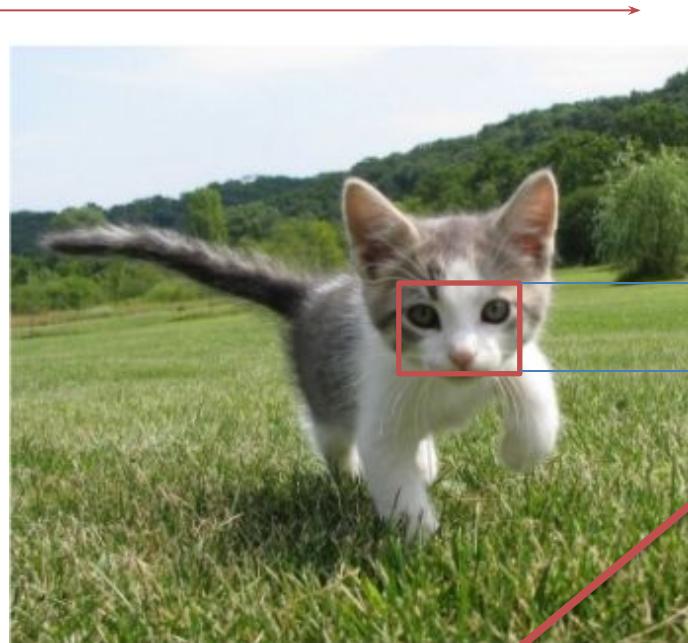
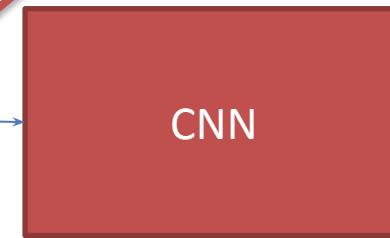


Image patch



CNN

Classify central pixel

CA  
T

Slide window over the input image and classify each image patch using a CNN

Highly inefficient!!

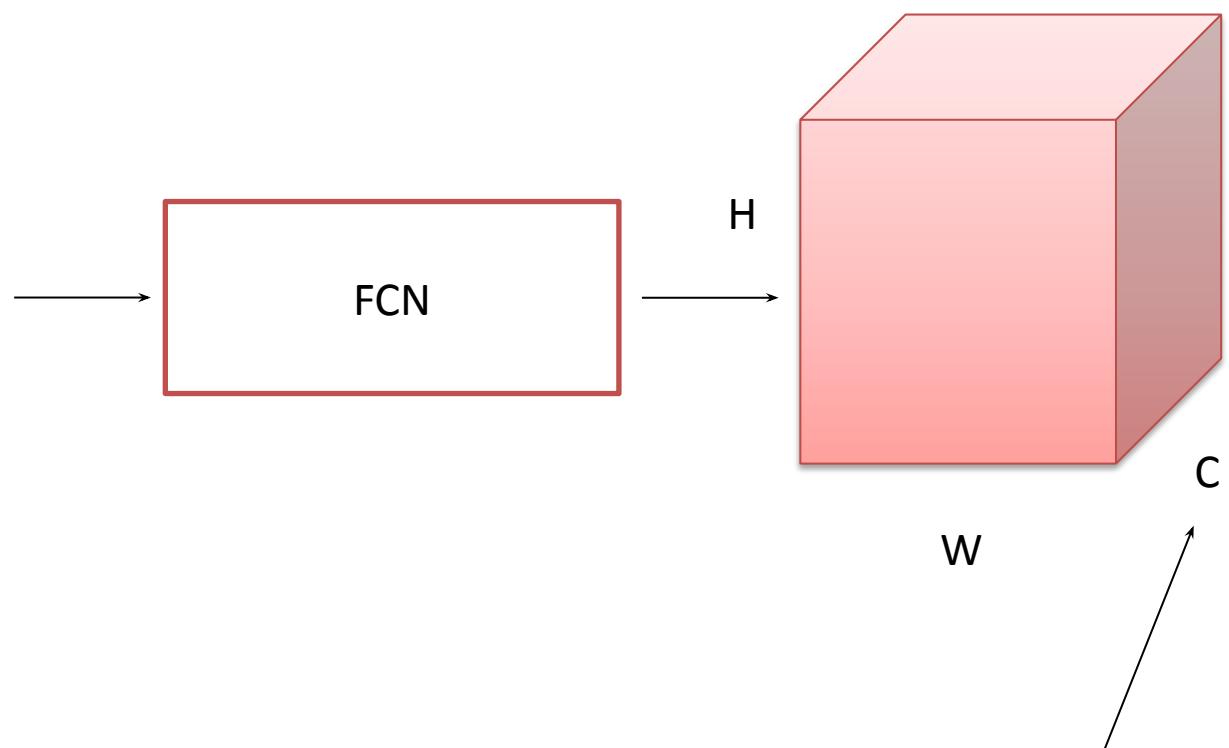
# Let's design a CNN to classify all pixels at once!

- How would we encode the output (we need to output a class label for each pixel)?
- What would be the input and the output size of this network?

Fully convolutional neural networks

# Fully convolutional neural networks

The output should have the same spatial size as the input



Output depth:  
Number of classes



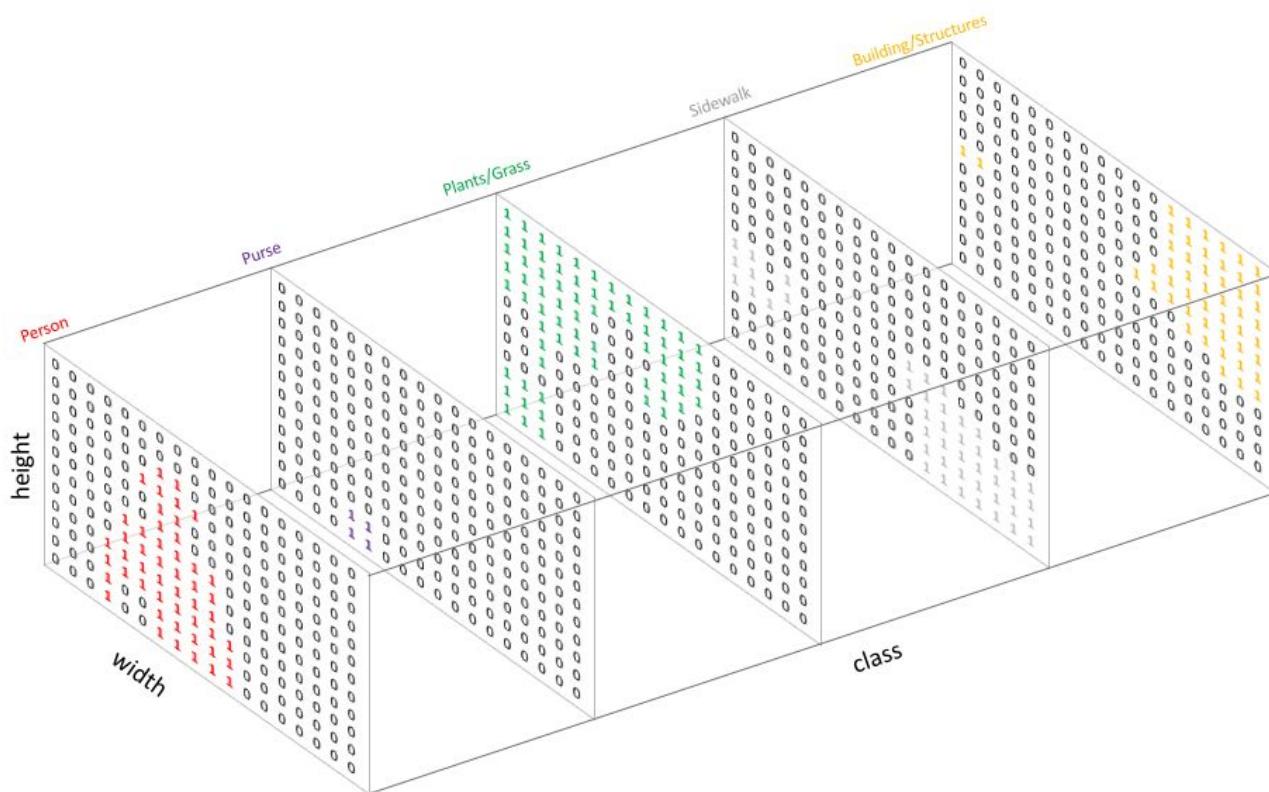
Input

segmented →

- 1: Person
- 2: Purse
- 3: Plants/Grass
- 4: Sidewalk
- 5: Building/Structures

3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	5	5	5	5	5	5
5	5	3	3	3	3	3	3	3	1	1	1	1	1	1	3	3	5	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	1	1	1	1	1	4	4	4	5	5	5	5	5
4	4	3	4	1	1	1	1	1	1	1	1	1	1	1	4	4	4	4	4	5	5	5
4	4	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4
3	3	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4
3	3	3	1	2	2	1	1	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4

Semantic Labels



# Output layer

Apply cross entropy  
pixel-wise on the last  
layer of the network

2	1
1	0

Ground truth label image

3 classes (0, 1, 2)

One hot encodings

2 – [0, 0, 1]

1 – [0, 1, 0]

0 – [1, 0, 0]

0	0
0	1

Class 0

0	1
1	0

Class 1

1	0
0	0

Class 2

# Fully convolutional neural networks (FCN)

- How to preserve the spatial size of the output map (input size must be equal to the output size)?

# Fully convolutional neural networks

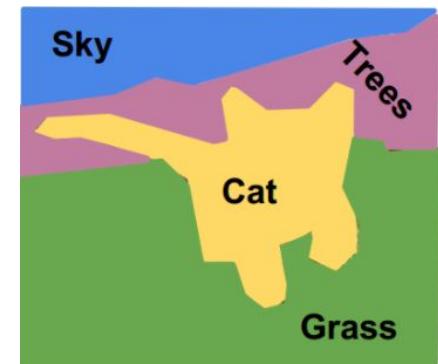
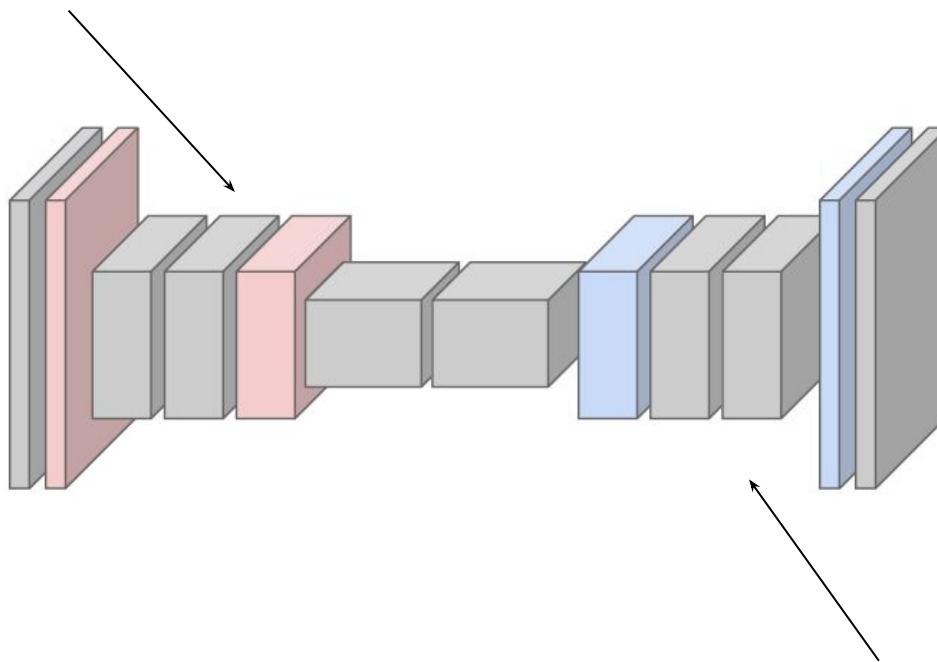
- How to preserve the spatial size of the output map (input size must be equal to the output size)?
  - Don't use any layers that reduce the spatial size

# Fully convolutional neural networks

- How to preserve the spatial size of the output map (input size must be equal to the output size)?
  - Don't use any layers that reduce the spatial size -> **inefficient**
  - Use two paths in the network: a down-sampling and an up-sampling path

# Fully convolutional neural networks

Down-sampling: strided  
convolutions, pooling layers



Up-sampling: NN, bilinear  
interpolation, max unpooling,  
transposed convolution

# Up-sampling techniques

# Up-sampling

Nearest neighbour

4	5
10	20

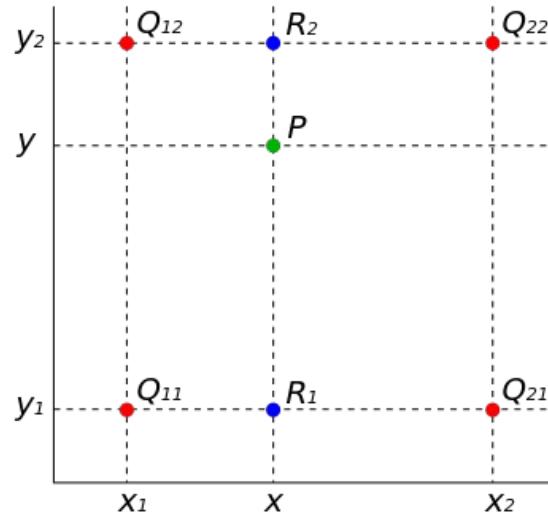
2x2

4	4	5	5
4	4	5	5
10	10	20	20
10	10	20	20

4x4

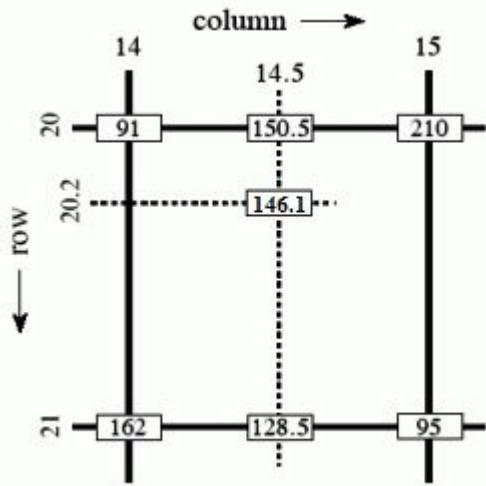
# Up-sampling

## Bilinear interpolation



$$f(x, y_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}),$$

$$f(x, y_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}).$$



$$I_{20,14.5} = \frac{15 - 14.5}{15 - 14} \cdot 91 + \frac{14.5 - 14}{15 - 14} \cdot 210 = 150.5,$$

$$I_{21,14.5} = \frac{15 - 14.5}{15 - 14} \cdot 162 + \frac{14.5 - 14}{15 - 14} \cdot 95 = 128.5,$$

$$I_{20.2,14.5} = \frac{21 - 20.2}{21 - 20} \cdot 150.5 + \frac{20.2 - 20}{21 - 20} \cdot 128.5 = 146.1.$$

# Up-sampling

Bilinear interpolation

10	20
30	40

2x2

10	12.5	17.5	20
15	17.5	22.5	25
25	27.5	32.5	35
30	32.5	37.5	40

4x4

# Up-sampling in torch

## UPSAMPLE ↴

```
CLASS torch.nn.Upsample(size=None, scale_factor=None, mode='nearest', align_corners=None,
recompute_scale_factor=None) [SOURCE]
```

Upsamples a given multi-channel 1D (temporal), 2D (spatial) or 3D (volumetric) data.

The input data is assumed to be of the form *minibatch x channels x [optional depth] x [optional height] x width*. Hence, for spatial inputs, we expect a 4D Tensor and for volumetric inputs, we expect a 5D Tensor.

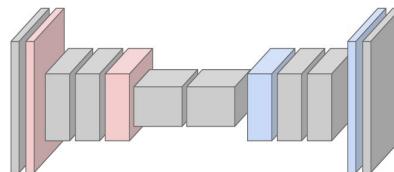
The algorithms available for upsampling are nearest neighbor and linear, bilinear, bicubic and trilinear for 3D, 4D and 5D input Tensor, respectively.

One can either give a `scale_factor` or the target output `size` to calculate the output size. (You cannot give both, as it is ambiguous)

## Interpolation:

- nearest
- bilinear
- bicubic
- trilinear

# Max un-pooling



Symmetric structure of the FCN

1	4	3	1
2	3	5	2
10	9	2	3
7	3	4	20

4	5
10	20

0	4	0	0
0	0	5	0
10	0	0	0
0	0	0	20

**Pooling:** remember the position of the maximum element within the receptive field of the layer

DOWNSAMPLING PATH

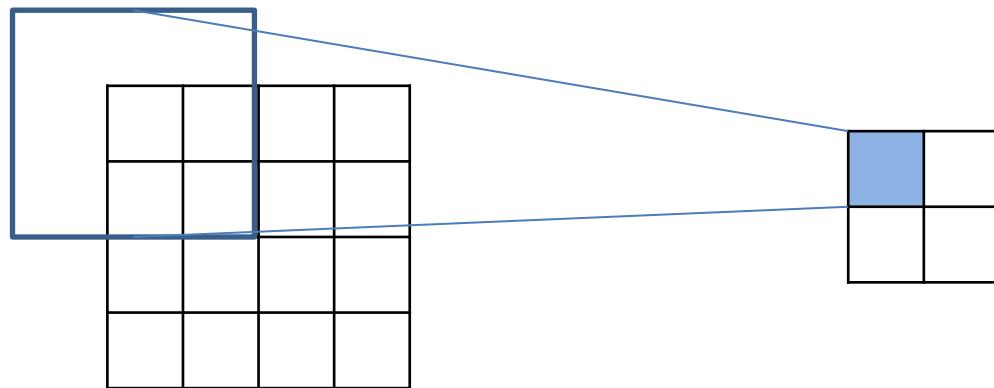
**Max Un-Pooling:** use the position of the max

UP-SAMPLING  
PATH

# Transposed convolutions

## Learnable up-sampling

Remember *strided* convolutions

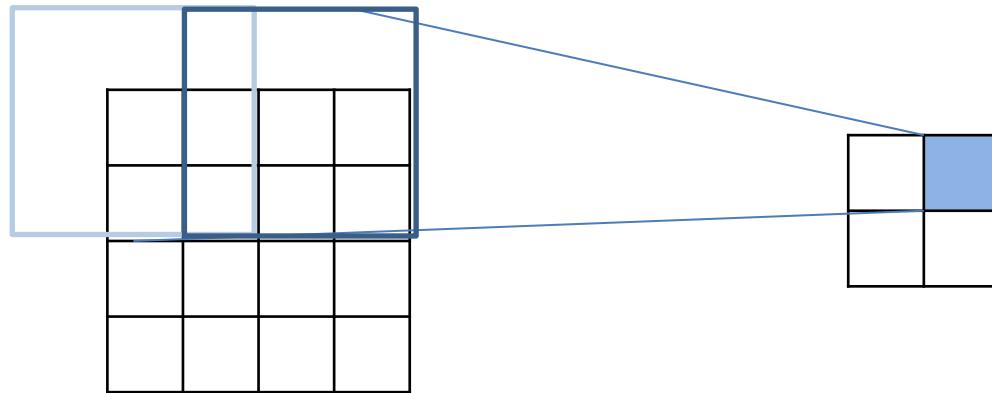


3x3, stride 2, pad 1

# Transposed convolutions

## Learnable up-sampling

Remember *strided* convolutions

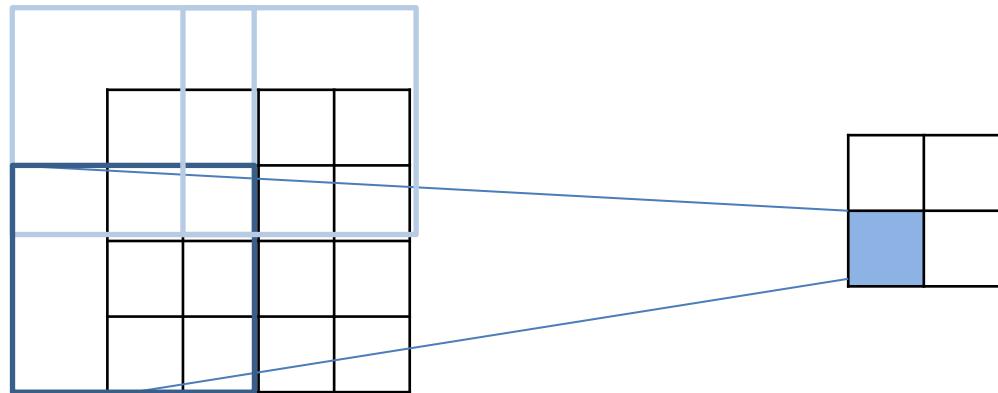


3x3, stride 2, pad 1

# Transposed convolutions

## Learnable up-sampling

Remember *strided* convolutions

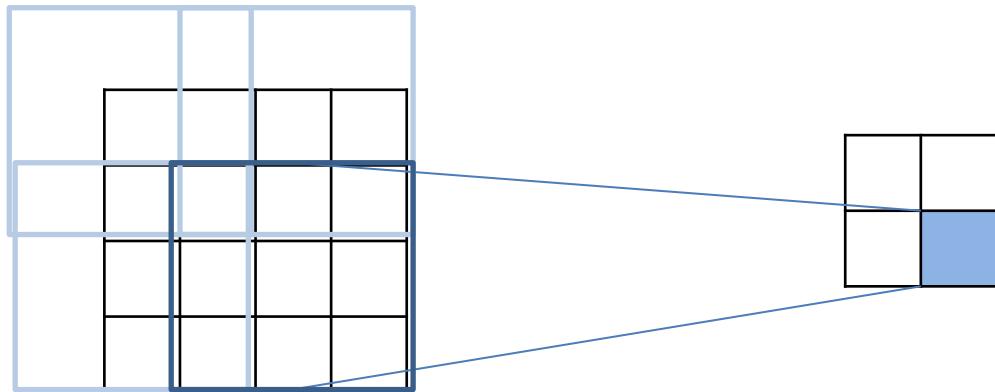


3x3, stride 2, pad 1

# Transposed convolutions

## Learnable up-sampling

Remember *strided* convolutions

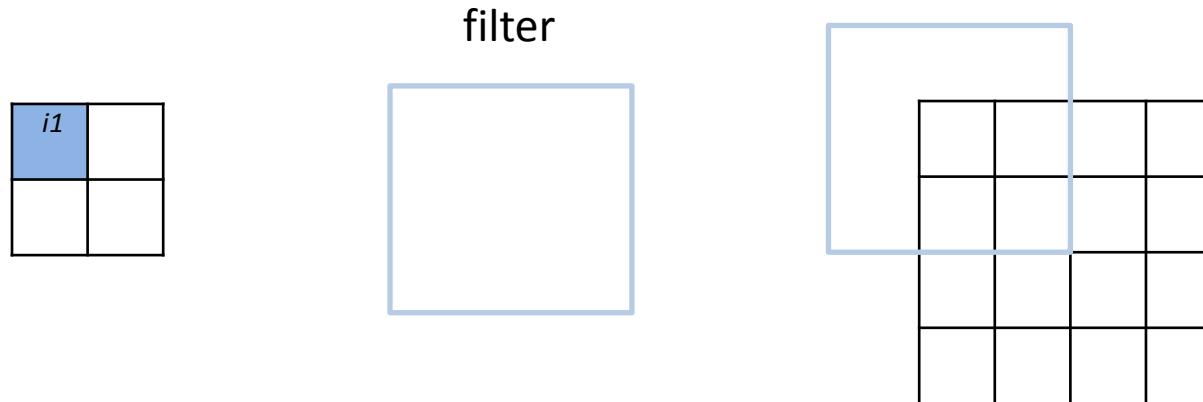


3x3, stride 2, pad 1

# Transposed convolutions

## Learnable up-sampling

NOW: transposed convolutions



The input feature  $i_1$  is multiplied with the filter values  
 $i_1$  is like a weight for the filter value.

Copy the “weighted” filter in the corresponding output

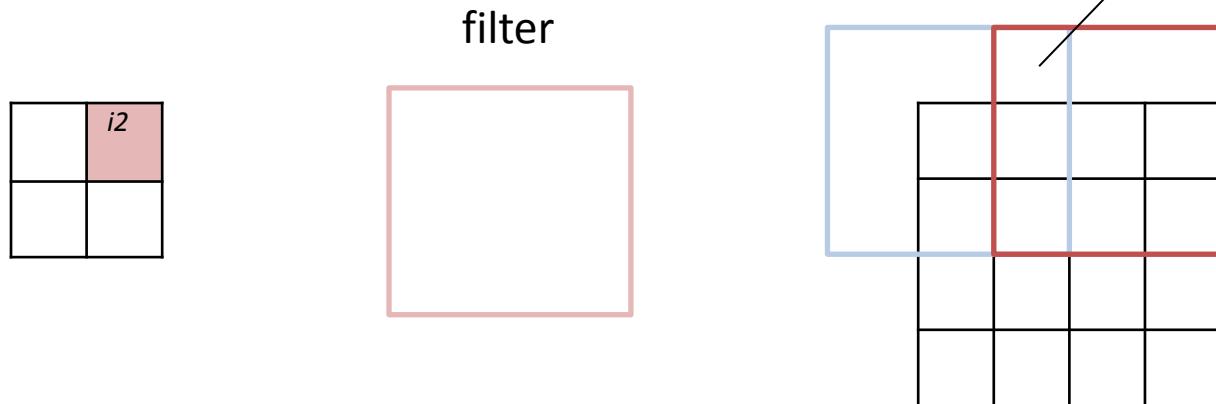
3x3, stride 2, pad 1

# Transposed convolutions

## Learnable up-sampling

NOW: transposed convolutions

Where there is  
overlap, sum up  
the values



The input feature  $i2$  is multiplied with the filter values  
 $i2$  is like a weight for the filter value.

Copy the “weighted” filter in the corresponding output

3x3, stride 2, pad 1

# Transposed convolutions

## Learnable up-sampling

The weights of the filter are learned

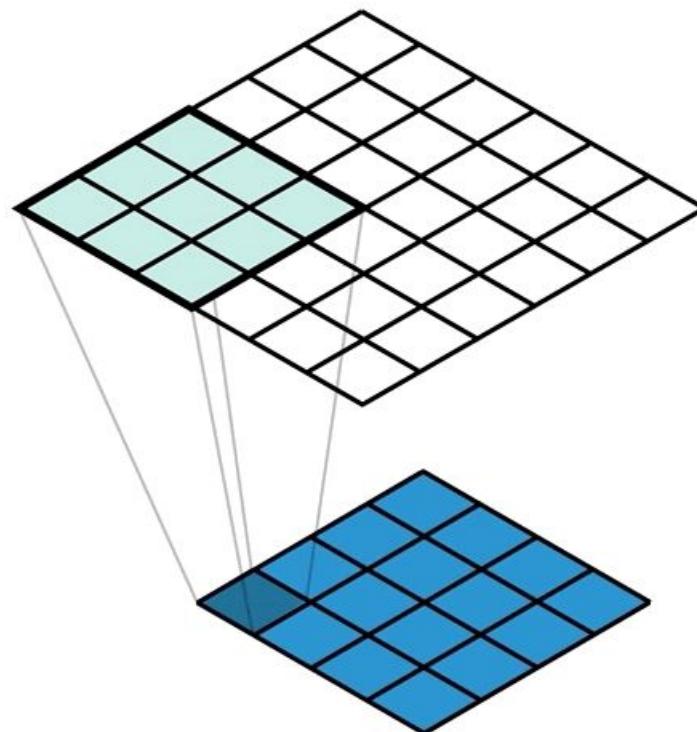
At each position multiply the filter values with the corresponding position of the input layer to get the result of the output  
Add overlapping positions

Input	Kernel	=			Output																																																										
<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>1</td></tr><tr><td>2</td><td>3</td></tr></table>	0	1	2	3	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>1</td></tr><tr><td>2</td><td>3</td></tr></table>	0	1	2	3	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>0</td><td></td></tr><tr><td>0</td><td>0</td><td></td></tr><tr><td></td><td></td><td></td></tr></table>	0	0		0	0					$+ \quad$	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td></td><td>0</td><td>1</td></tr><tr><td></td><td>2</td><td>3</td></tr><tr><td></td><td></td><td></td></tr></table>		0	1		2	3				$+ \quad$	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>2</td><td></td></tr><tr><td>4</td><td>6</td><td></td></tr><tr><td></td><td></td><td></td></tr></table>	0	2		4	6					$+ \quad$	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td></td><td></td><td></td></tr><tr><td></td><td>0</td><td>3</td></tr><tr><td></td><td>6</td><td>9</td></tr></table>					0	3		6	9	$= \quad$	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>4</td><td>6</td></tr><tr><td>4</td><td>12</td><td>9</td></tr></table>	0	0	1	0	4	6	4	12	9
0	1																																																														
2	3																																																														
0	1																																																														
2	3																																																														
0	0																																																														
0	0																																																														
	0	1																																																													
	2	3																																																													
0	2																																																														
4	6																																																														
	0	3																																																													
	6	9																																																													
0	0	1																																																													
0	4	6																																																													
4	12	9																																																													

Example: 1 stride and 0 padding

# Transposed convolutions

**Learnable** up-sampling



# Transposed convolutions in torch

## CONVTRANSPOSE2D

---

```
CLASS torch.nn.ConvTranspose2d(in_channels, out_channels, kernel_size, stride=1, padding=0,  
    output_padding=0, groups=1, bias=True, dilation=1, padding_mode='zeros', device=None,  
    dtype=None) [SOURCE]
```

Applies a 2D transposed convolution operator over an input image composed of several input planes.

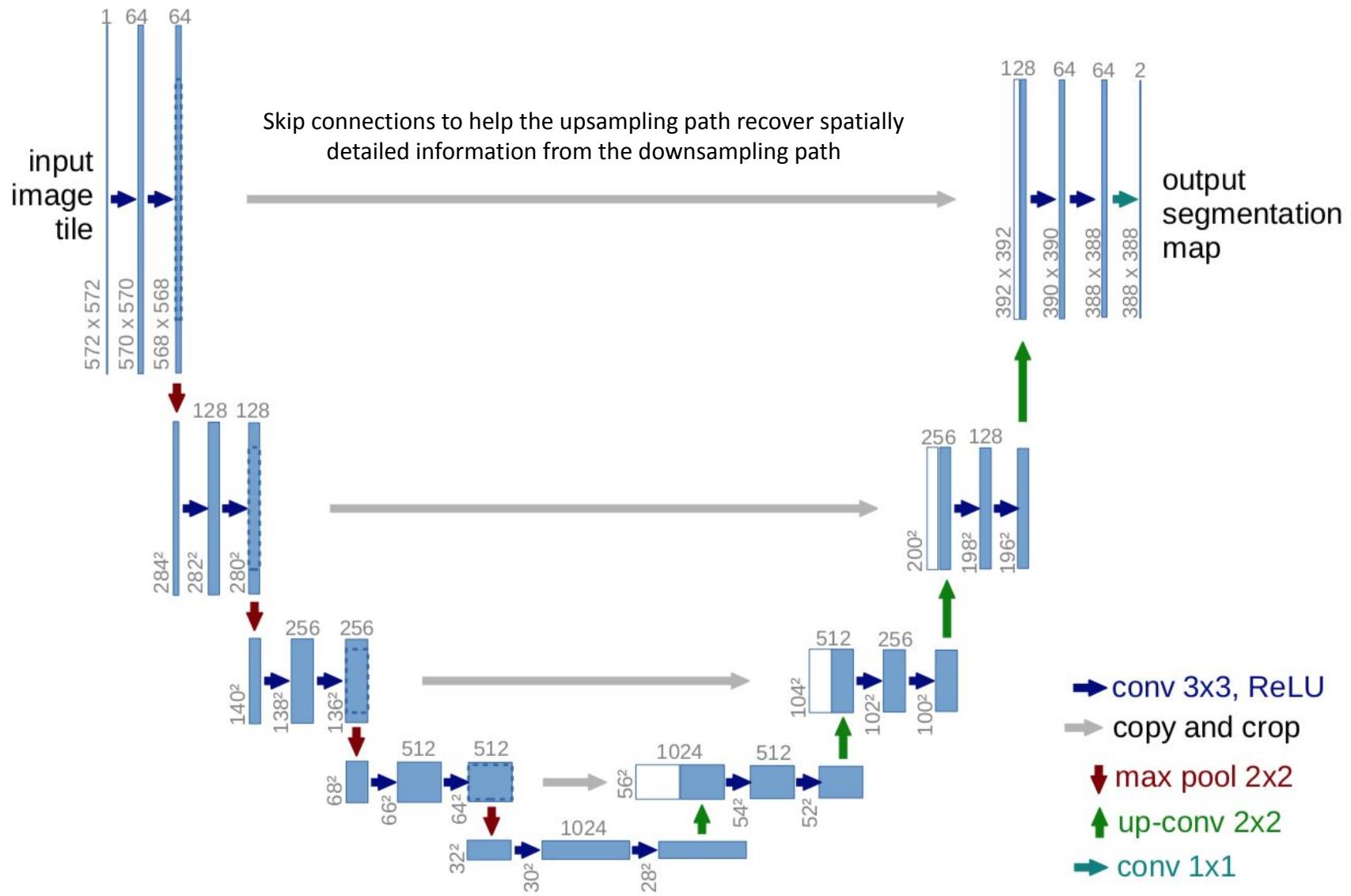
This module can be seen as the gradient of Conv2d with respect to its input. It is also known as a fractionally-strided convolution or a deconvolution (although it is not an actual deconvolution operation as it does not compute a true inverse of convolution). For more information, see the visualizations [here](#) and the [Deconvolutional Networks](#) paper.

This module supports [TensorFloat32](#).

On certain ROCm devices, when using float16 inputs this module will use [different precision](#) for backward.

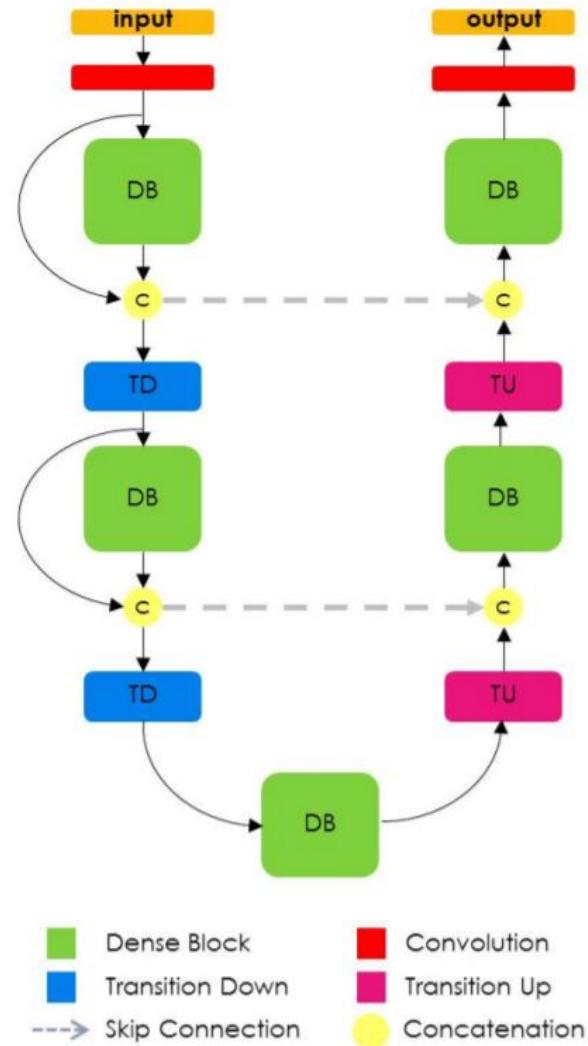
- `stride` controls the stride for the cross-correlation.
- `padding` controls the amount of implicit zero padding on both sides for  $\text{dilation} * (\text{kernel\_size} - 1) + \text{padding}$  number of points. See note below for details.
- `output_padding` controls the additional size added to one side of the output shape. See note below for details.
- `dilation` controls the spacing between the kernel points; also known as the à trous algorithm. It is harder to describe, but the link [here](#) has a nice visualization of what `dilation` does.

# U-Net, 2015



# 100 layers tiramisu, 2017

- Similar to U-Net
- uses Dense Block for convolutions and transposed convolutions

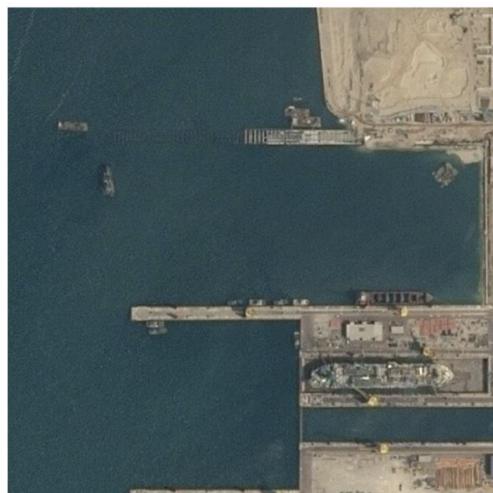


# Semantic segmentation metrics

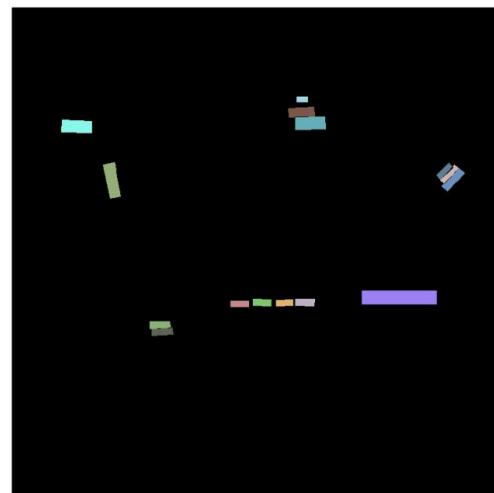
## Pixel accuracy

- Number of pixels classified correctly by the network

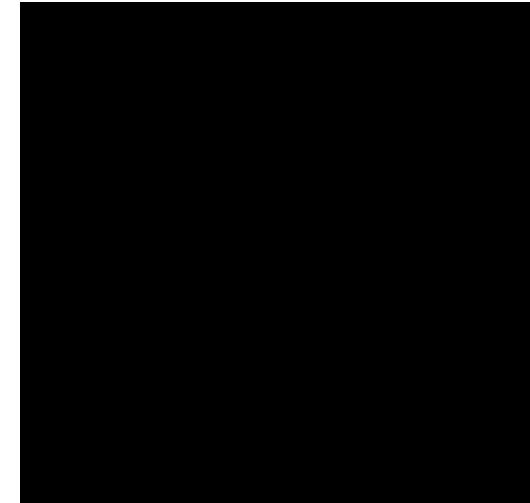
$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$



Image



Ground truth



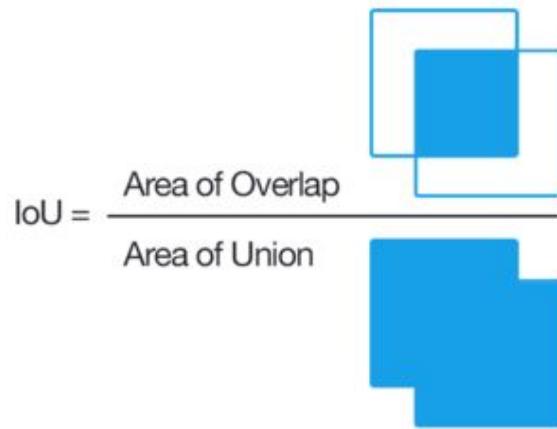
Prediction

Accuracy 95%



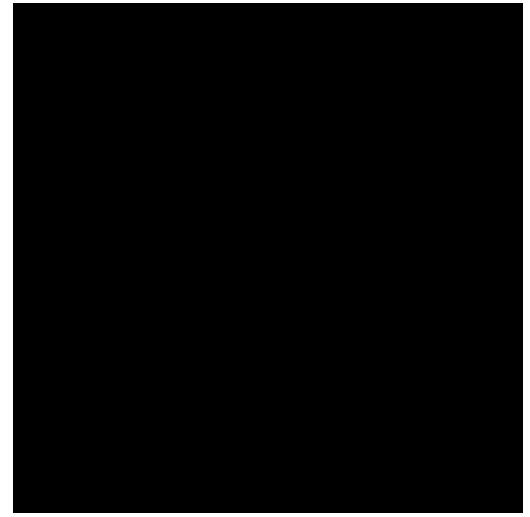
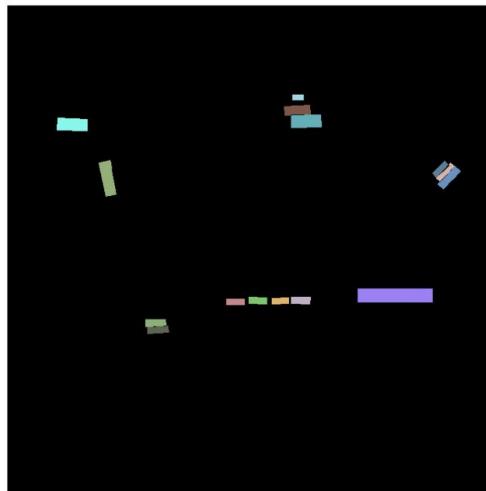
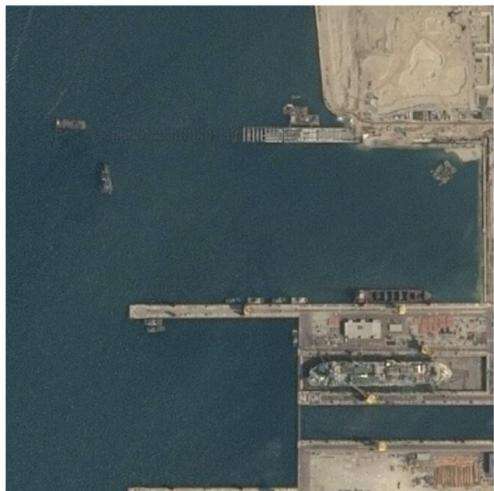
# Semantic segmentation metrics

## Intersection over Union



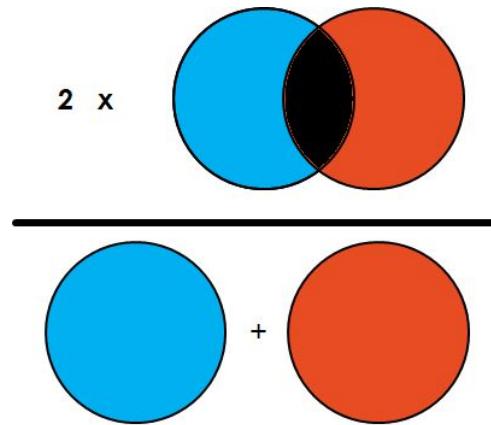
Mean IOU 47.5%

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

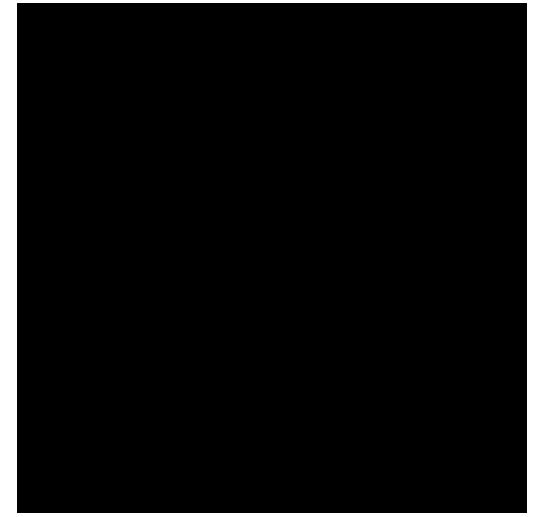
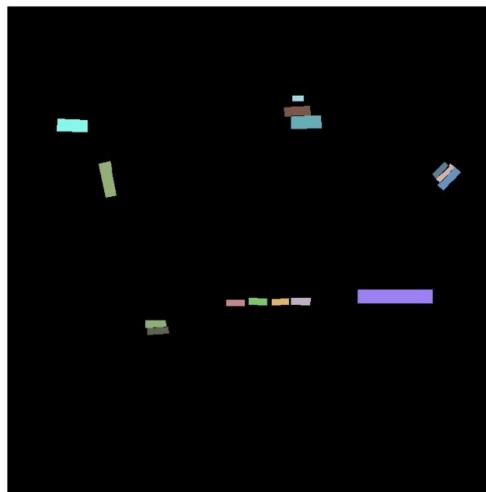
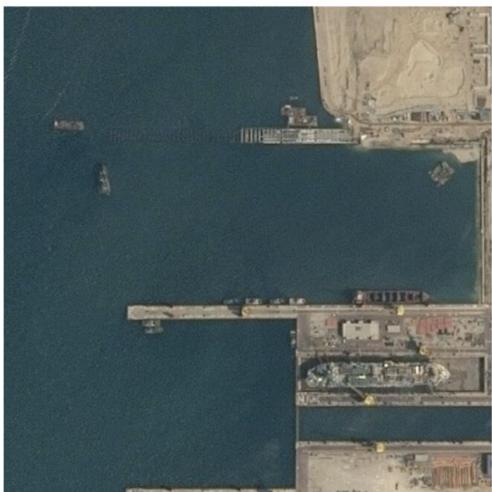


# Semantic segmentation metrics

## Dice Coefficient

$$\frac{2 \times \text{Intersection Area}}{\text{Union Area}}$$


Dice score 47.5%



# Semantic segmentation

## Examples

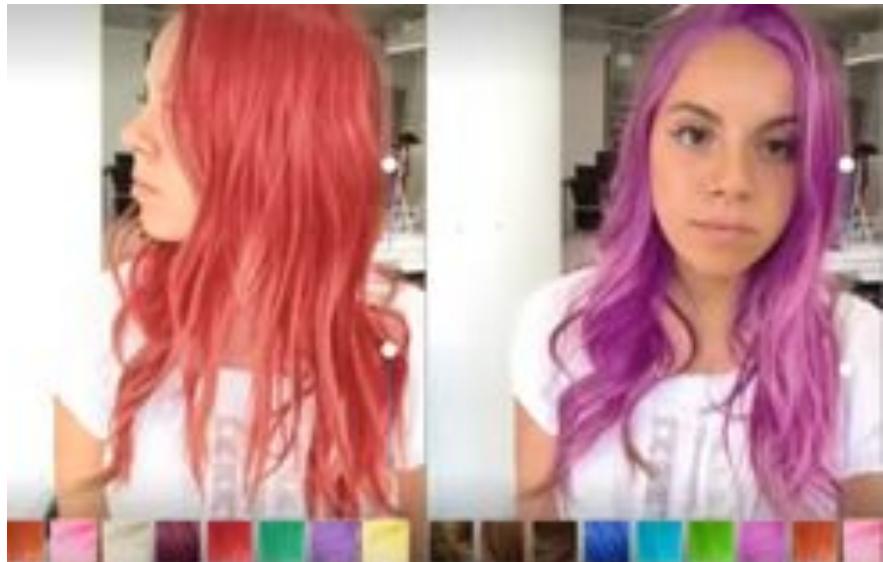
Autonomous driving



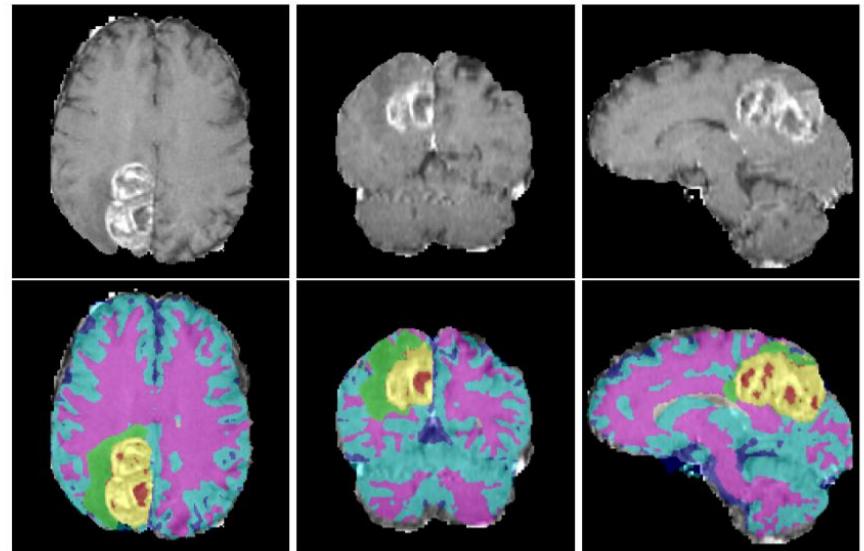
Sky	Building	Pole	Road	Sidewalk	Vegetation	Sign	Fence	Car	Pedestrian	Cyclist
-----	----------	------	------	----------	------------	------	-------	-----	------------	---------

# Semantic segmentation

## Examples



Real time hair colouring



Medical image segmentation  
<https://arxiv.org/pdf/1810.05732.pdf>

<https://news.developer.nvidia.com/3d-real-time-video-hair-coloration/>

# Computer Vision and Deep Learning

Lecture 10

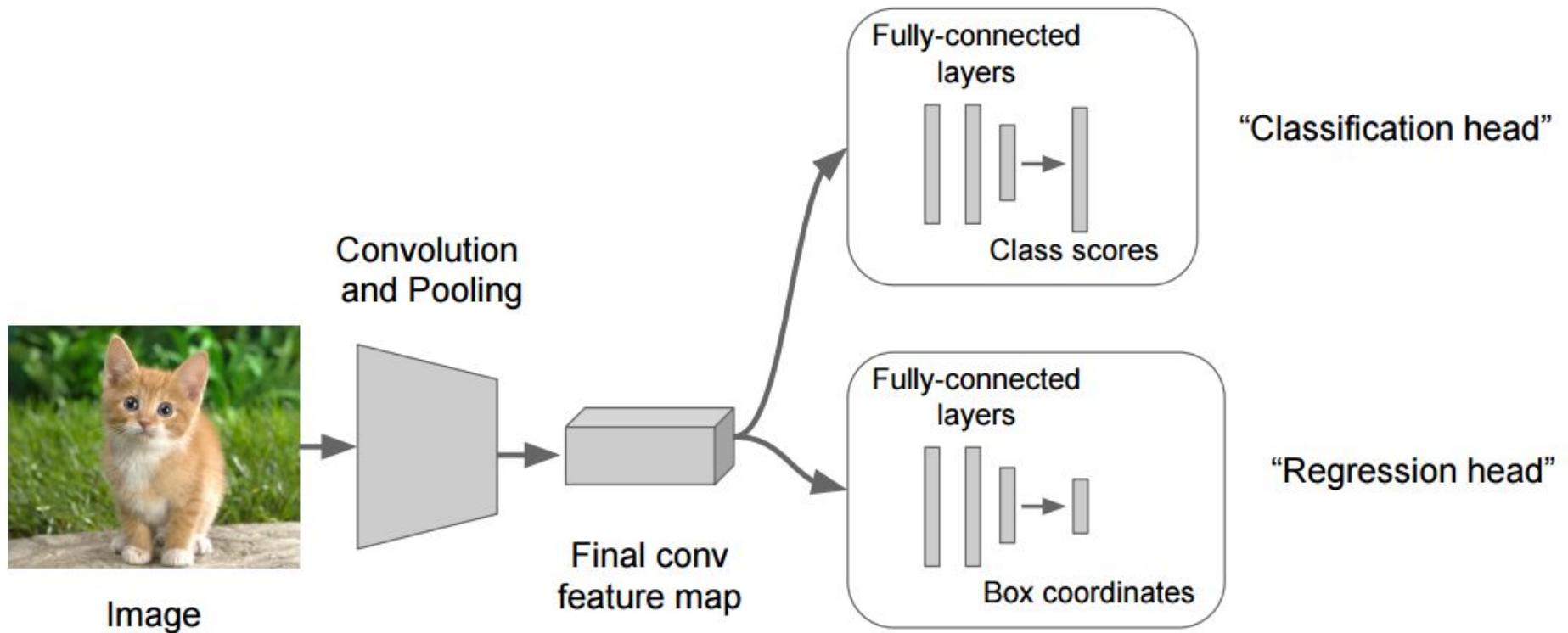
# Object localization

- What object is this image and where is this object located in the image?



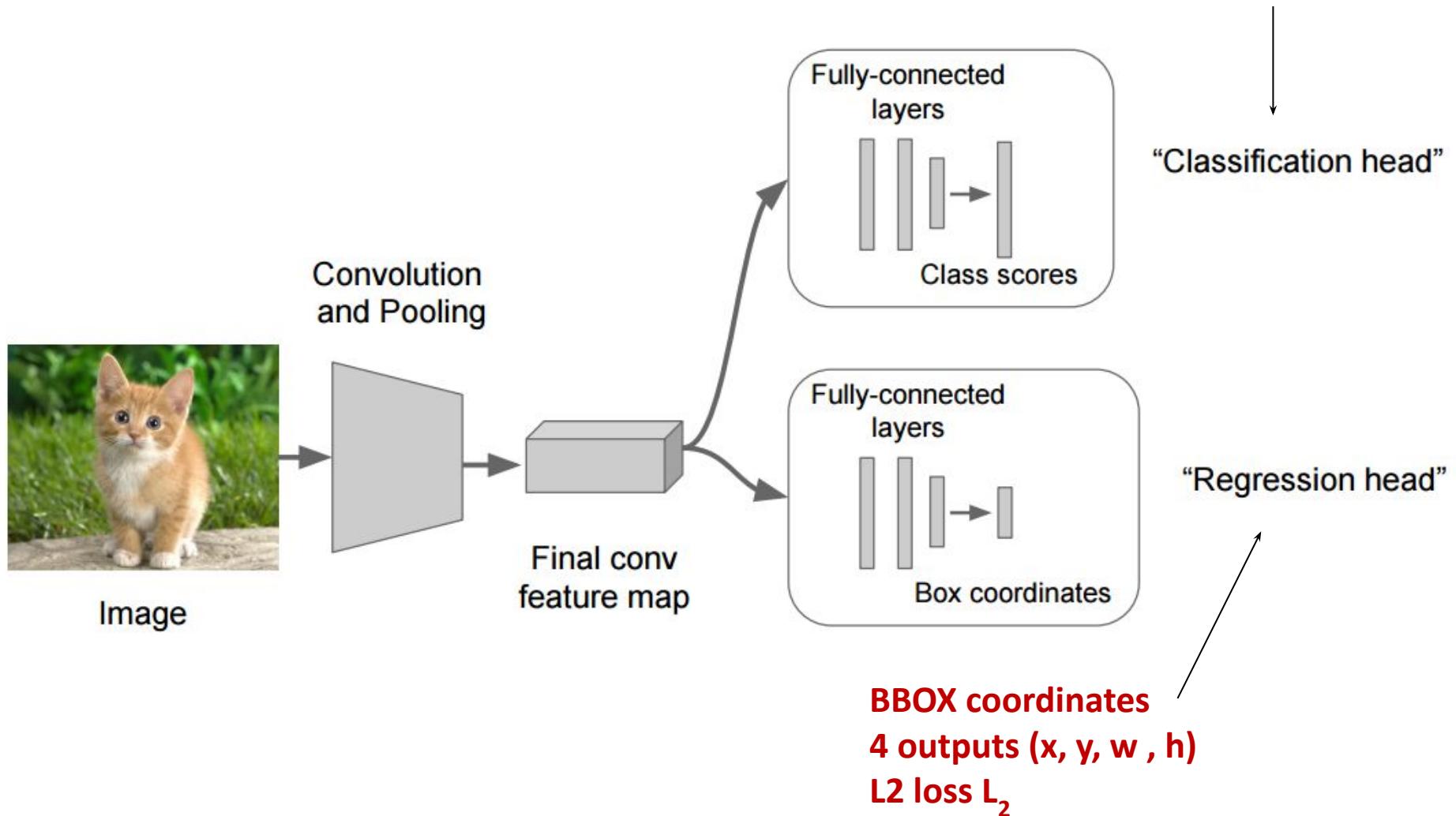
CAT

# Object localization



# Object localization

C class scores  
Softmax loss  $L_s$



# Object detection

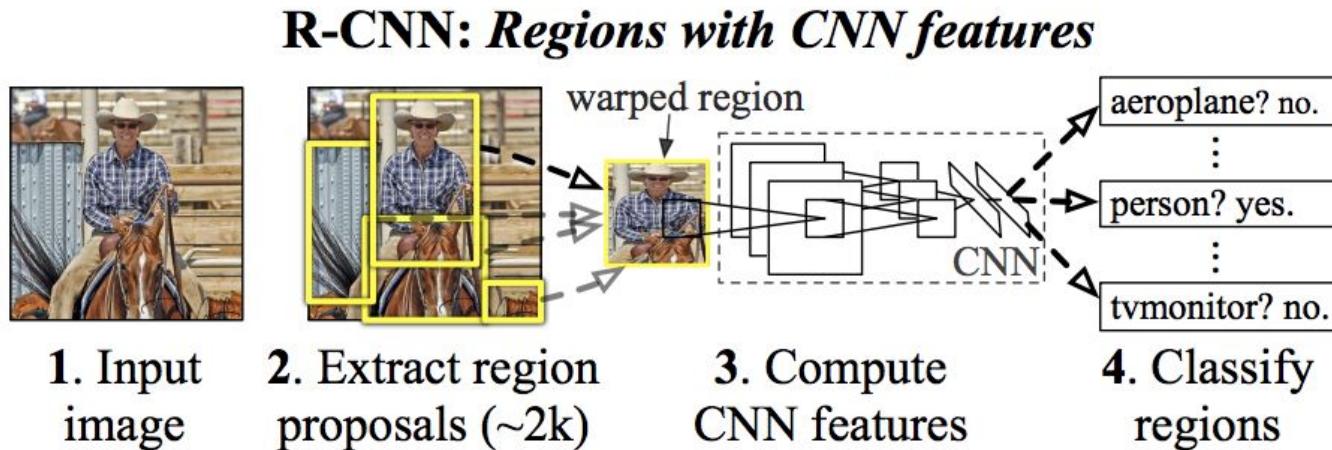
- Determine the **class (label)** and the **position** of EACH object in the input image
- We can't use the same approach as for localization
  - Each image would require a different number of outputs
- Sliding window approach?

# Proposal based object detection

# R-CNN

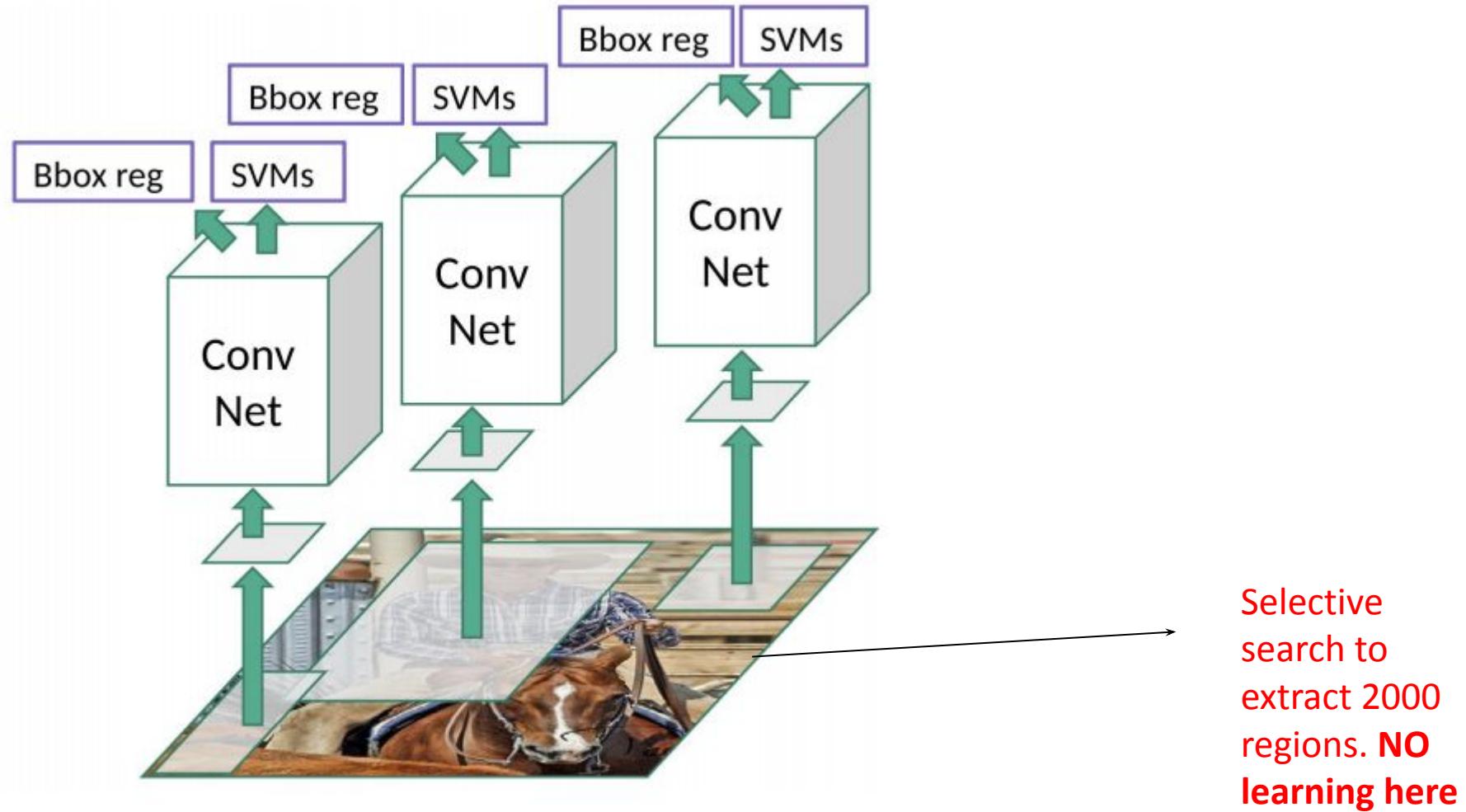
Region-based Convolutional Network, 2014

- Idea:
  - Use an algorithm (or network) to find region of interests (ROIs) that are likely to contain an object
  - **Localize** (label + bounding box) localize the object in each in region



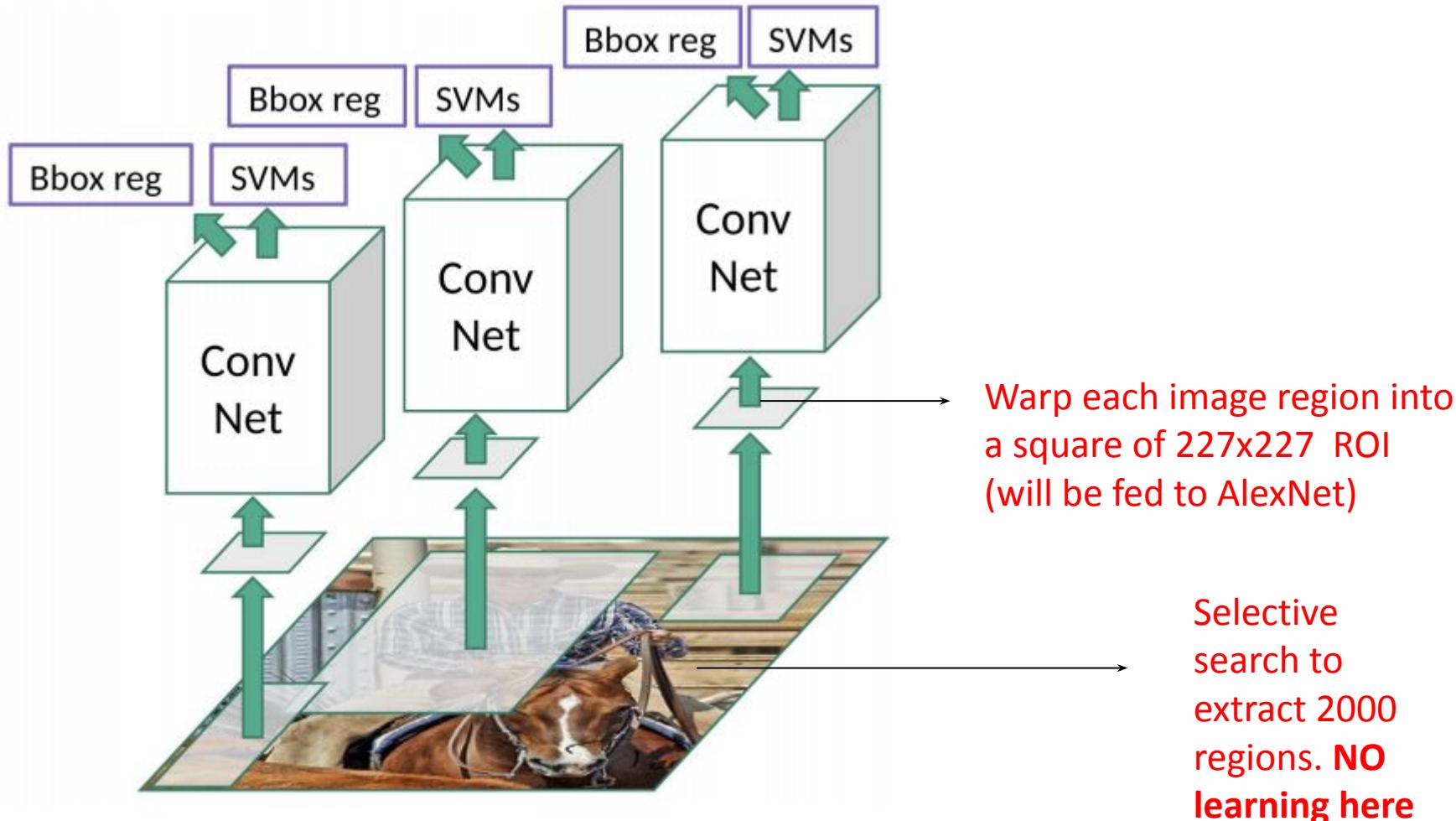
# R-CNN

## Region-based Convolutional Network, 2014



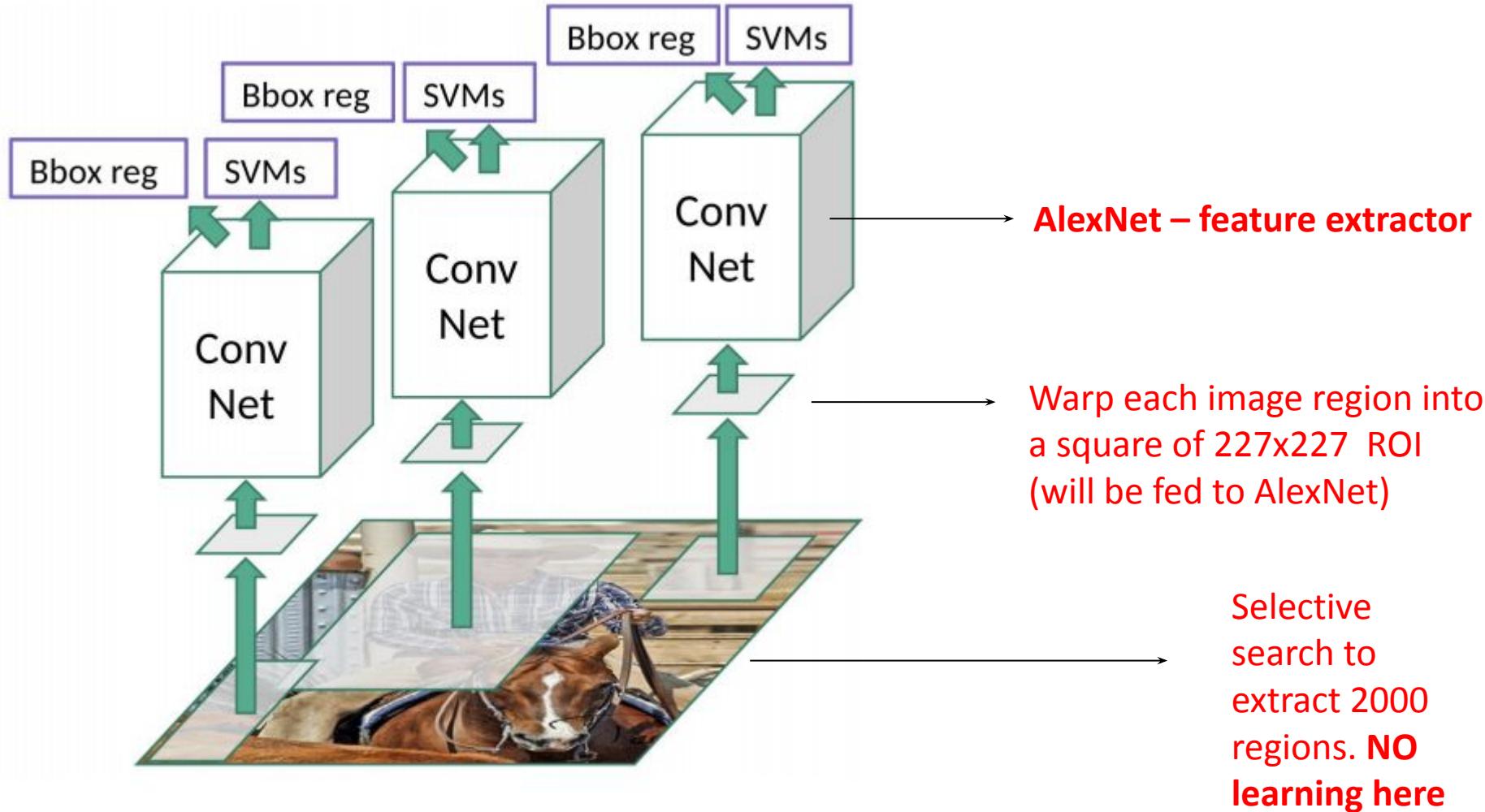
# R-CNN

## Region-based Convolutional Network, 2014



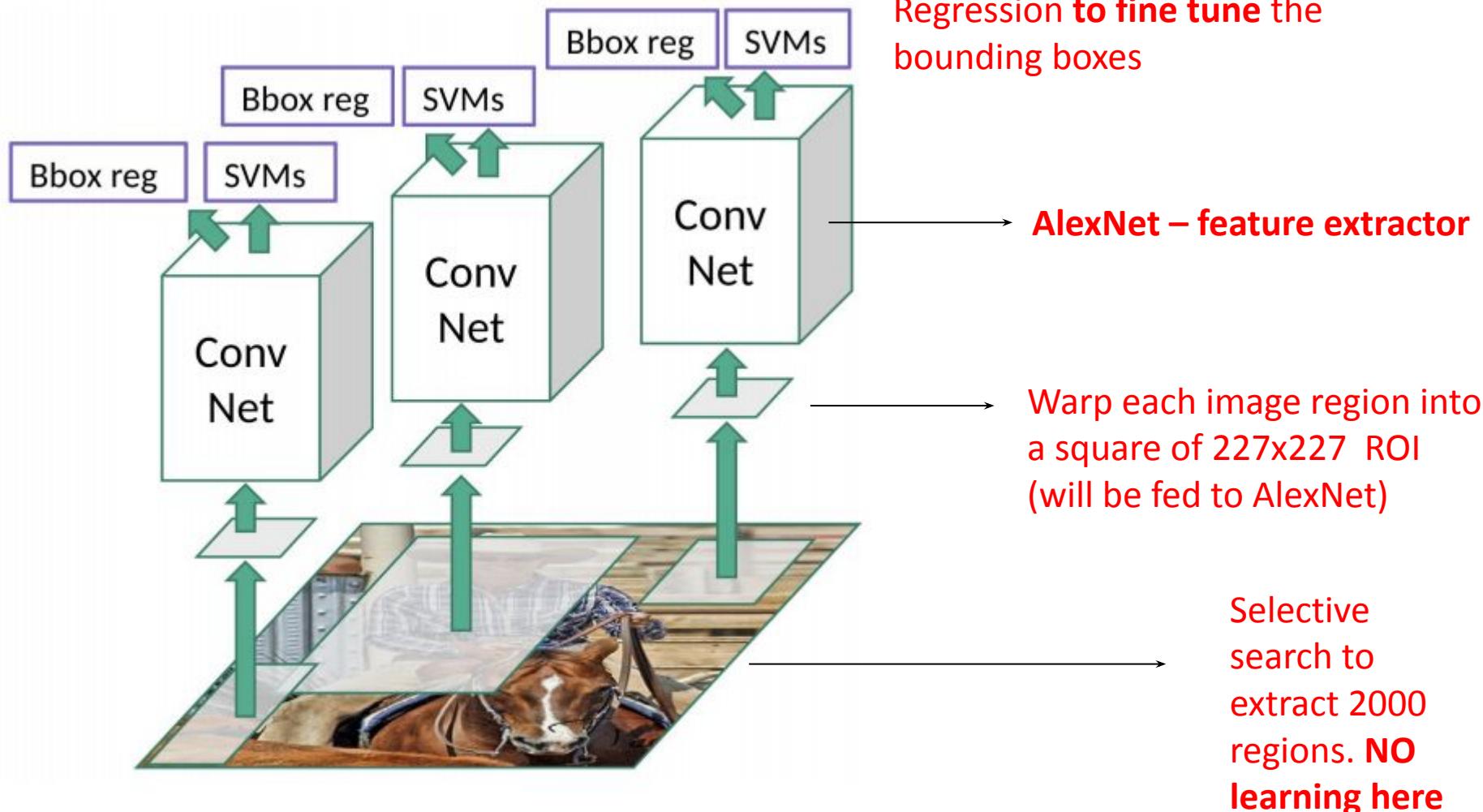
# R-CNN

## Region-based Convolutional Network, 2014



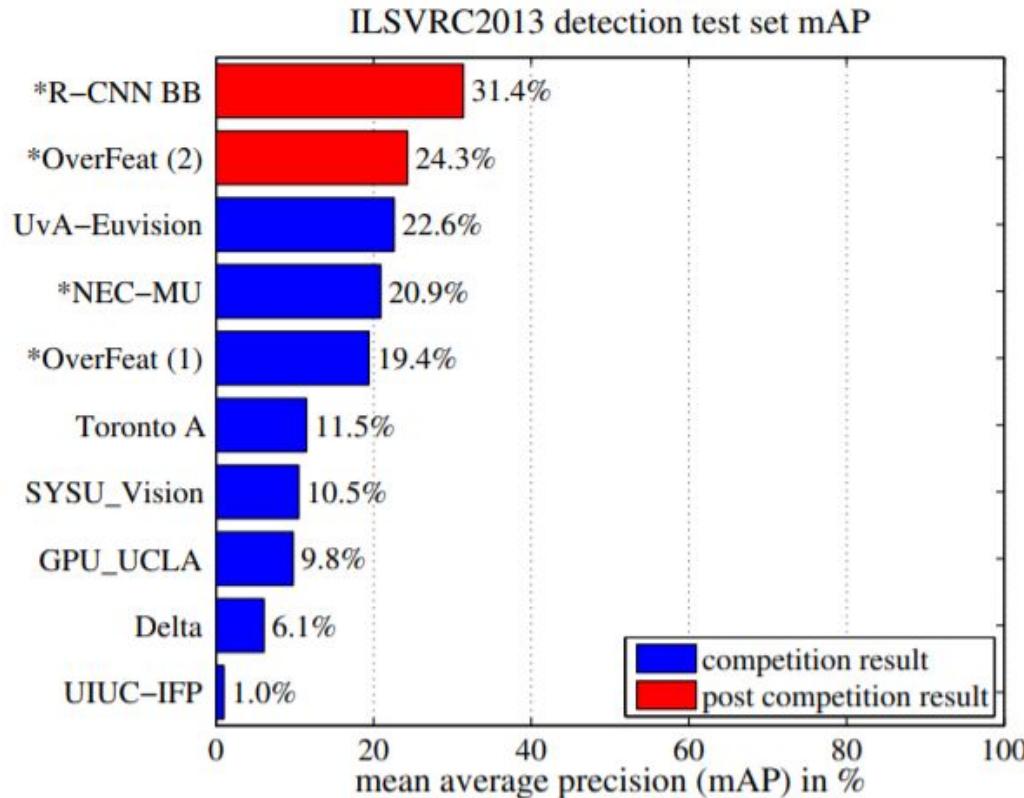
# R-CNN

## Region-based Convolutional Network, 2014



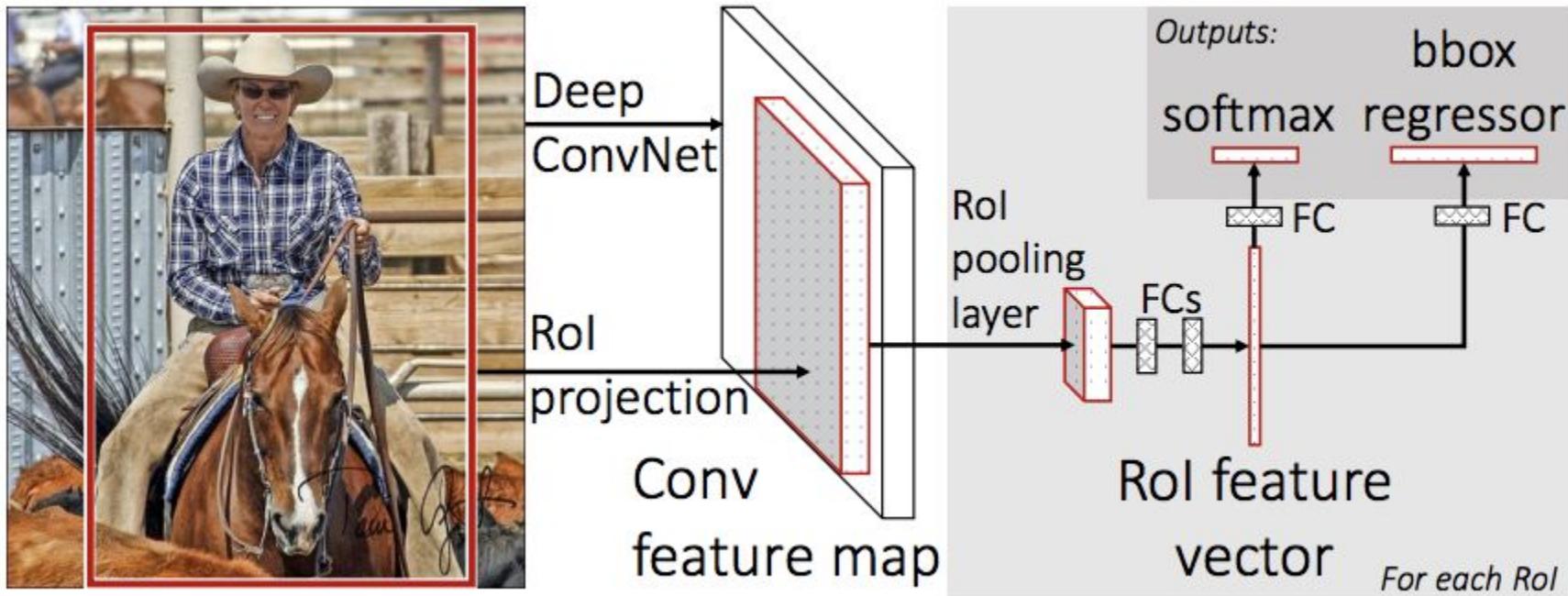
# R-CNN

- Running time: “13s/image on a GPU or 53s/image on a CPU”

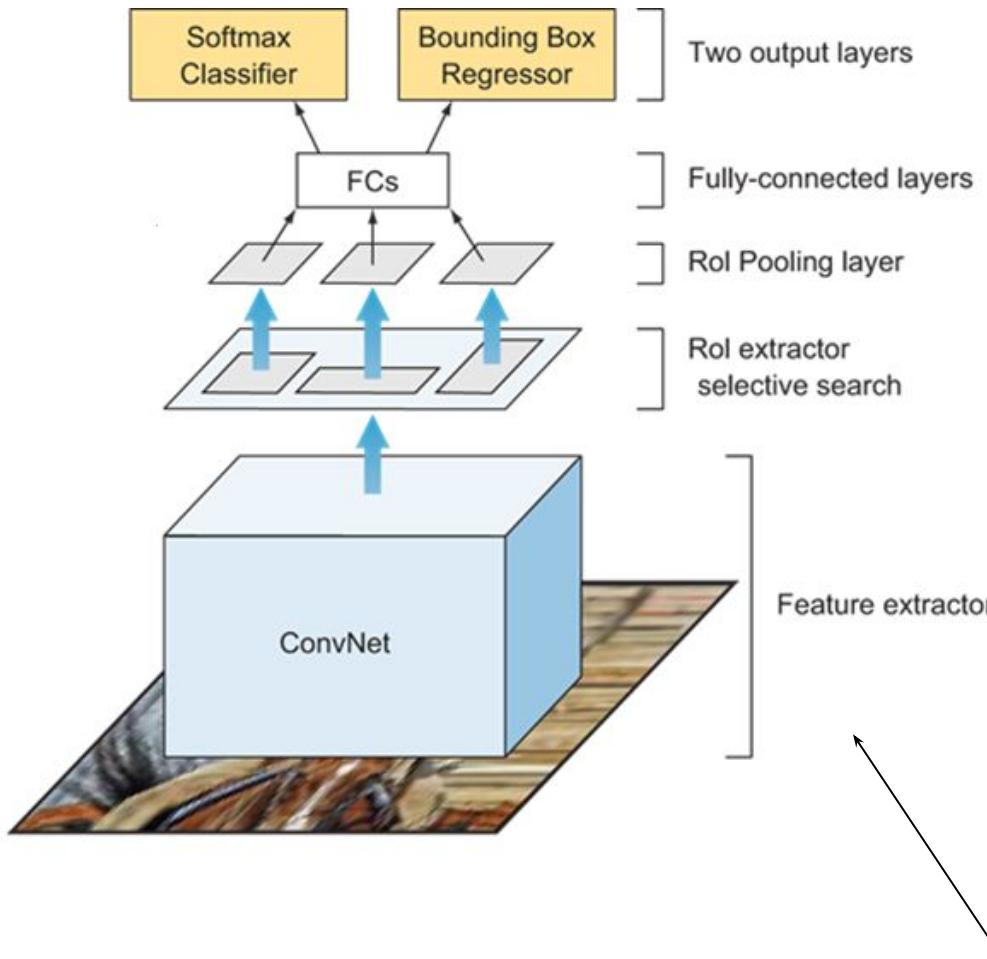


# Fast RCNN

- Idea: feed the image only once to the CNN to extract a ***feature map***, then crop and warp regions of this feature map

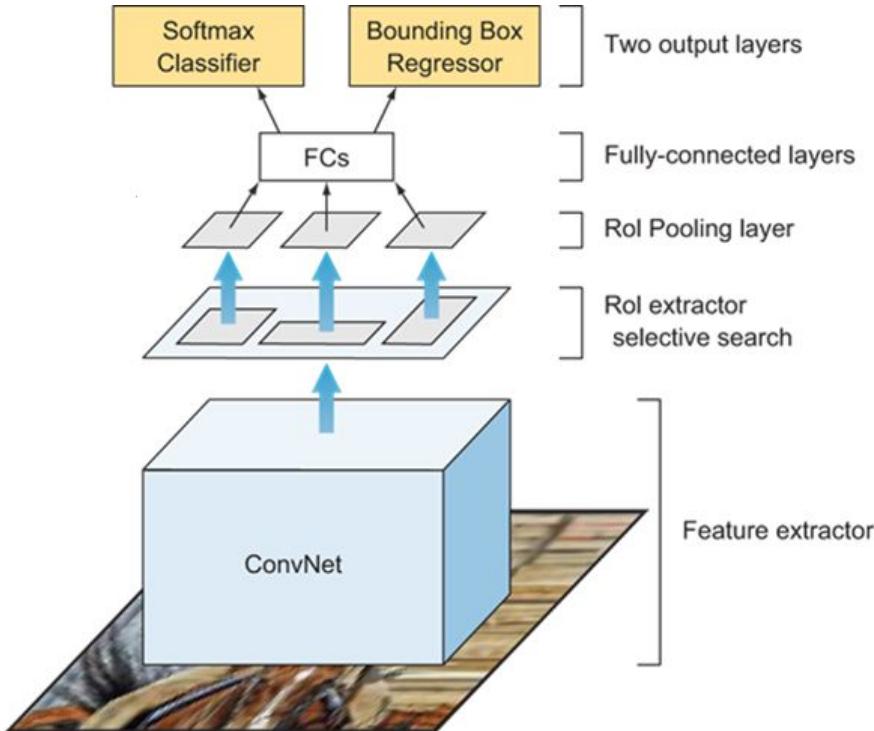


# Fast R-CNN



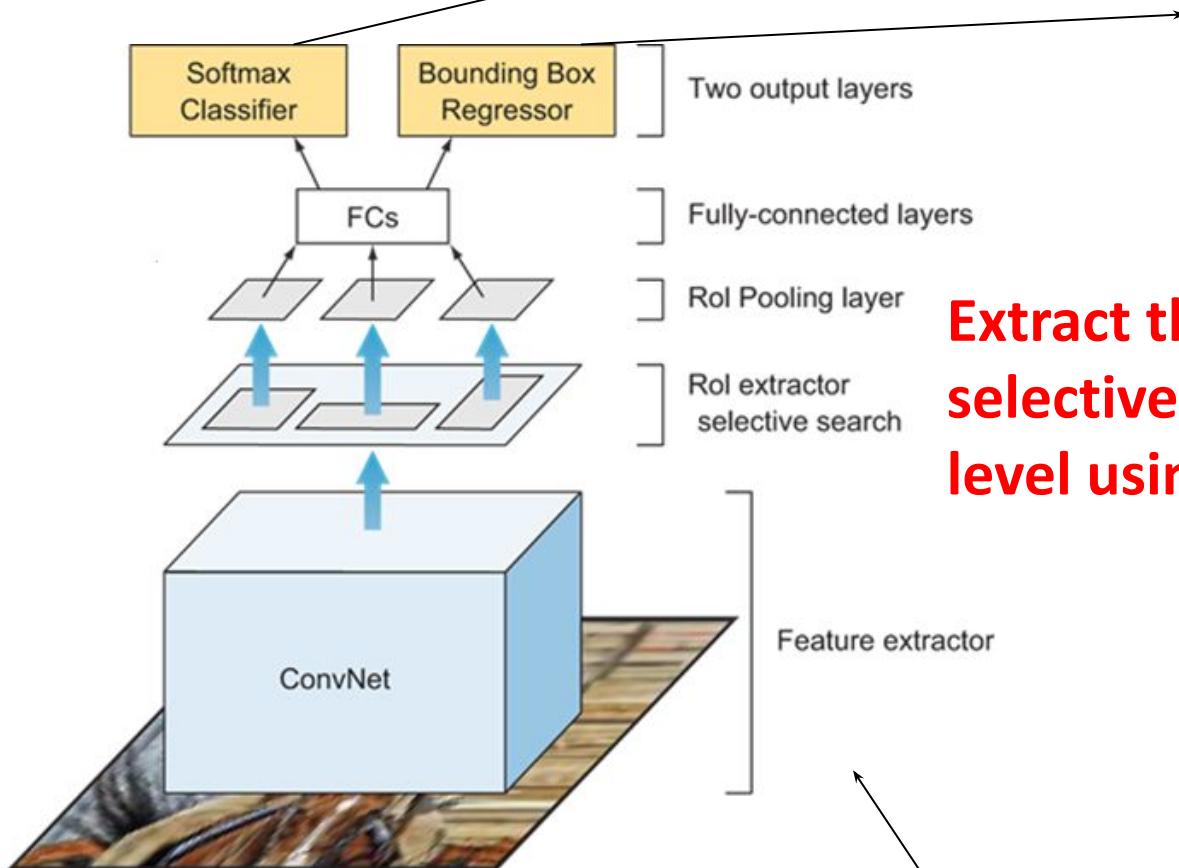
**Feed the ENTIRE image only once though the CONV net**

# Fast R-CNN



**Extract the ROIs (proposed by selective search) at feature map level using ROI Pooling**

# Fast R-CNN



N classes + background

Bounding boxes  
( $x, y, w, h$ )

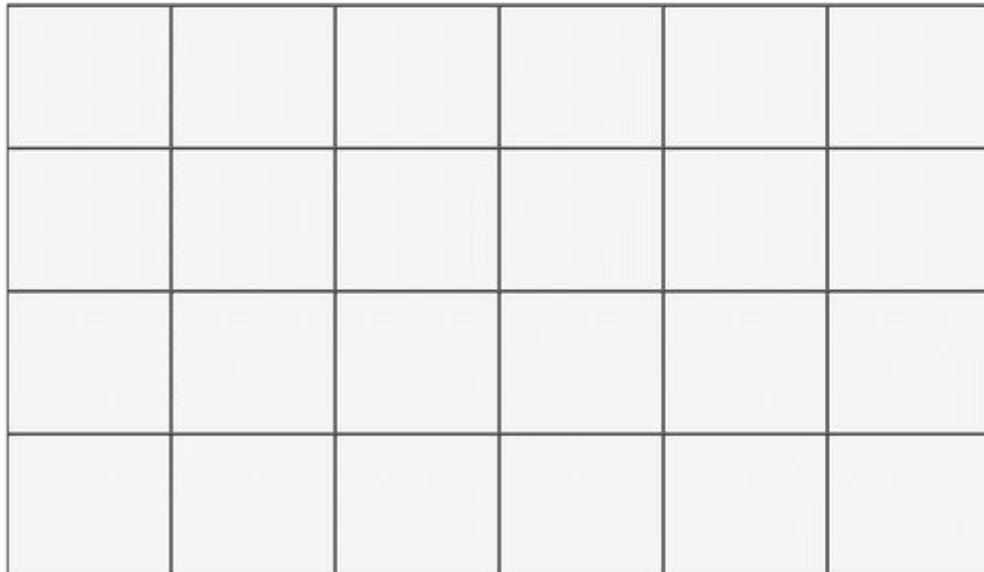
Extract the ROIs (proposed by selective search) at feature map level using RoI Pooling

Feed the ENTIRE image only once though the CONV net

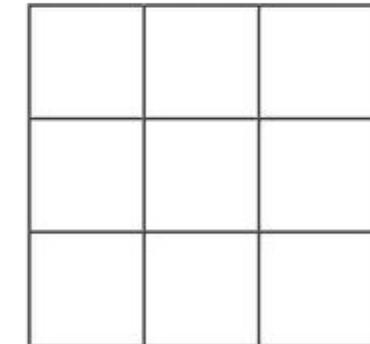
# Roi Pooling

Divide the  $H \times W$  ROI window into an  $h \times w$  grid of sub-windows of approximate size  $H/h \times W/w$  and then max-pooling the values in each sub-window into the corresponding output grid cell.

4x6 Roi



3x3 Roi Pooling



# Roi Pooling

$$W = 6, H = 4$$

$$w = 3, h = 3$$

$$sz_w = 6/3=2, sz_h=4/3 = 1$$

4x6 Roi


3x3 Roi Pooling


# Roi Pooling

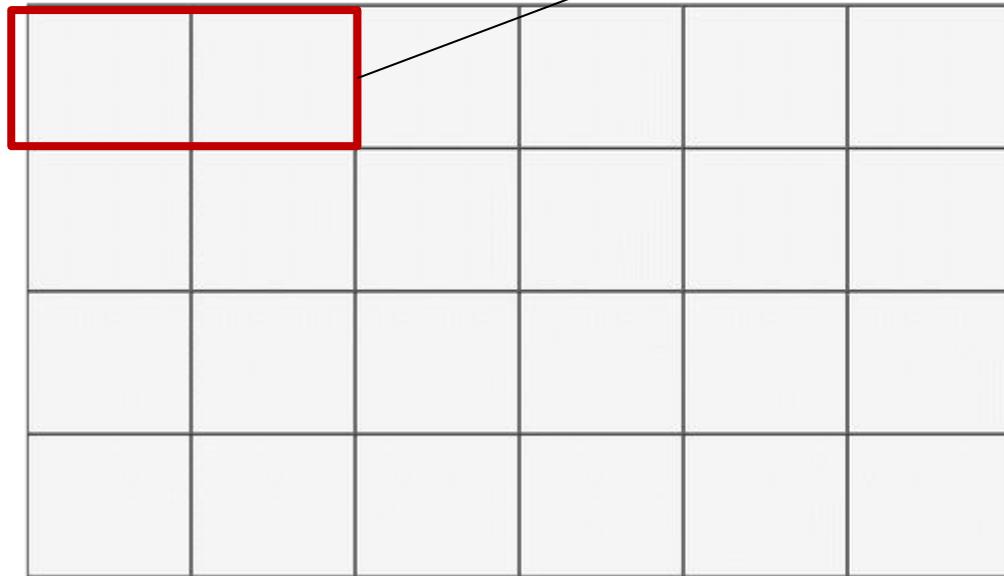
$W = 6, H = 4$

$w = 3, h = 3$

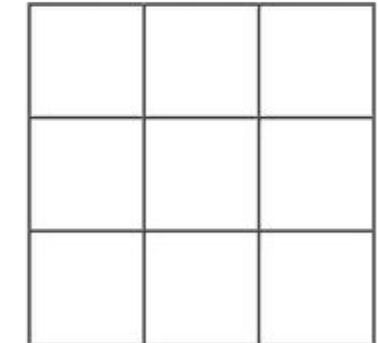
$sz_w = 6/3=2, sz_h=4/3 = 1$

**Apply max pooling in  
each ROI**

4x6 Roi



3x3 Roi Pooling



# Roi Pooling

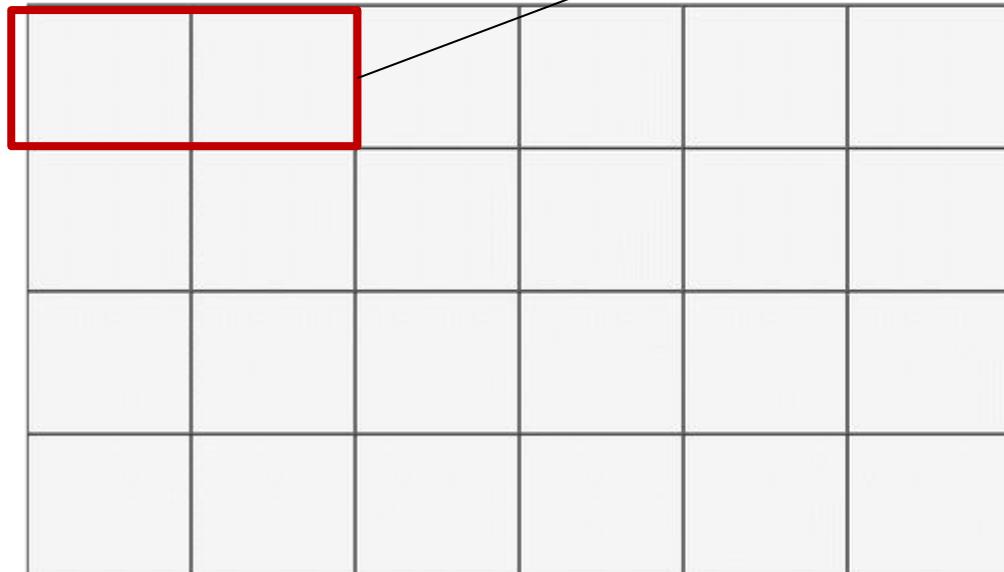
$W = 6, H = 4$

$w = 3, h = 3$

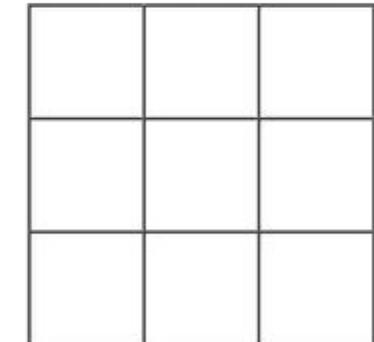
$sz_w = 6/3=2, sz_h=4/3 = 1$

Apply max pooling in  
each ROI

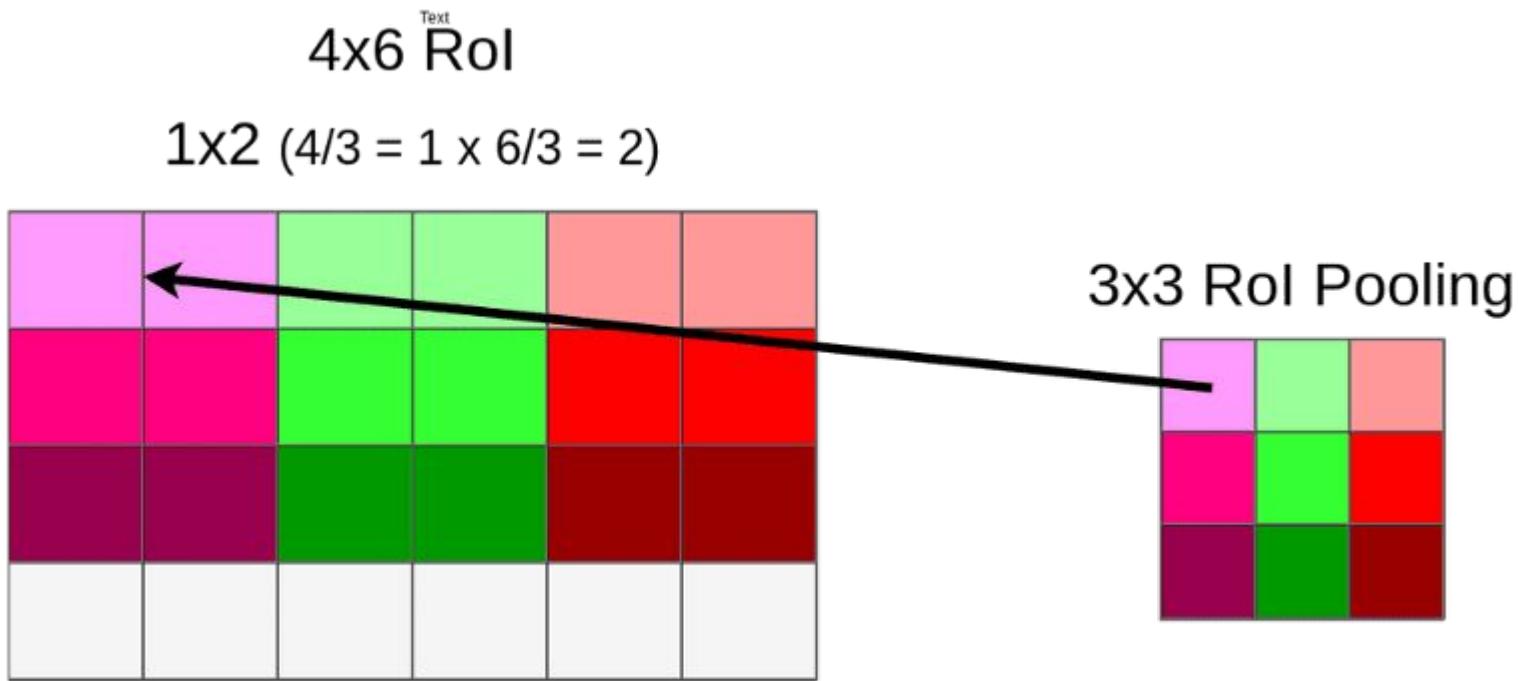
4x6 Roi



3x3 Roi Pooling



# ROI Pooling example

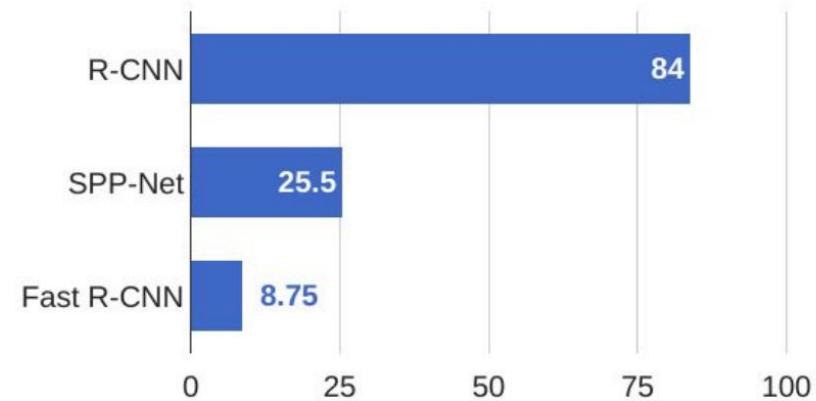


<https://towardsdatascience.com/understanding-region-of-interest-part-2-roi-align-and-roi-warped-f795196fc193>

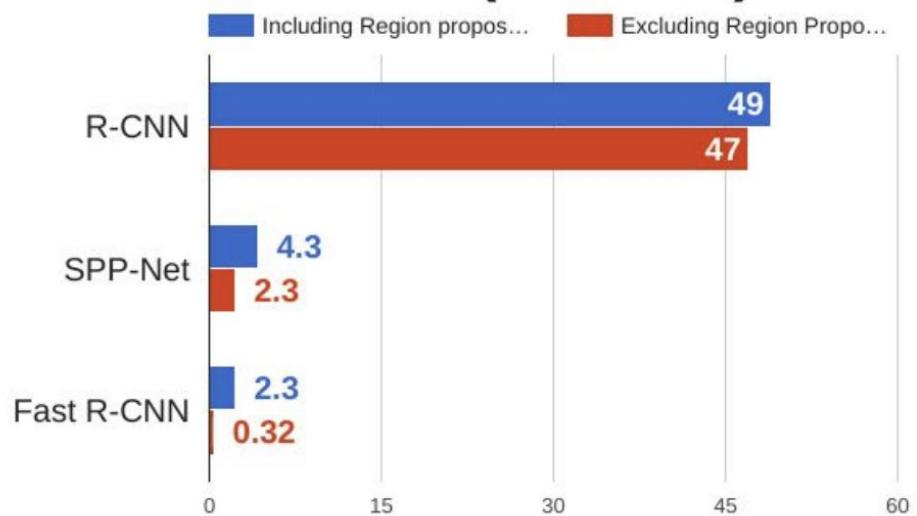
<https://towardsdatascience.com/understanding-region-of-interest-part-1-roi-pooling-e4f5dd65bb44>

# Fast RCNN

Training time (Hours)



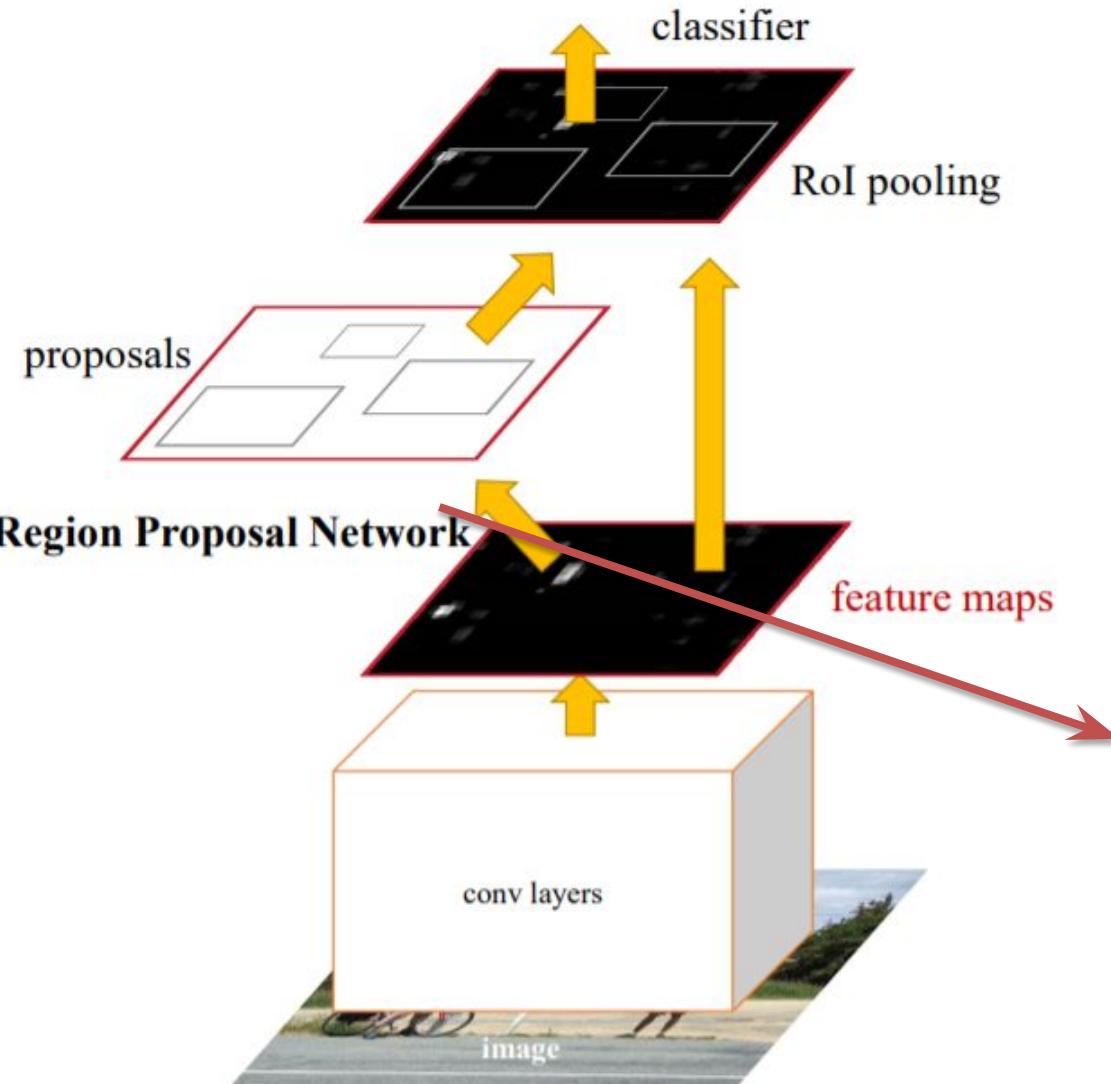
Test time (seconds)



What is the most time consuming part of Fast RCNN?

# Faster-RCNN

Unified framework for object detection



The rest of the network is similar to Fast RCNN: use ROI pooling to crop feature maps and classify each region

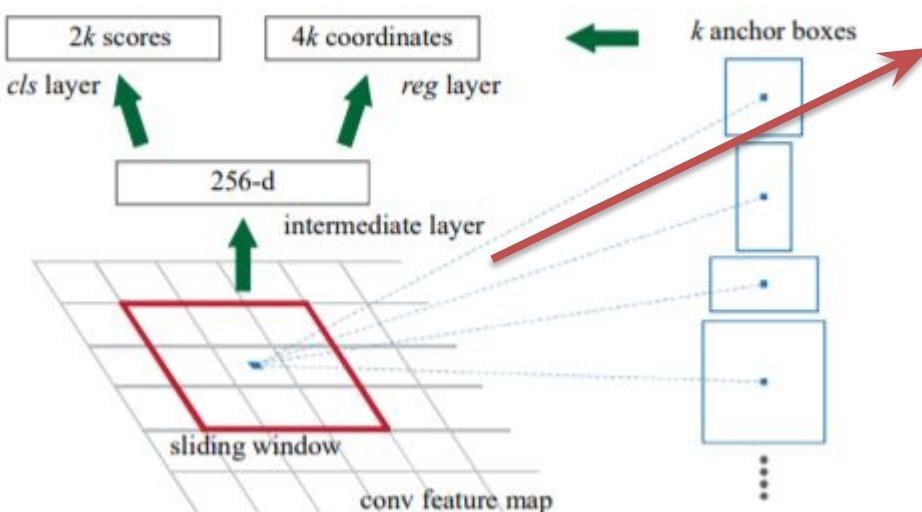
Region proposal network:  
replaces selective search and  
predicts regions directly from  
feature maps

# Faster RCNN

- Multi-loss training
  - Region proposal network: objectness classification
  - Region proposal network: bbox coordinates
  - Final classification score
  - Final bbox coordinates

# Region proposal network

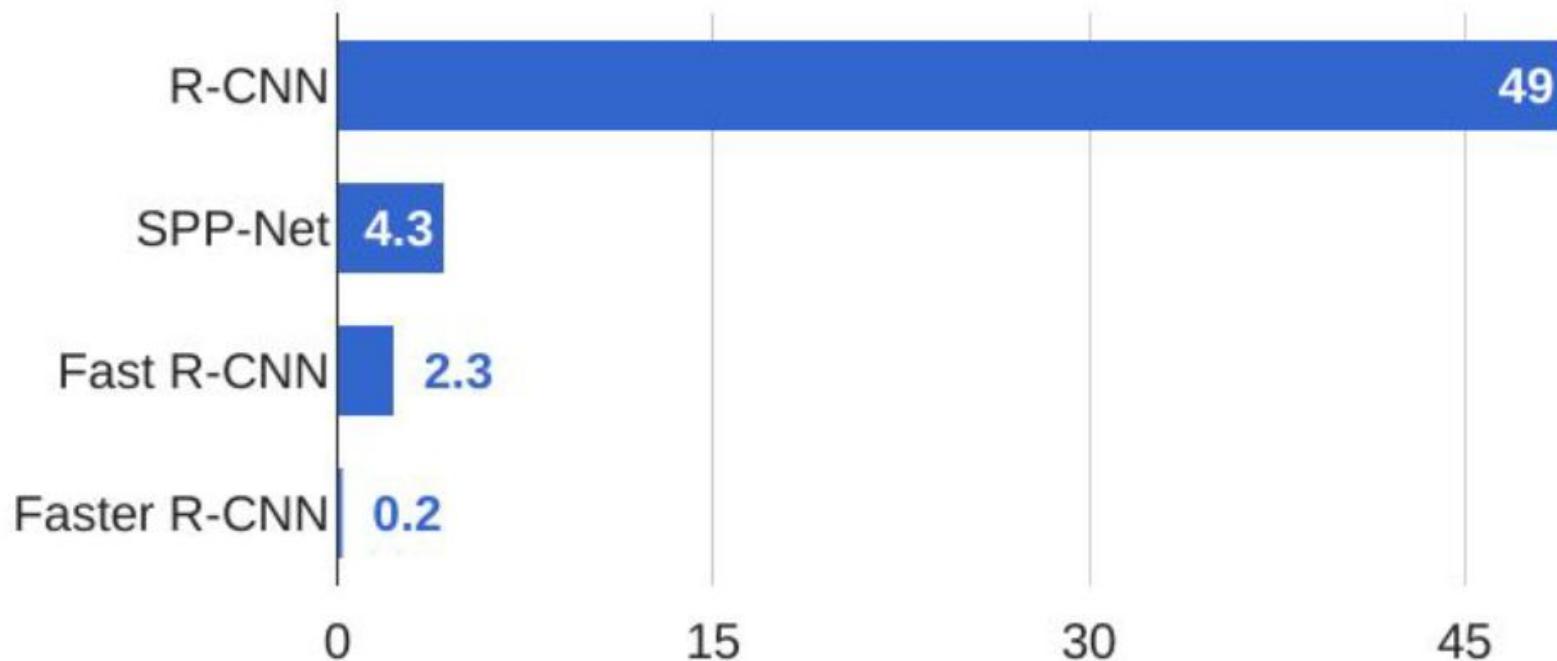
- takes an image (of any size) as input and outputs a set of rectangular object proposals, each with an objectness score



An **anchor box** of fixed size at each point in the feature map

- For each anchor box predict if there is an object inside it
- For “positives” predict corrections for the bounding boxes

## R-CNN Test-Time Speed



# Faster RCNN

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*). \quad (1)$$

Here,  $i$  is the index of an anchor in a mini-batch and  $p_i$  is the predicted probability of anchor  $i$  being an object. The ground-truth label  $p_i^*$  is 1 if the anchor is positive, and is 0 if the anchor is negative.  $t_i$  is a vector representing the 4 parameterized coordinates of the predicted bounding box, and  $t_i^*$  is that of the ground-truth box associated with a positive anchor. The classification loss  $L_{cls}$  is log loss over two classes (object *vs.* not object). For the regression loss, we use  $L_{reg}(t_i, t_i^*) = R(t_i - t_i^*)$  where  $R$  is the robust loss function (smooth L<sub>1</sub>) defined in [2]. The term  $p_i^* L_{reg}$  means the regression loss is activated only for positive anchors ( $p_i^* = 1$ ) and is disabled otherwise ( $p_i^* = 0$ ). The outputs of the *cls* and *reg* layers consist of  $\{p_i\}$  and  $\{t_i\}$  respectively.

$$L_{1;smooth} = \begin{cases} |x| & \text{if } |x| > \alpha; \\ \frac{1}{|\alpha|}x^2 & \text{if } |x| \leq \alpha \end{cases}$$

$$\begin{aligned} t_x &= (x - x_a)/w_a, & t_y &= (y - y_a)/h_a, \\ t_w &= \log(w/w_a), & t_h &= \log(h/h_a), \\ t_x^* &= (x^* - x_a)/w_a, & t_y^* &= (y^* - y_a)/h_a, \\ t_w^* &= \log(w^*/w_a), & t_h^* &= \log(h^*/h_a), \end{aligned}$$

$t^*$  - ground truth

$t$  – prediction

$h_a, w_a$  – anchor box

bounding-box regression from an anchor box to a nearby ground-truth box.

	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image (with proposals)	50 seconds	2 seconds	<b>0.2 seconds</b>
(Speedup)	1x	25x	<b>250x</b>
mAP (VOC 2007)	66.0	<b>66.9</b>	<b>66.9</b>

# Instance segmentation

# Mask R-CNN

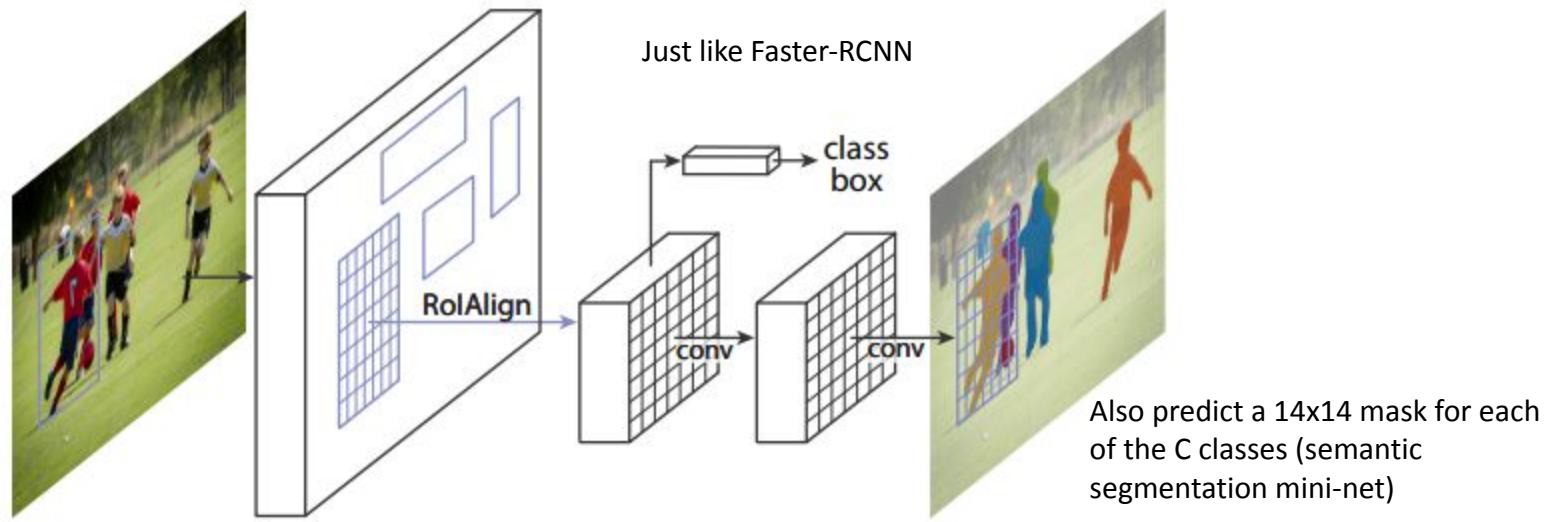


Figure 1. The **Mask R-CNN** framework for instance segmentation.

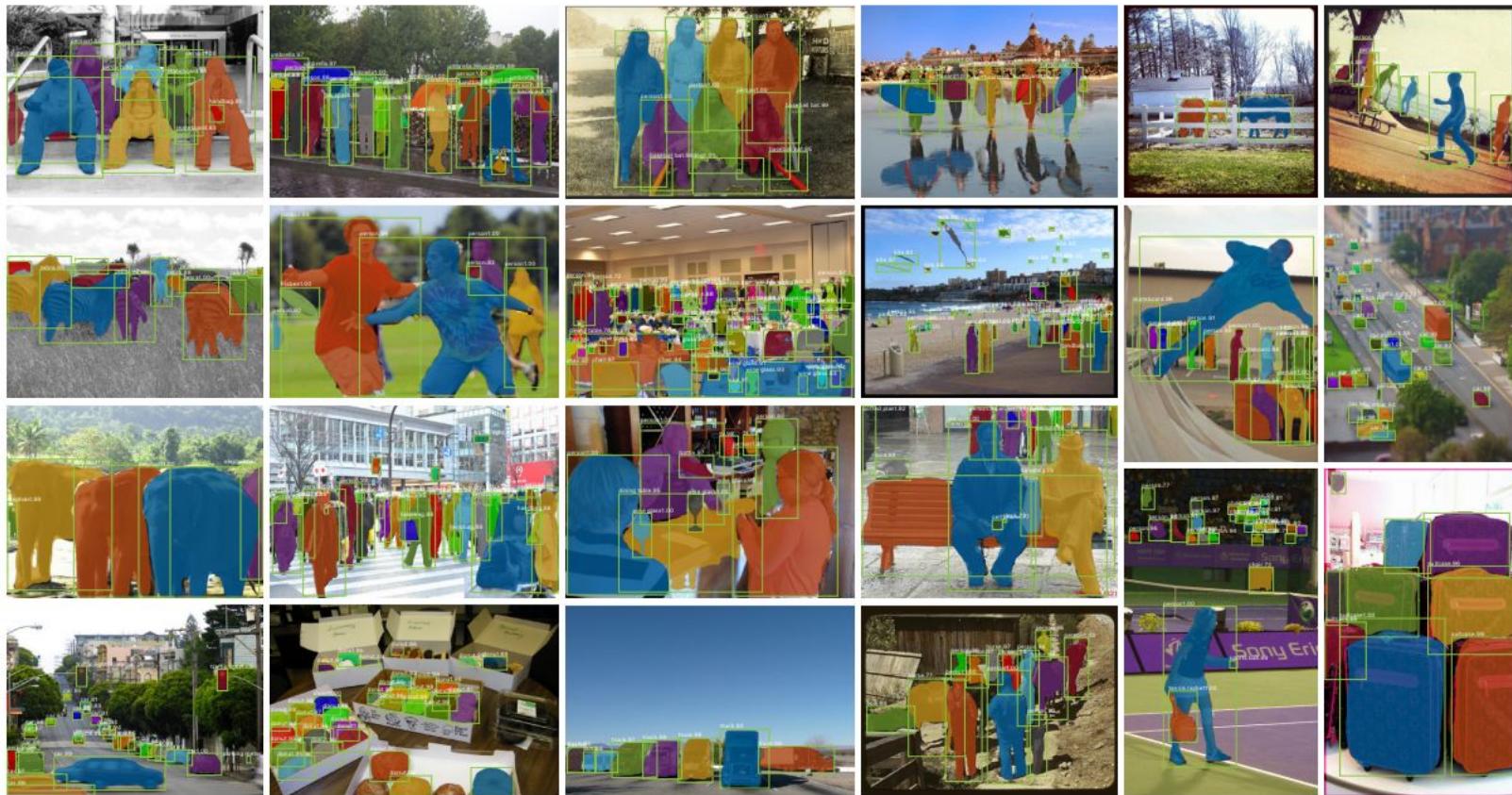


Figure 5. More results of **Mask R-CNN** on COCO test images, using ResNet-101-FPN and running at 5 fps, with 35.7 mask AP (Table 1).

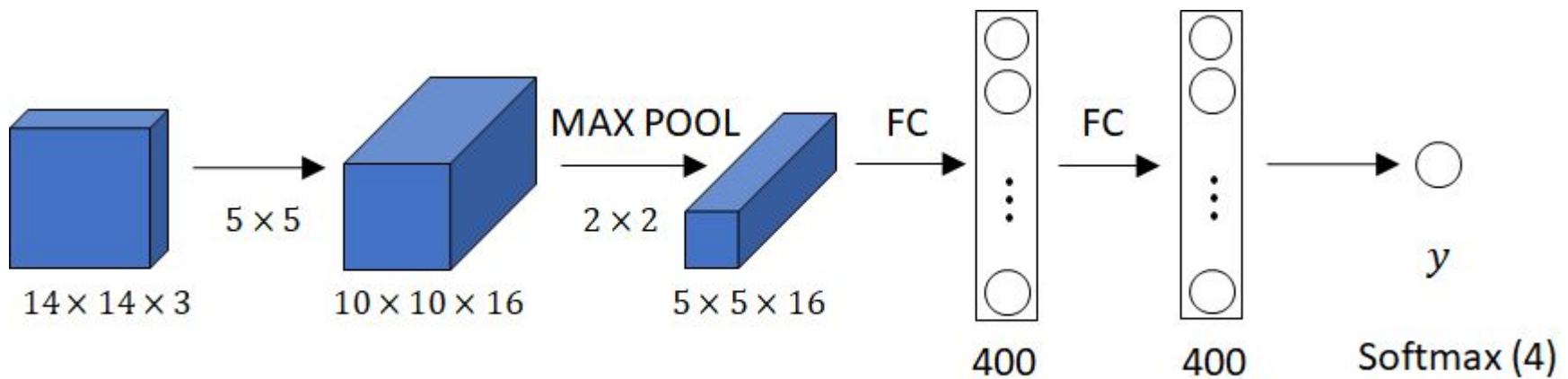
# Object detection without proposals



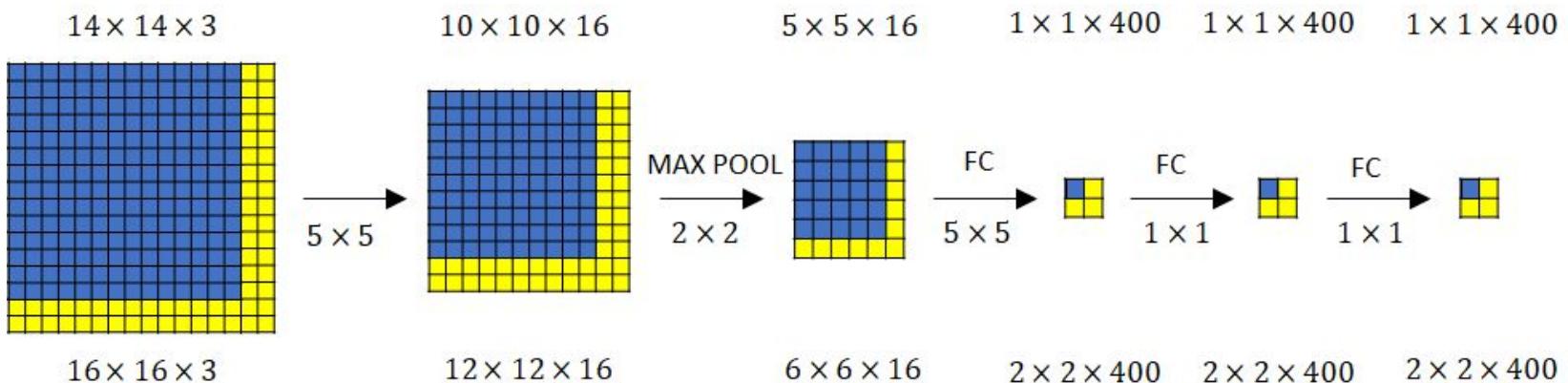
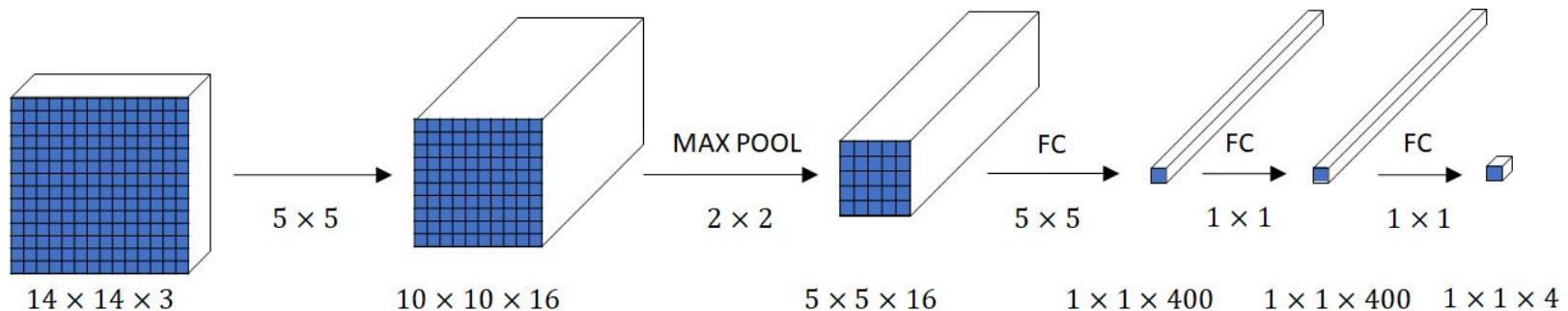
100

100

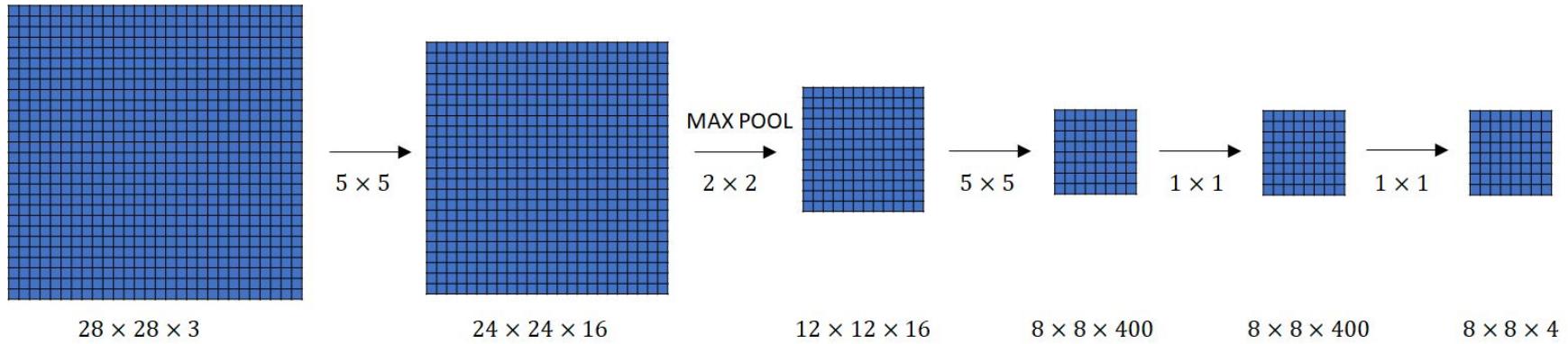
# Convolutional implementation of sliding windows



# Convolutional implementation of sliding windows



# Convolutional implementation of sliding windows



# YOLO

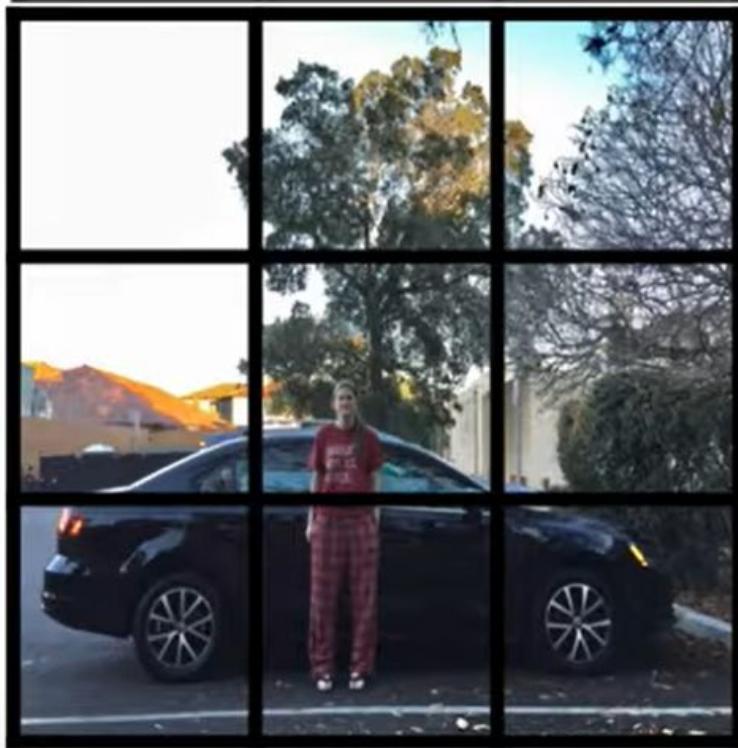
You only look once

- Detection is modelled as a regression problem
  - image is divided into an  $S \times S$  grid
  - for each grid cell predict:
    - B bounding boxes (4 values per bbox)
    - confidence for those boxes,
    - C class probas

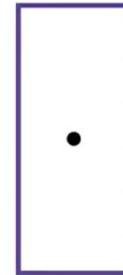
# YOLO

## Anchor boxes

- Allows the object detector to specialize better (detect “thinner” or “wider” objects)

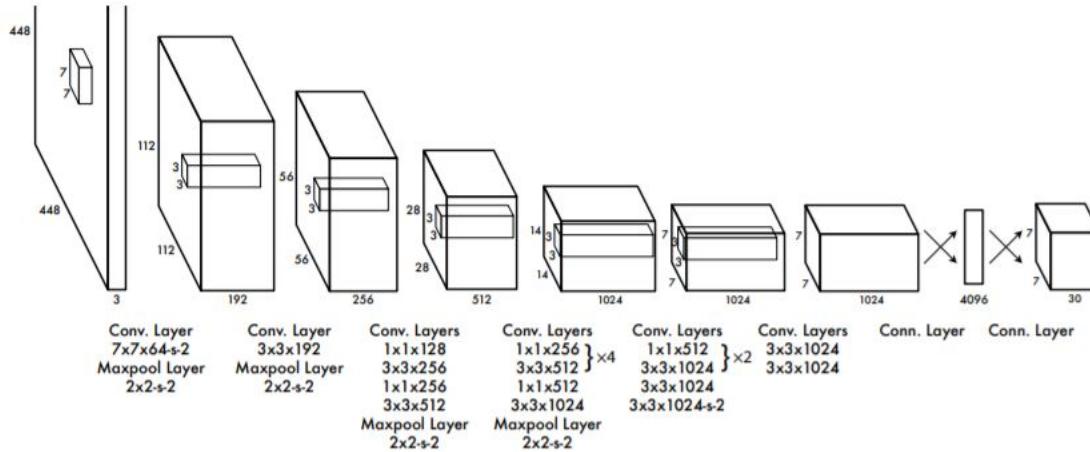


Anchor box 1:



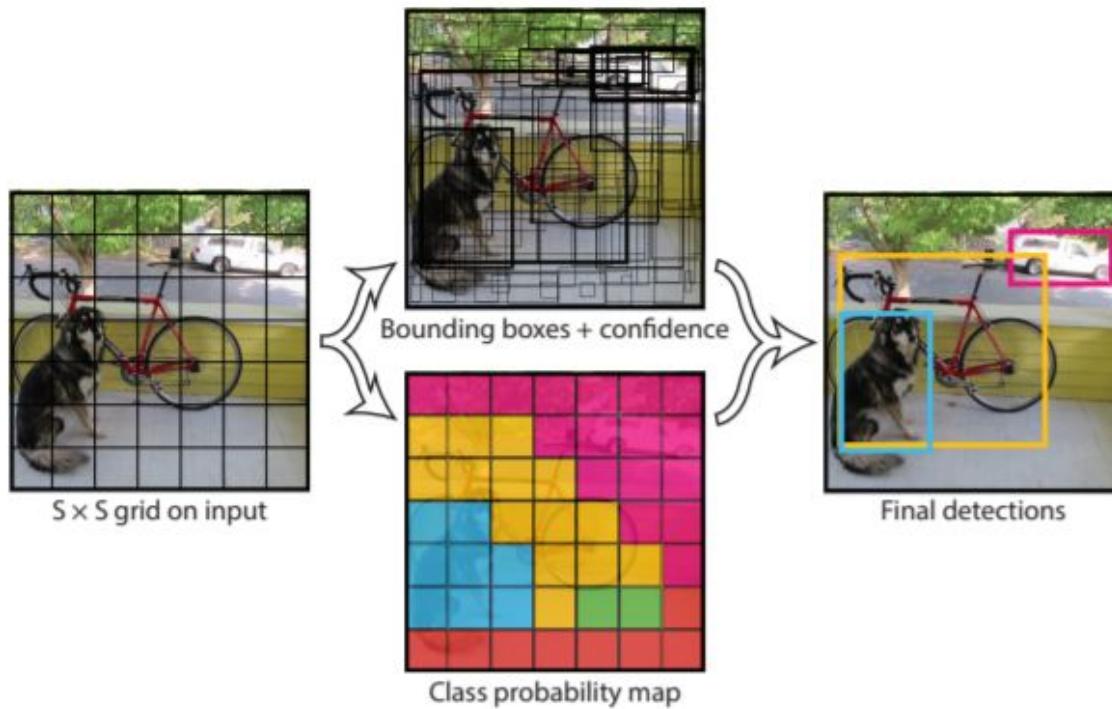
Anchor box 2:



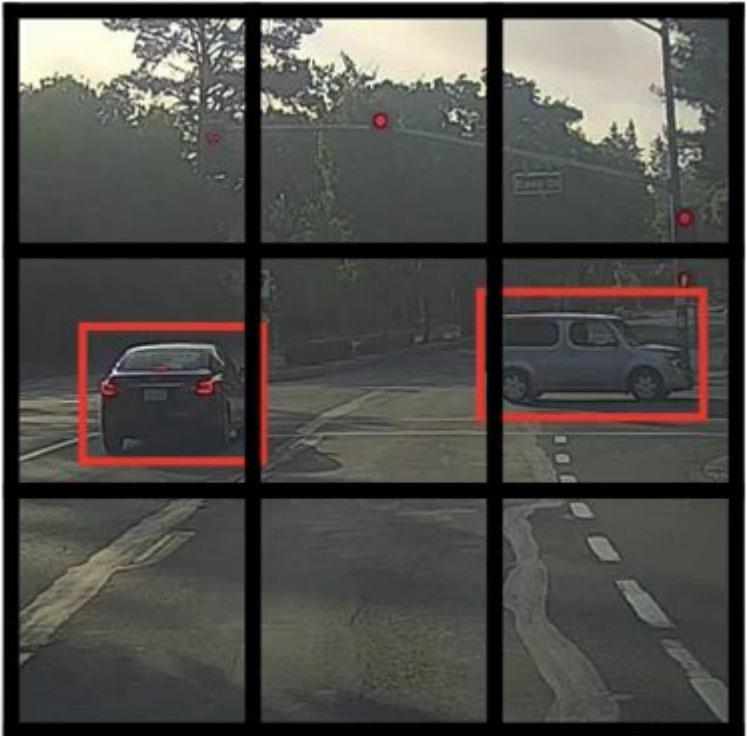


**Figure 3: The Architecture.** Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating  $1 \times 1$  convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution ( $224 \times 224$  input image) and then double the resolution for detection.

For evaluating YOLO on PASCAL VOC, we use  $S = 7$ ,  $B = 2$ . PASCAL VOC has 20 labelled classes so  $C = 20$ . Our final prediction is a  $7 \times 7 \times 30$  tensor



**Figure 2: The Model.** Our system models detection as a regression problem. It divides the image into an  $S \times S$  grid and for each grid cell predicts  $B$  bounding boxes, confidence for those boxes, and  $C$  class probabilities. These predictions are encoded as an  $S \times S \times (B * 5 + C)$  tensor.



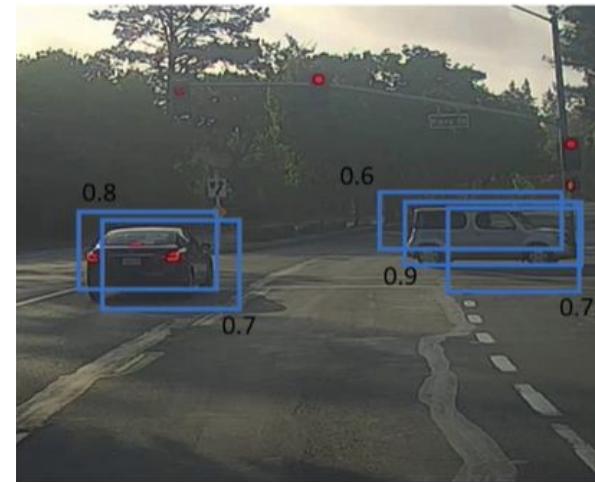
100

100

# YOLO

Non maximum suppression

- Discard all predictions with  $\text{proba} \leq \text{threshold}$
- **while** there are any remaining boxes
  - Select the box with the largest proba ( $\text{maxBB}$ ) as a prediction
  - Discard boxes with  $\text{IoU} \geq 0.5$  with  $\text{maxBB}$



# YOLO

## Loss function

This is the loss function for yolo v1 paper.

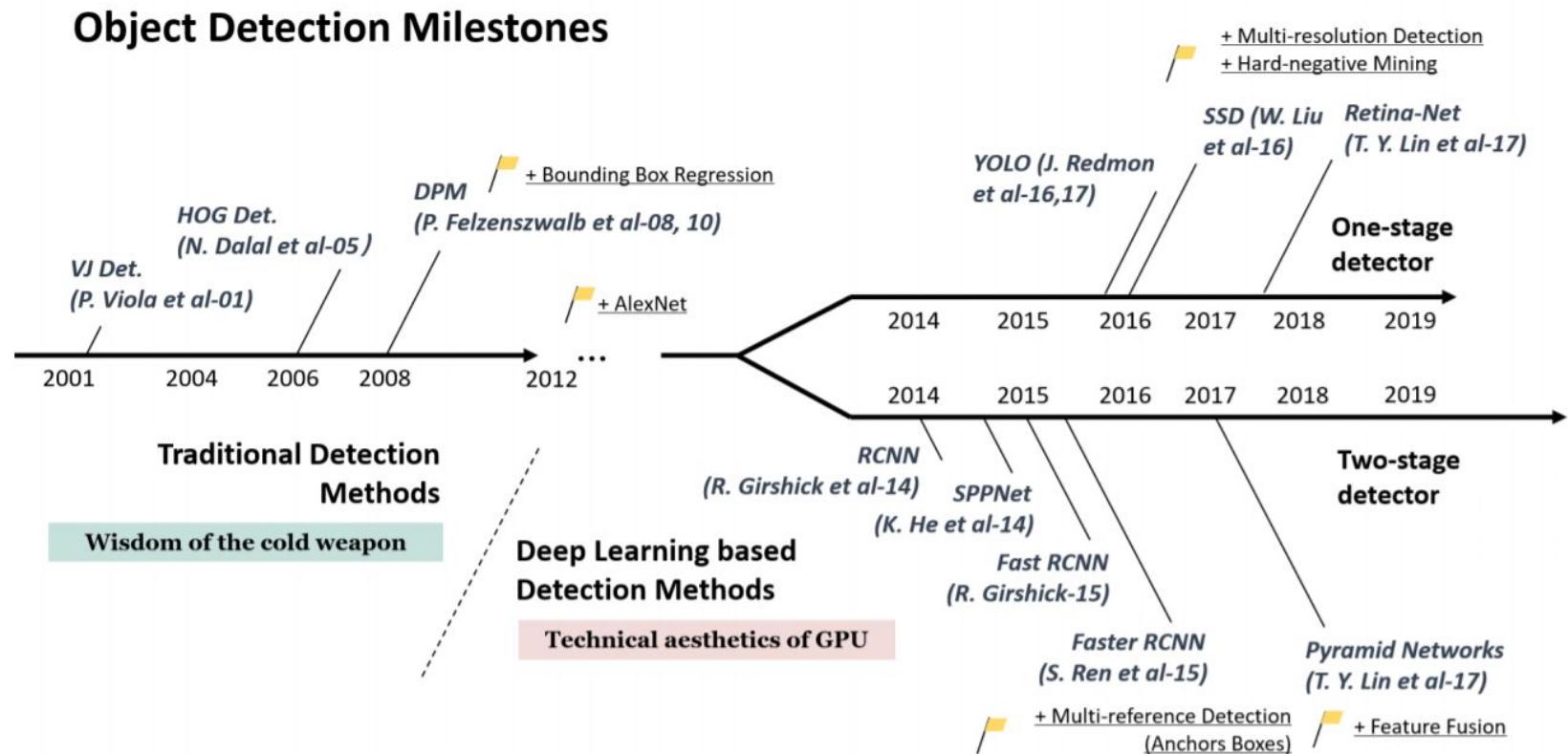
Here each bounding box predicts an objectness score and a 4 coordinates, but the class predictions are per cell. Hence this:

$$\begin{aligned} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[ (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\ & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\ & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \quad (3) \end{aligned}$$

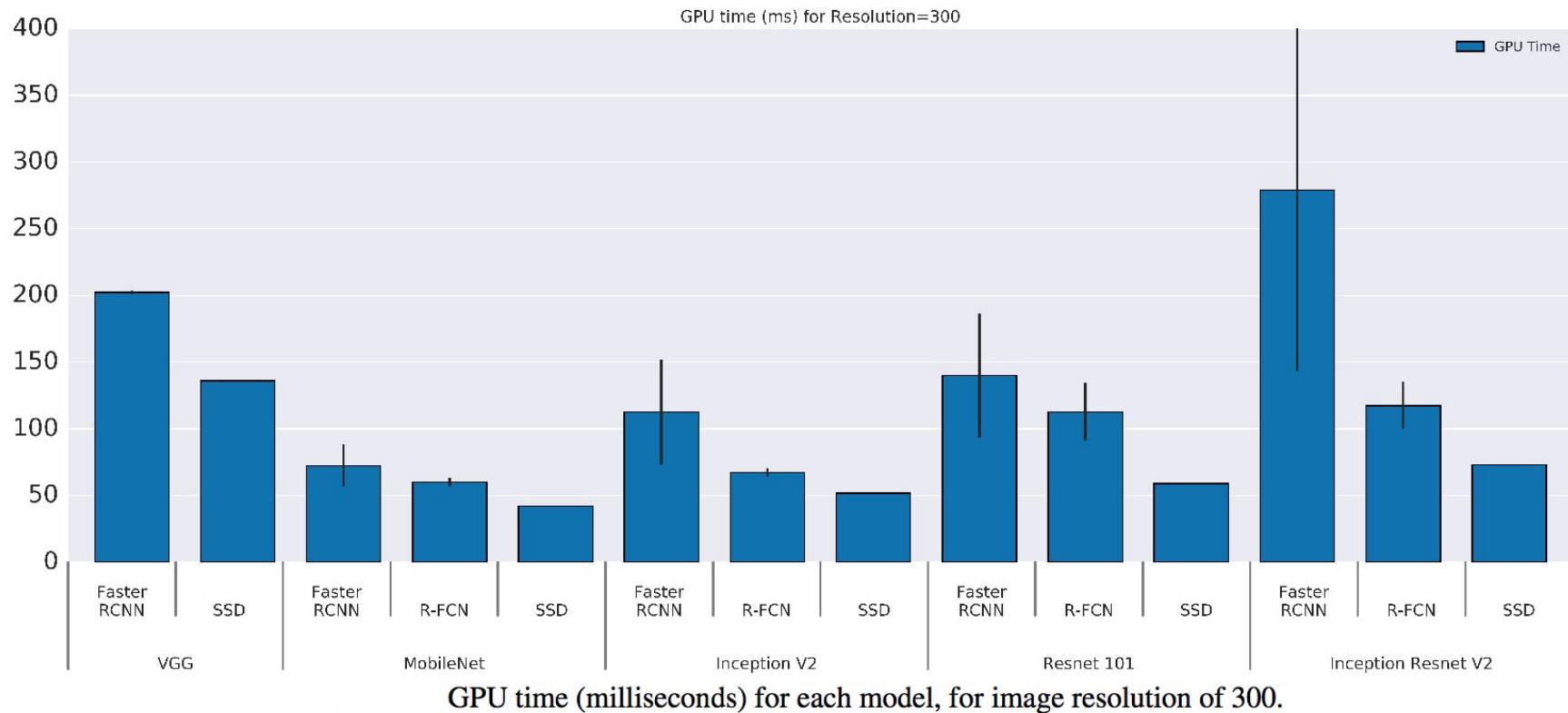
where  $\mathbb{1}_i^{\text{obj}}$  denotes if object appears in cell  $i$  and  $\mathbb{1}_{ij}^{\text{obj}}$  denotes that the  $j$ th bounding box predictor in cell  $i$  is “responsible” for that prediction.

# Object detectors

## Object Detection in 20 Years: A Survey



# Object detectors



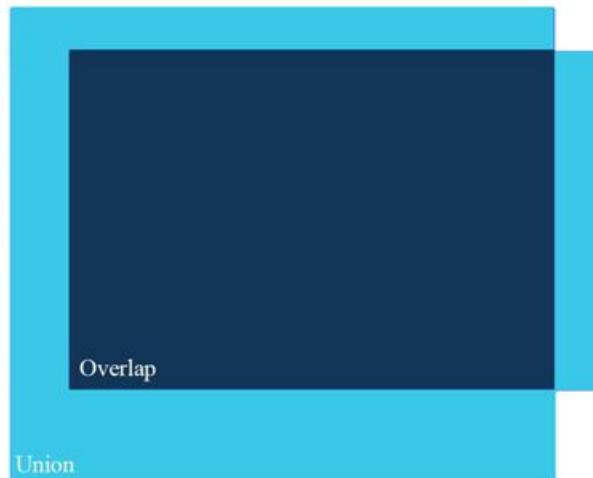
Real-Time Detectors	Train	mAP	FPS
100Hz DPM [30]	2007	16.0	100
30Hz DPM [30]	2007	26.1	30
Fast YOLO	2007+2012	52.7	<b>155</b>
YOLO	2007+2012	<b>63.4</b>	45
<hr/>			
Less Than Real-Time			
Fastest DPM [37]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[27]	2007+2012	73.2	7
Faster R-CNN ZF [27]	2007+2012	62.1	18

# Evaluation metrics for object detection



- Ground truth
- Prediction

$$IoU = \frac{\text{area of overlap}}{\text{area of union}}$$

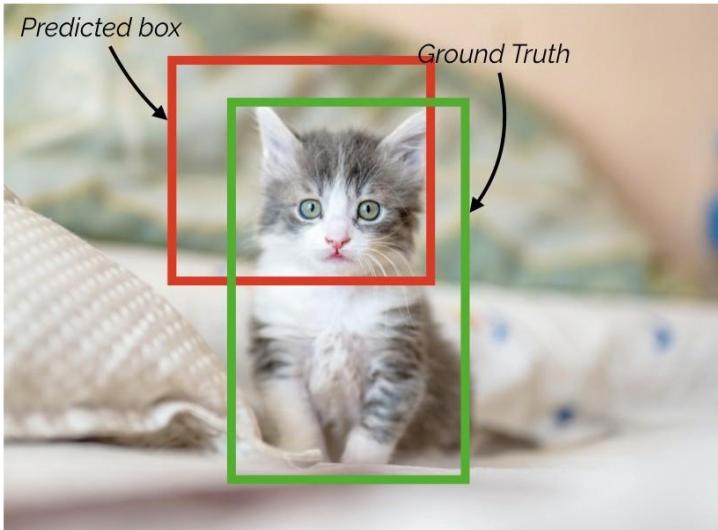


Use IOU to determine if the object is a TP, FP, or a FN

Remember **precision** and **recall**?

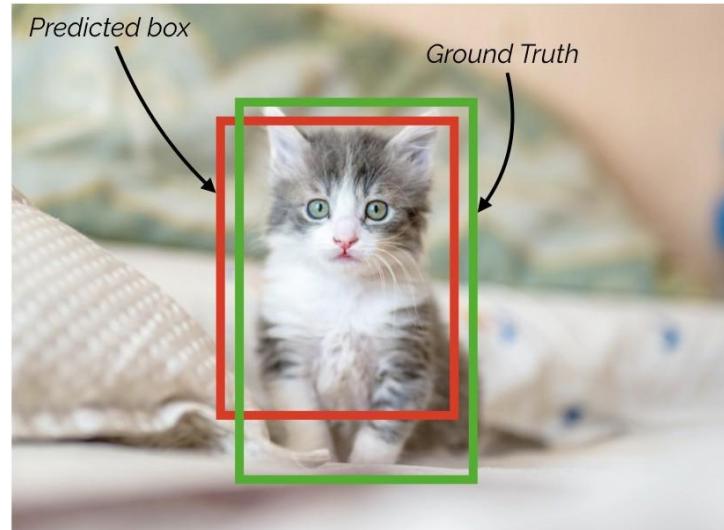
**If IoU threshold = 0.5**

**False Positive (FP)**



$$IoU = \sim 0.3$$

**True Positive (TP)**



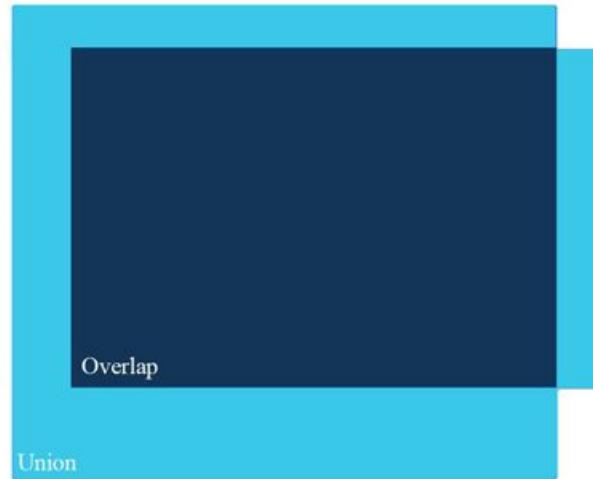
$$IoU = \sim 0.7$$

# Evaluation metrics for object detection



- Ground truth
- Prediction

$$IoU = \frac{\text{area of overlap}}{\text{area of union}}$$



Use **IOU** and a **threshold** to determine if the object is a TP, FP, or a FN

Remember **precision** and **recall**?

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

# Evaluation metrics for object detection

## AP

**Average Precision (AP)** is finding the area under the precision-recall curve.

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0	0.4
3	False	0.67	0.4
4	False	0.5	0.4
5	False	0.4	0.4
6	True	0.5	0.6
7	True	0.57	0.8
8	False	0.5	0.8
9	False	0.44	0.8
10	True	0.5	1.0

# Evaluation metrics for object detection

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0	0.4
3	False	0.67	0.4
4	False	0.5	0.4
5	False	0.4	0.4
6	True	0.5	0.6
7	True	0.57	0.8
8	False	0.5	0.8
9	False	0.44	0.8
10	True	0.5	1.0



TP?  
FP?  
FN?  
Precision?  
Recall?

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

# Computer Vision and Deep Learning

Lecture 11

# Understanding what networks learn

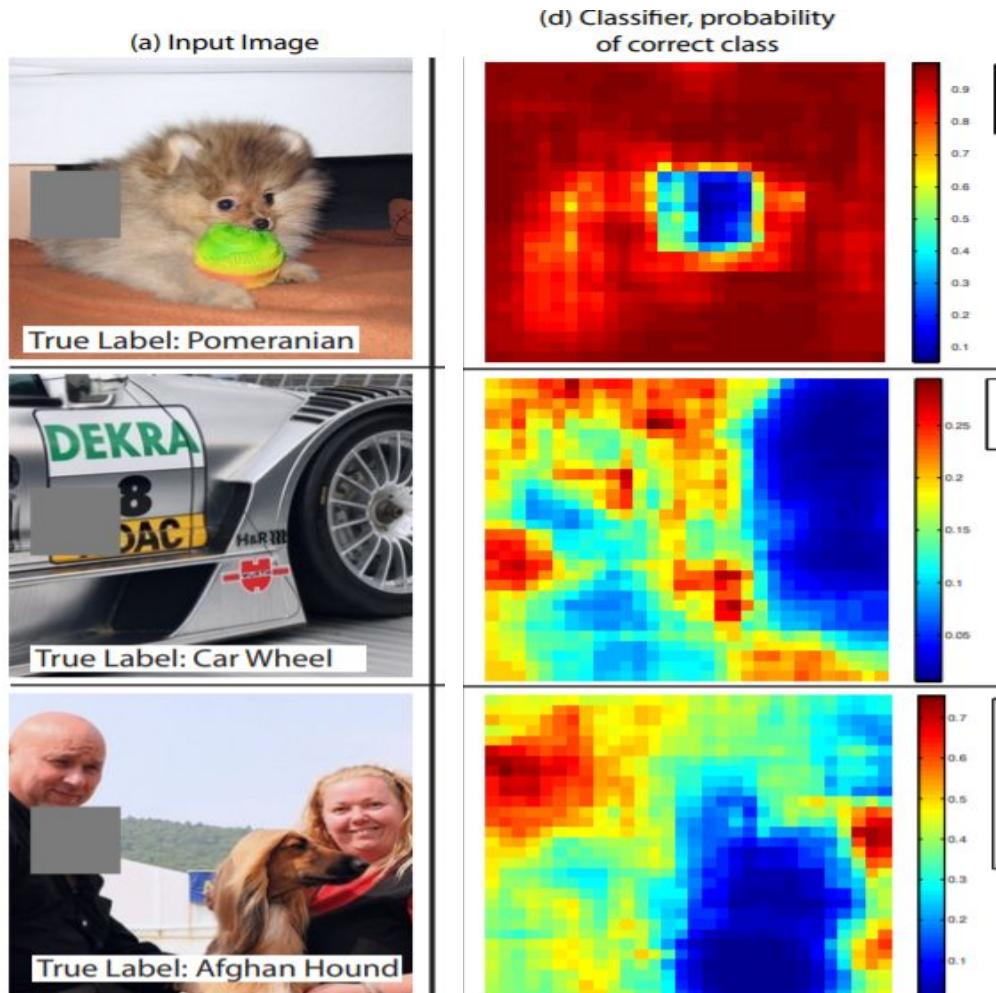
*Much learning does not teach understanding.* - Heraclitus



# Today's agenda

- Visualization
  - Filters
  - Activations
  - Areas that trigger a neuron
  - Embeddings
  - DeepDream
- Adversarial examples

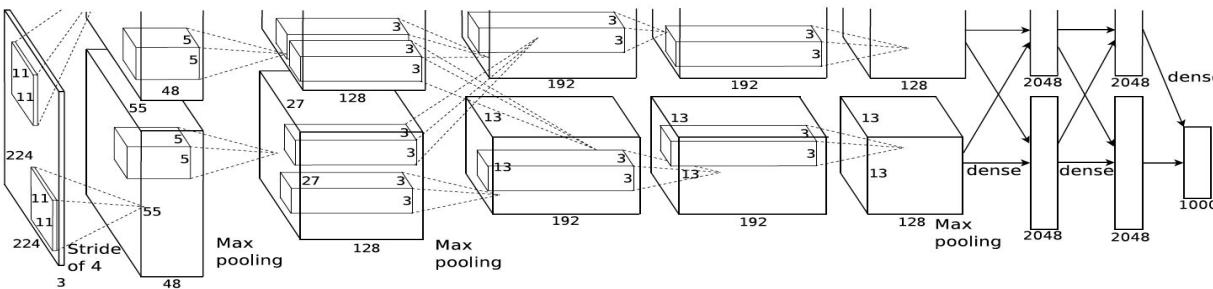
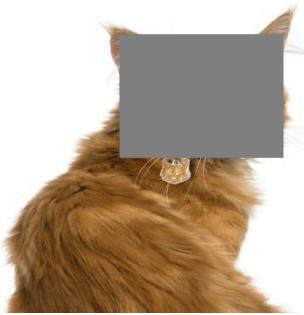
# Saliency maps via image occlusions



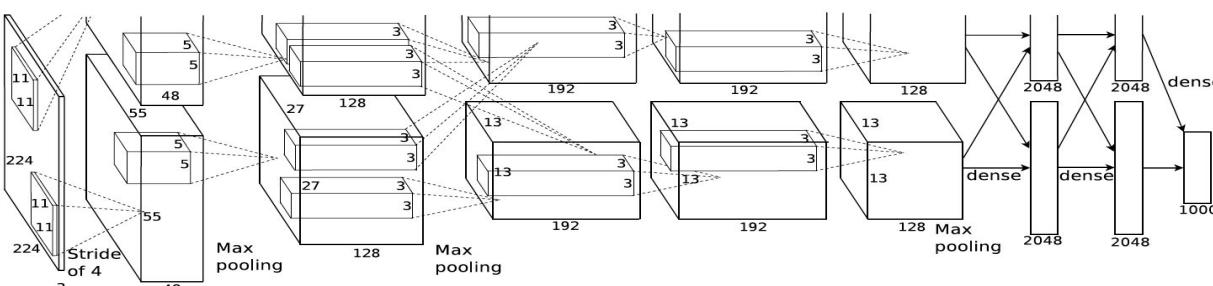
**Which are the important parts of an image?**

Slide an occluding patch over the input image and display (as a heat map) the probability of the correct class

# Saliency maps via image occlusions

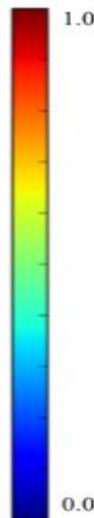
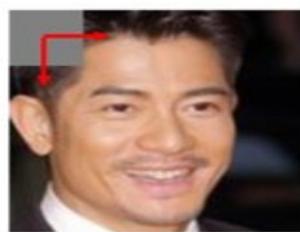


$P(\text{cat}) = 0.4$

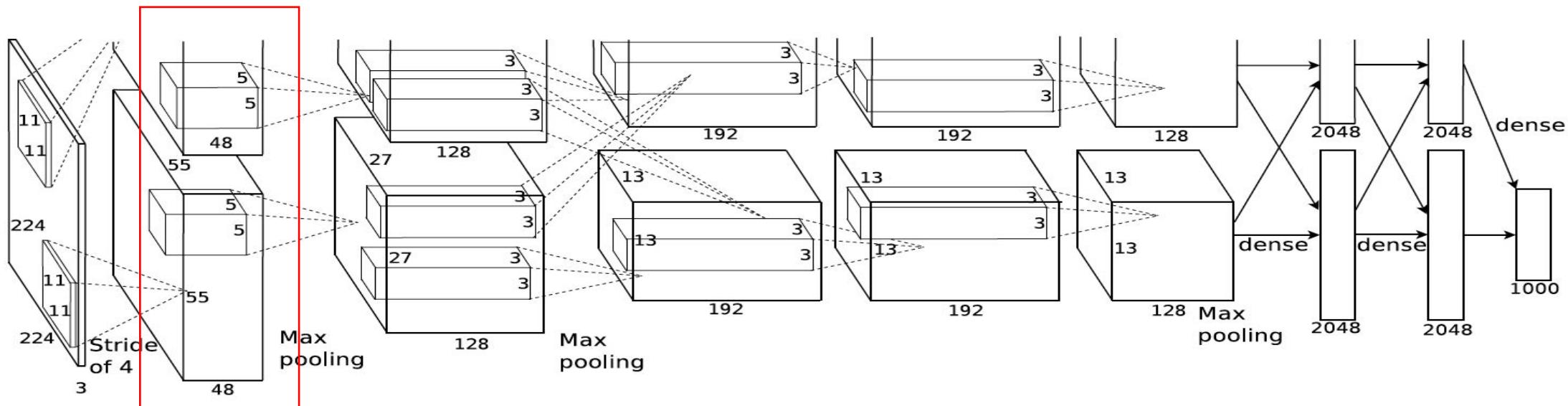


$P(\text{cat}) = 0.96$

# Saliency maps via image occlusions



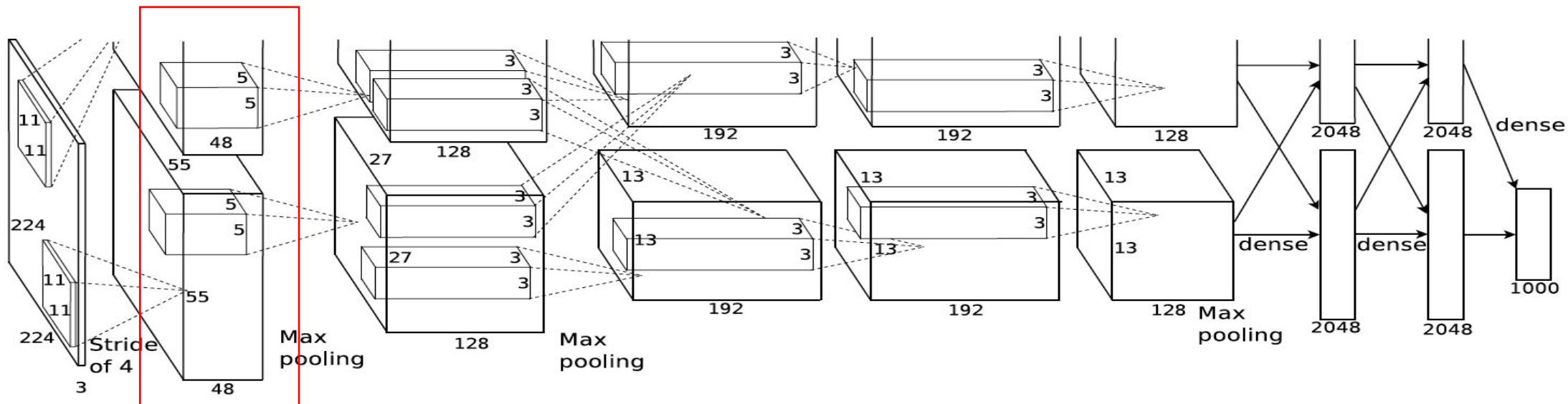
# Filter visualization – 1<sup>st</sup> layer



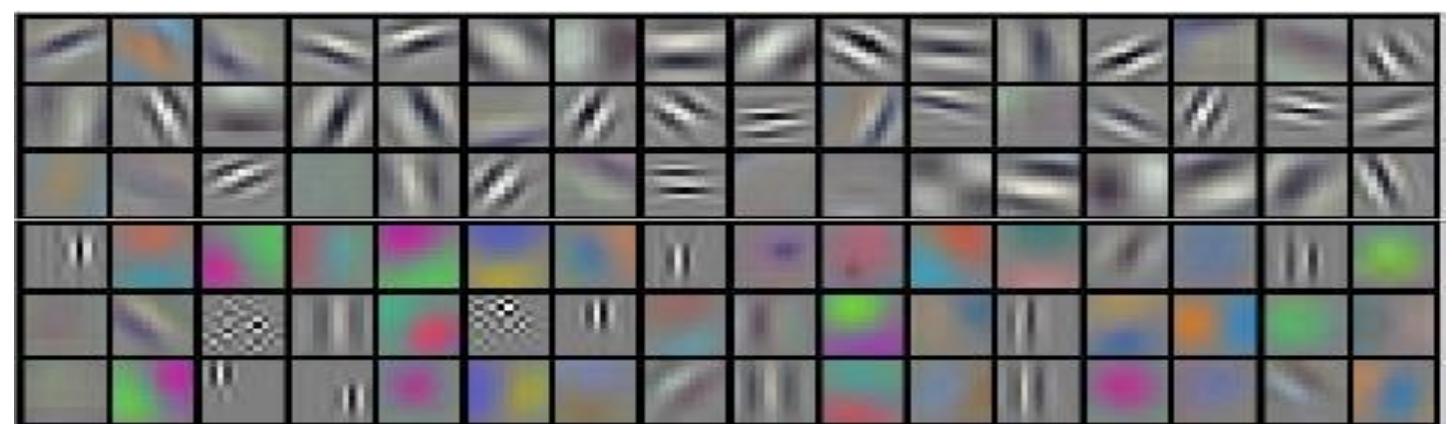
First layer operates directly on image pixels; visualize the filters used to extract the image features

Visualize this filter bank

# Filter visualization – 1<sup>st</sup> layer



First layer operates directly on image pixels; visualize the filters used to extract the image features



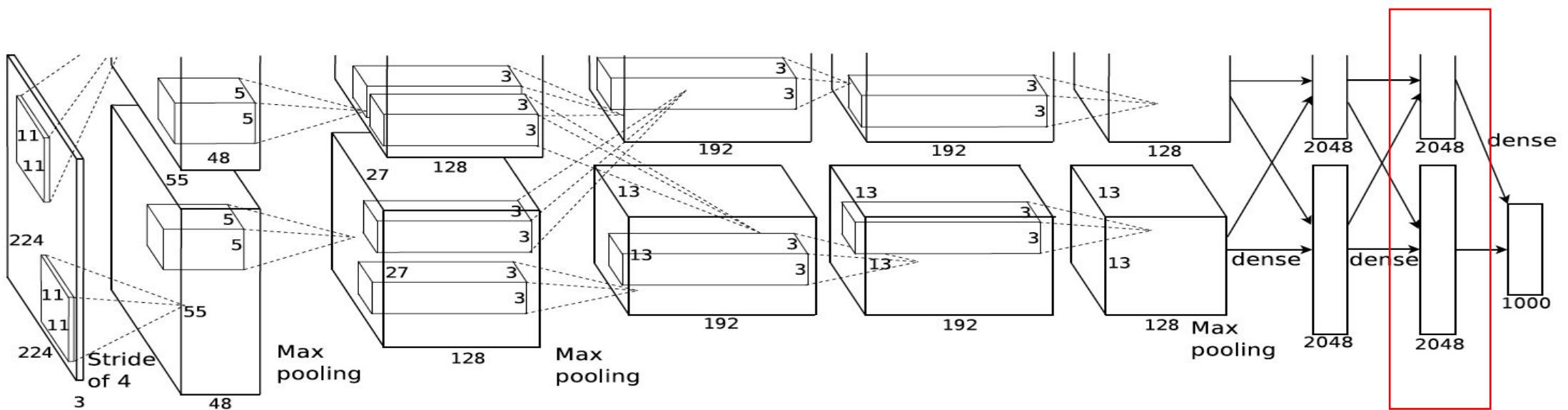
# Filter visualization – deeper layers

- Not that easy to interpret



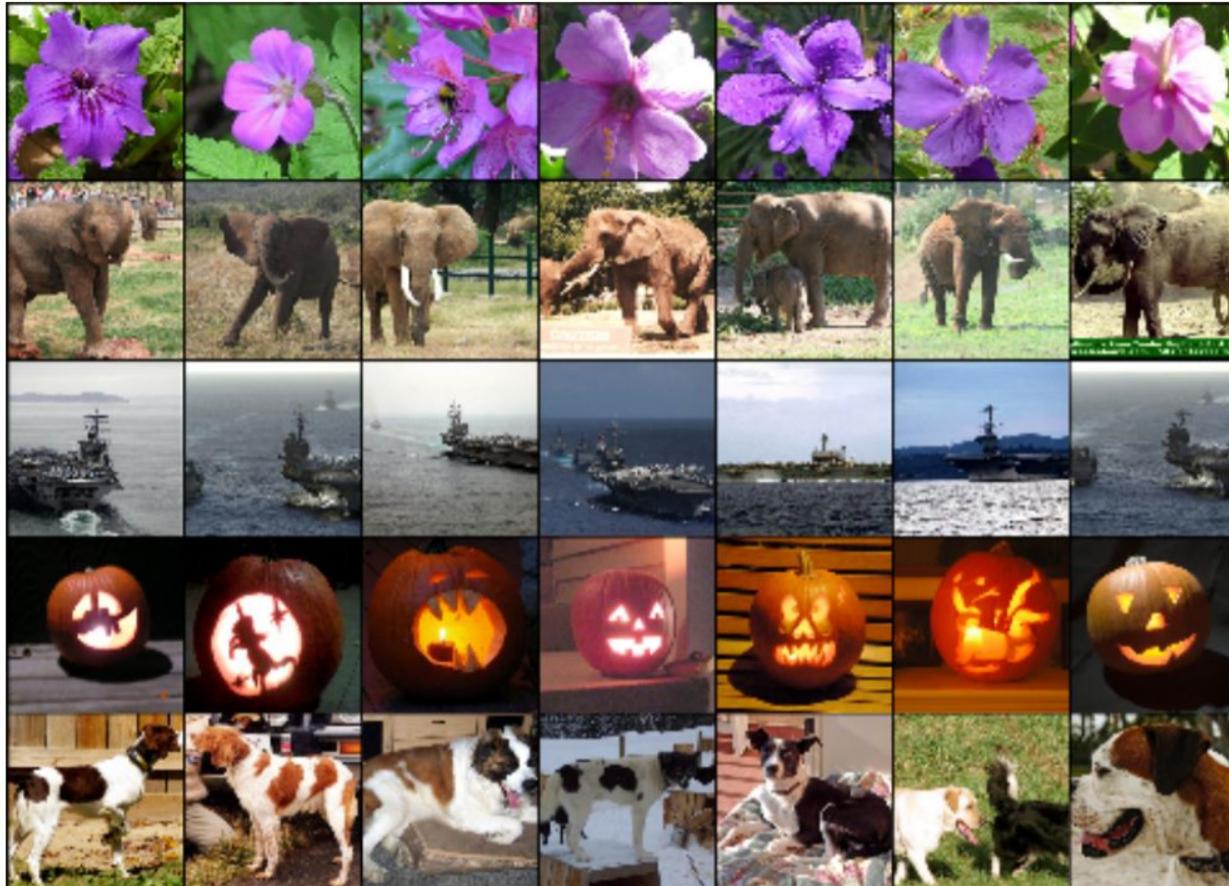
# Last layer visualization

- The last layer (before the classification layer) contains the most condensed representation of the image



# Last layer visualization

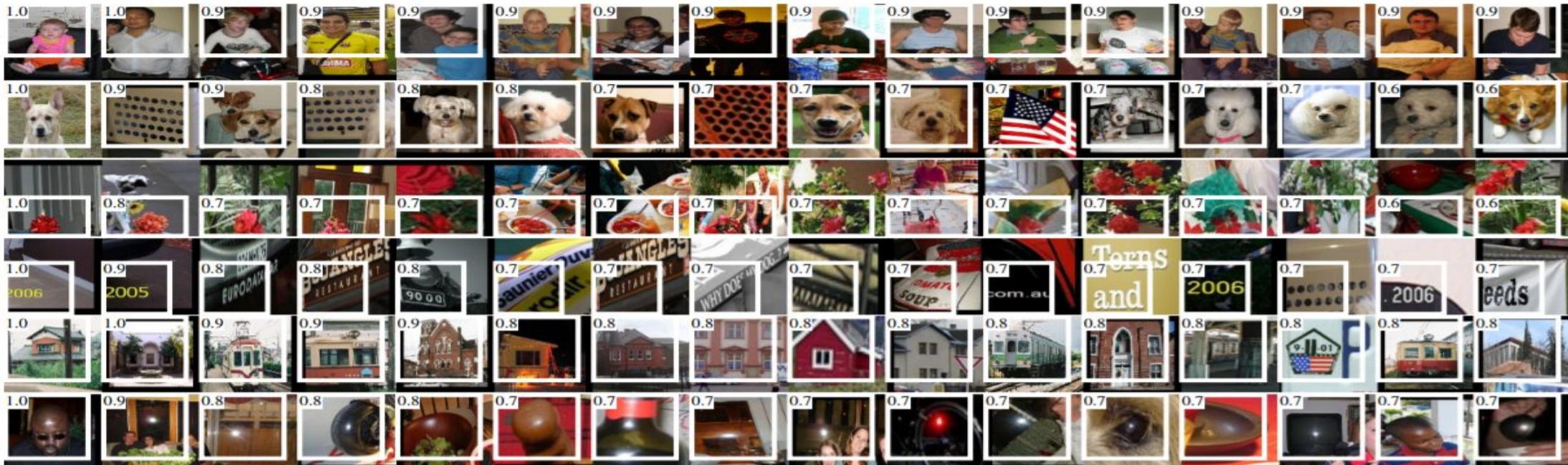
## Image embedding



Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image

# Visualize the patch that maximally activates a neuron

- Single out a particular unit (feature) in the network and use it as if it were an object detector in its own right
- compute the unit's activations on a large set of held-out region, sort the proposals from highest to lowest activation, perform non-maximum suppression, and then display the top-scoring region



**Figure 4: Top regions for six pool<sub>5</sub> units.** Receptive fields and activation values are drawn in white. Some units are aligned to concepts, such as people (row 1) or text (4). Other units capture texture and material properties, such as dot arrays (2) and specular reflections (6). <https://arxiv.org/pdf/1311.2524.pdf>

# Activation maps

Deep visualization toolbox #deepvis

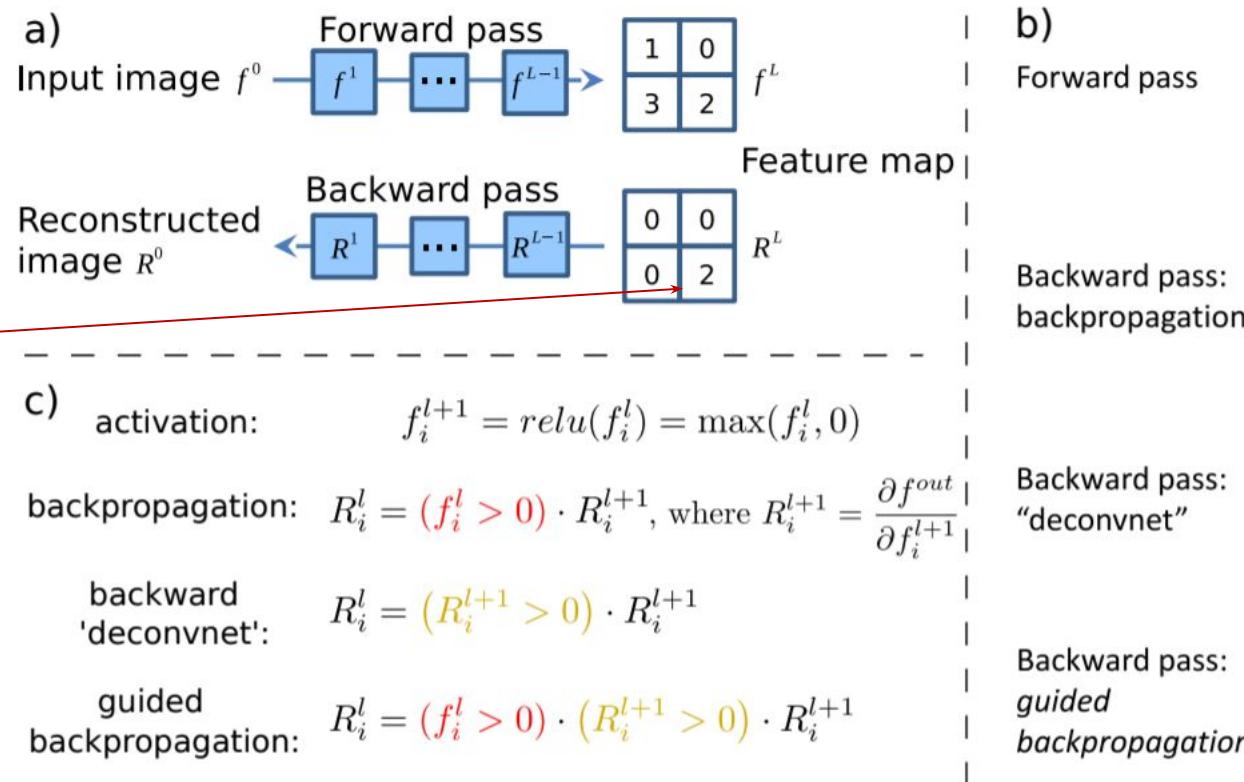
<https://www.youtube.com/watch?v=AgkfIQ4IGaM>

# Activation maps

- Compute the gradient of a given neuron in the network with respect to the input image
  - Select the layer, set the gradient at that level to be all zeros except for the neuron of interest (where we set the gradient at 1)
  - Apply back-propagation to the image

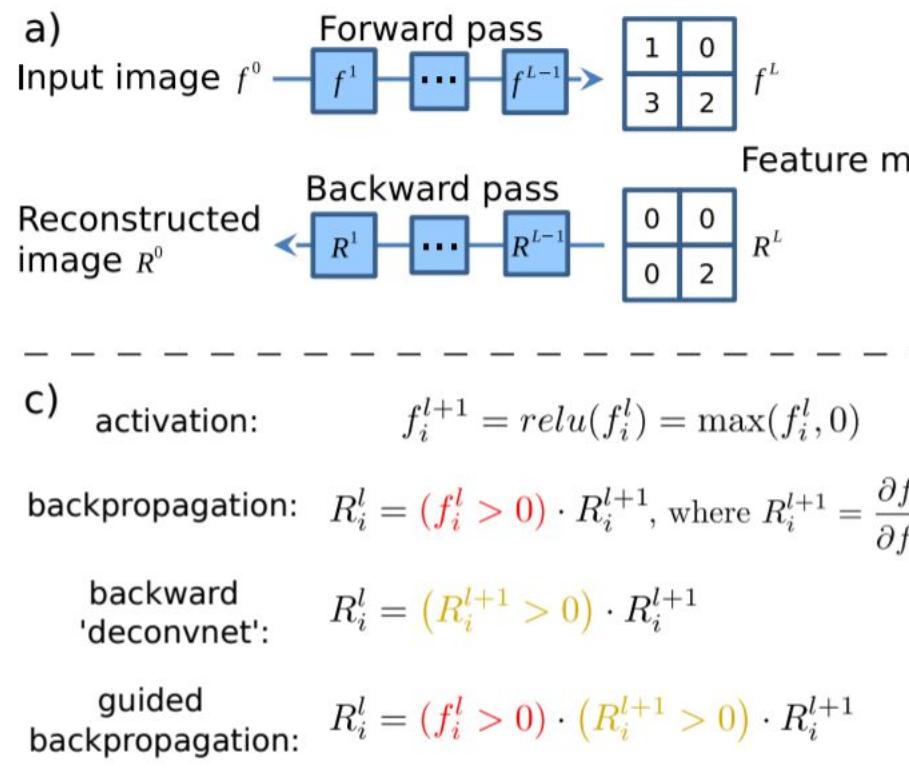
# Activation maps

Zero out all the gradients except for the neuron of interest



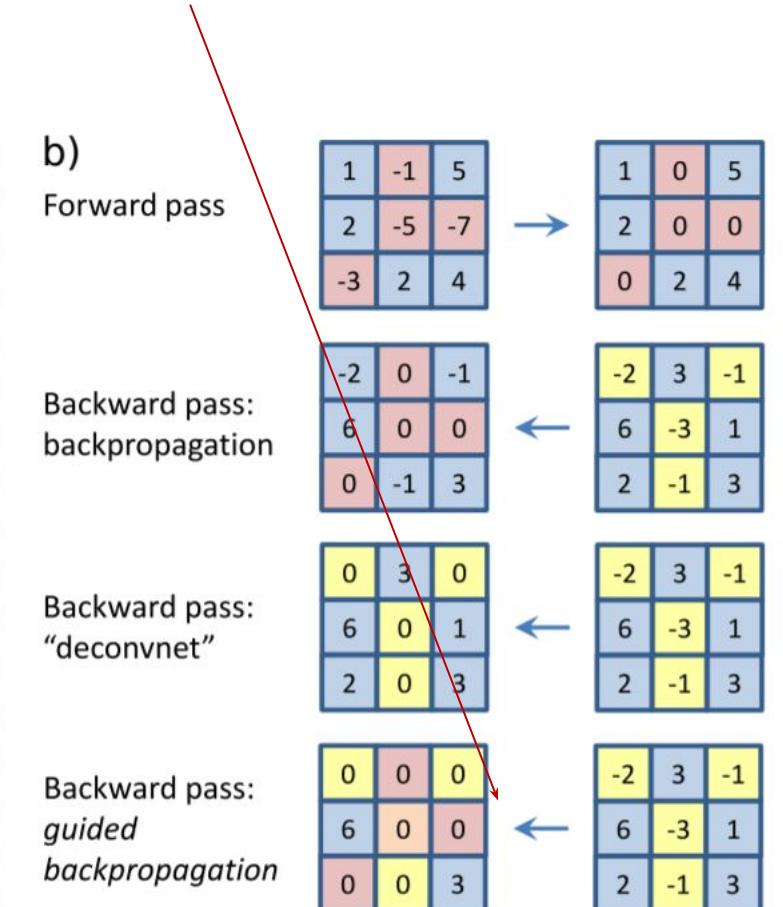
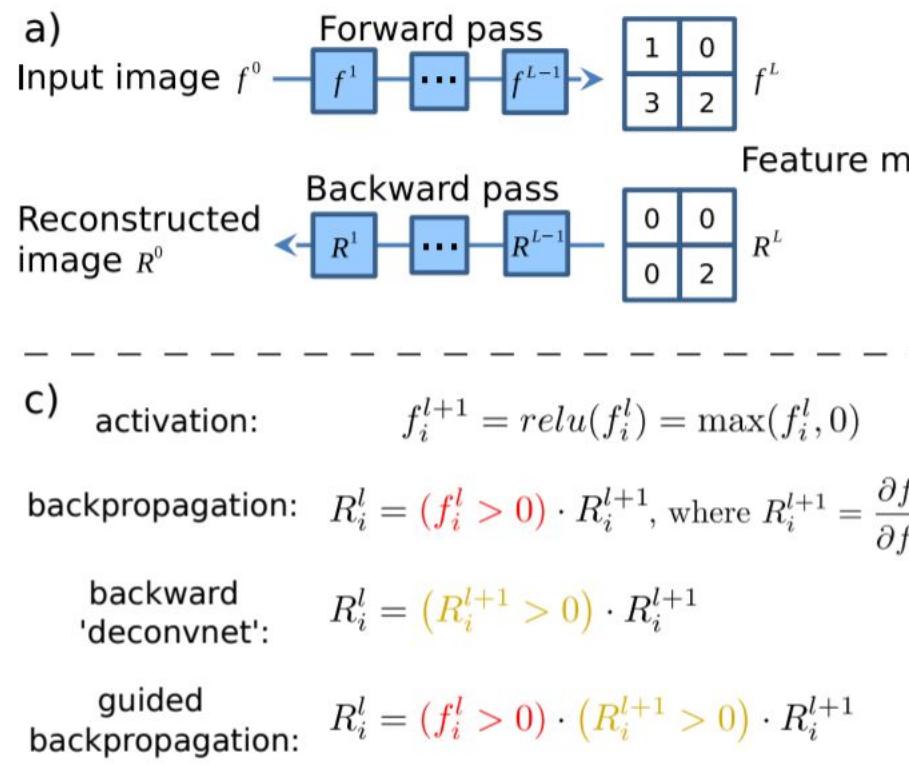
# Activation maps

ReLU during backward pass:  
block the gradients for the  
neurons that had  
activations < 0



# Activation maps

Guided back-propagation: backprop only the parts that have a positive gradient



## Paths of influence

Pass only gradients that have a positive influence. Keep only the positive influence

# Activation maps

guided backpropagation



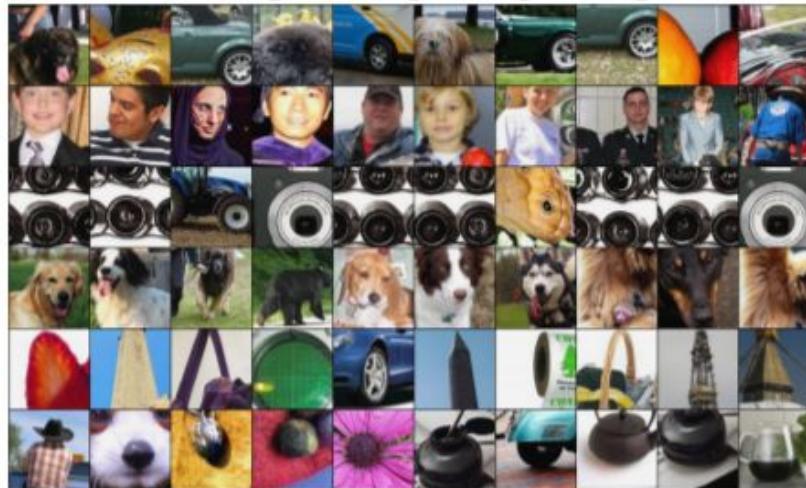
corresponding image crops



guided backpropagation



corresponding image crops



Visualization of patterns learned by the layer conv6 (top) and layer conv9 (bottom) of the network trained on ImageNet.

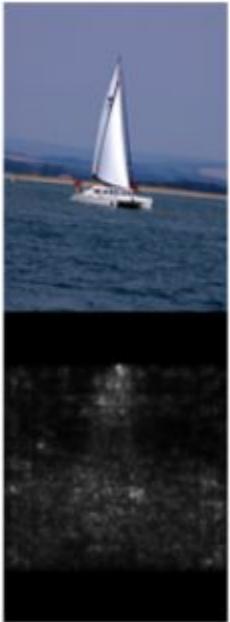
# Saliency maps – visualize the data gradient

Important image pixels

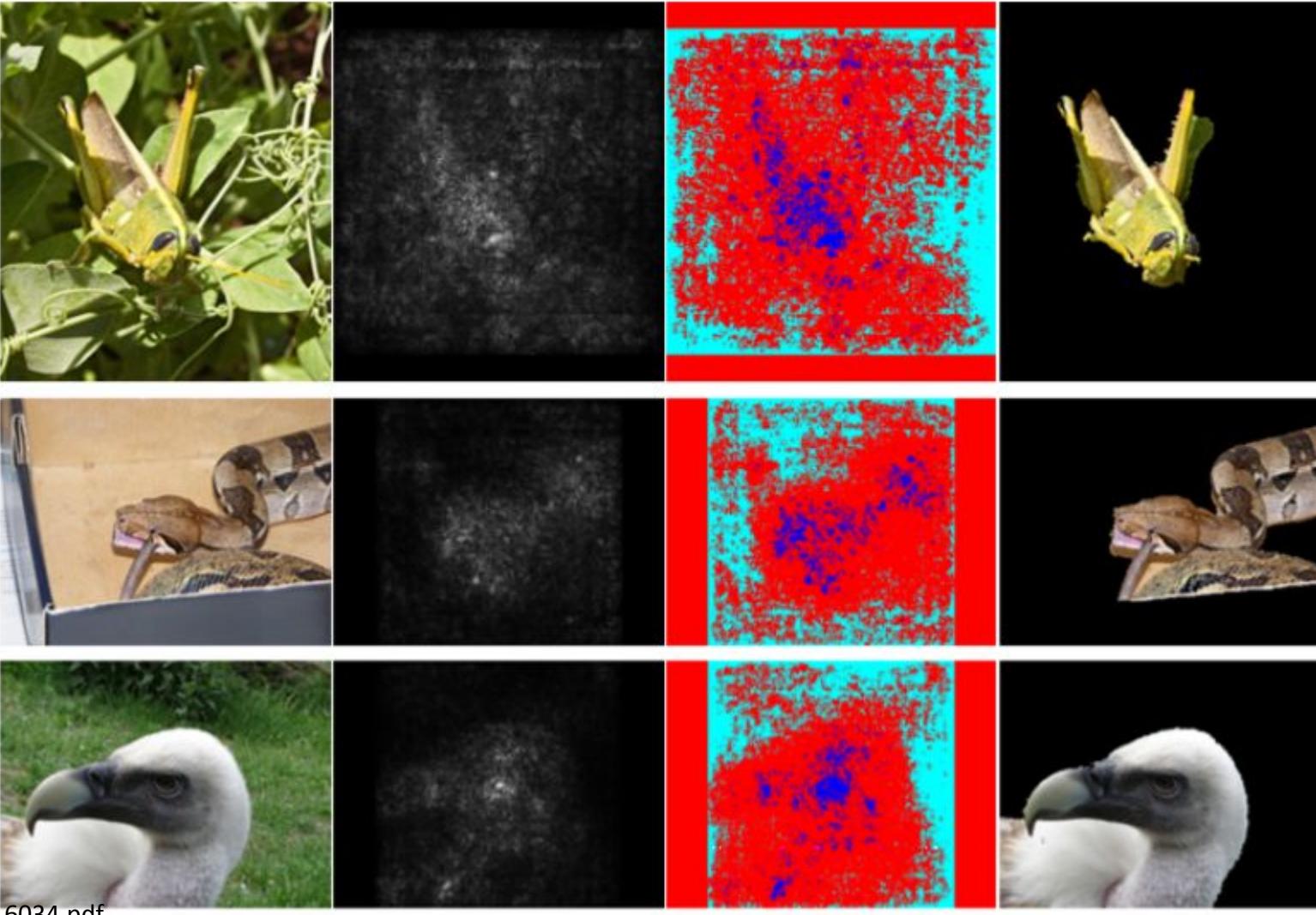
- Heat map of the gradients
- Compute gradient of the (un-normalized) class score with respect to image pixels and then take absolute value and max over RGB channels
- Strength of influence of each pixel on the class score

# Saliency maps – visualize the data gradient

Important image pixels

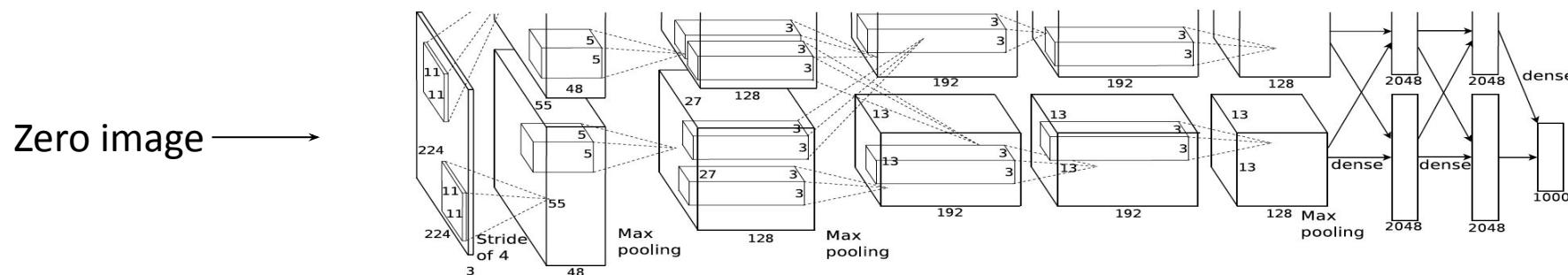


# Guided image segmentation



# Optimize the image

- Keep the parameters of the network fixed and optimize the input image for some class score



Forward pass

Set the gradient of the scores vector to be all zeros except for the class of interest and then back-prop to the image  
[0, 0, ..., 0, 1, ..., 0]

Image update

Forward image to the networks

repeat

More formally, let  $S_c(I)$  be the score of the class  $c$ , computed by the classification layer of the ConvNet for an image  $I$ . We would like to find an  $L_2$ -regularised image, such that the score  $S_c$  is high:

$$\arg \max_I S_c(I) - \lambda \|I\|_2^2, \quad (1)$$

# Optimize the image



dumbbell



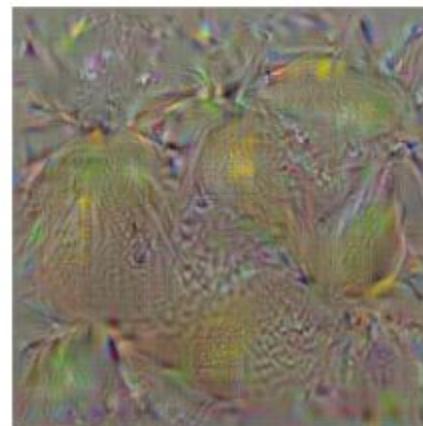
cup



dalmatian



bell pepper

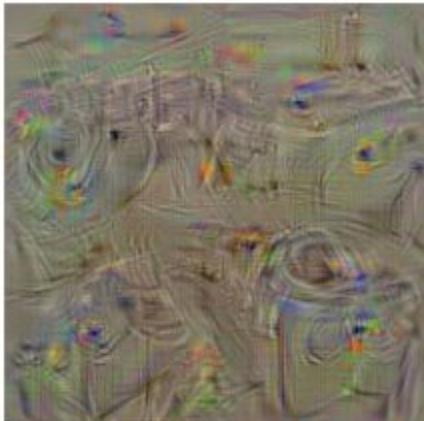


lemon

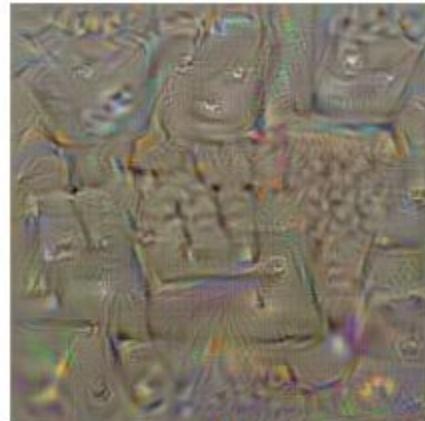


husky

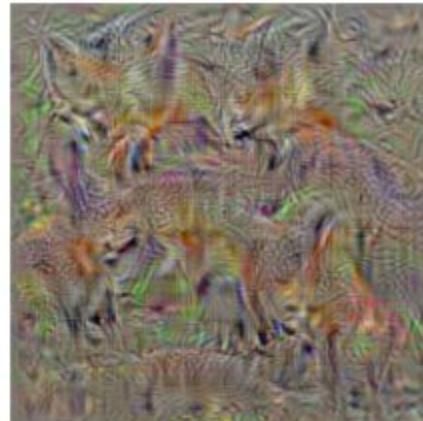
# Optimize the image



washing machine



computer keyboard



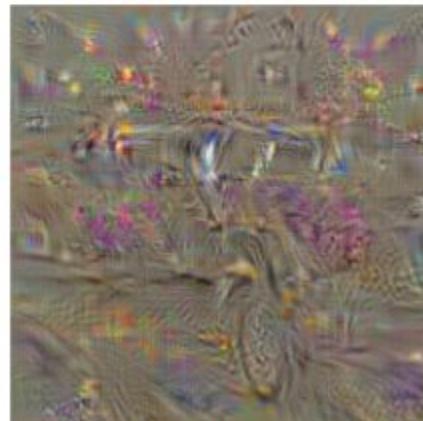
kit fox



goose



ostrich



limousine

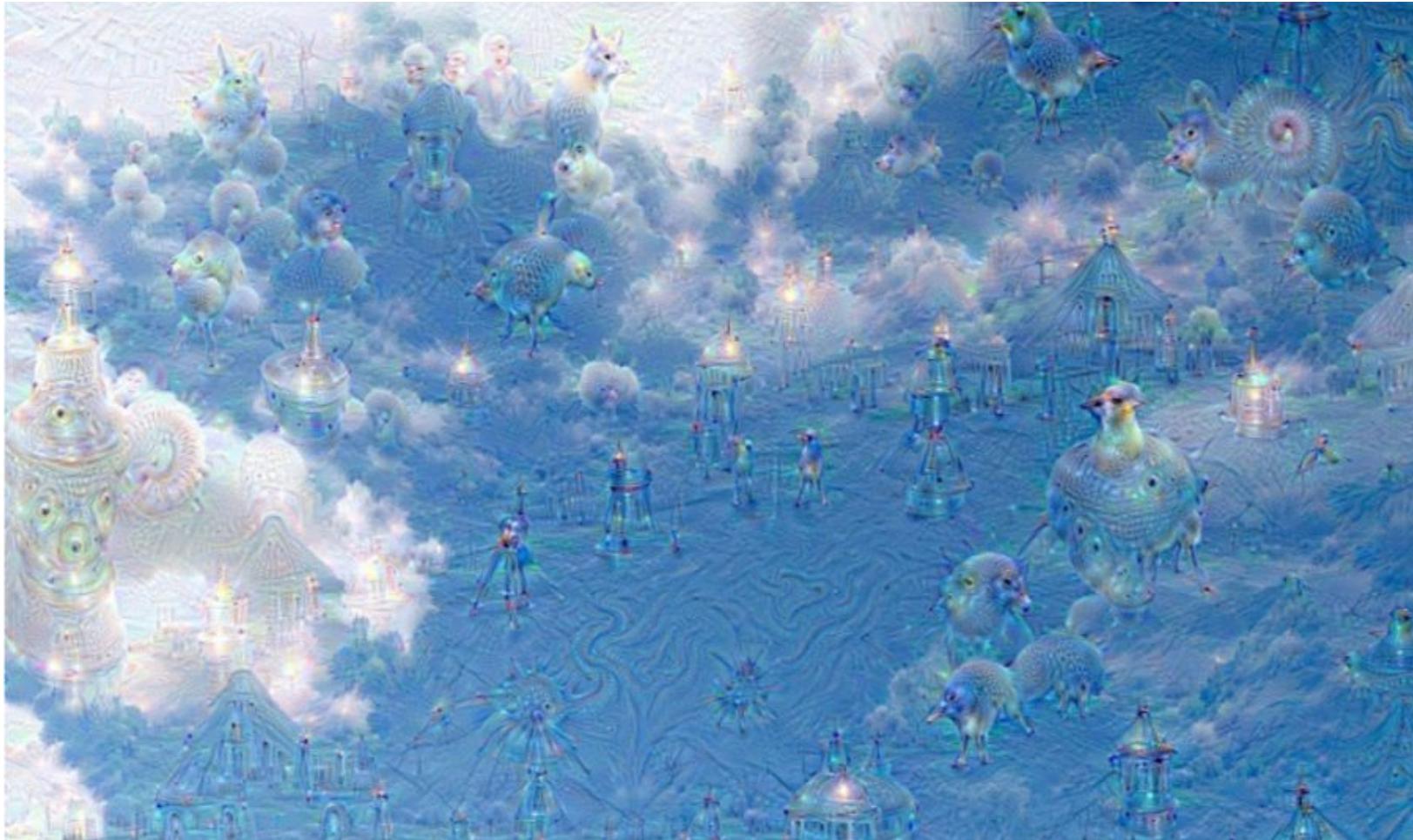
# Deep dream

- “Dream” at a given layer of the network (after the ReLu units)
  - Forward pass to the give image
  - Set the gradients to be equal to the activation
  - Backprop to the image
- Modify the image such that it amplifies the activations at any given layer

<https://github.com/google/deepdream/blob/master/dream.ipynb>

# Deep dream

Whatever was activates gets boosted



# Deep dream



"Admiral Dog!"



"The Pig-Snail"

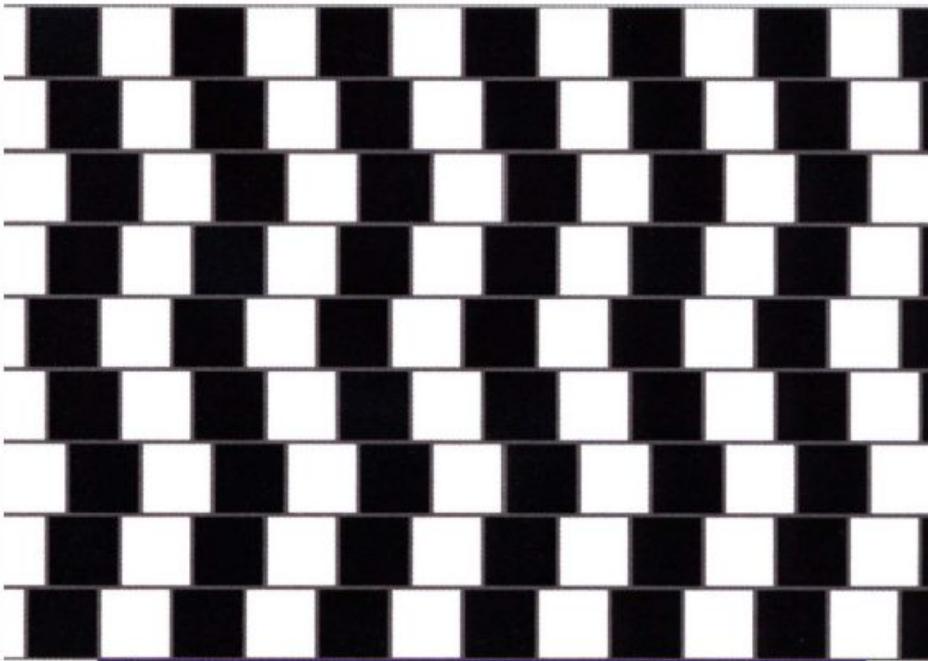


"The Camel-Bird"



"The Dog-Fish"

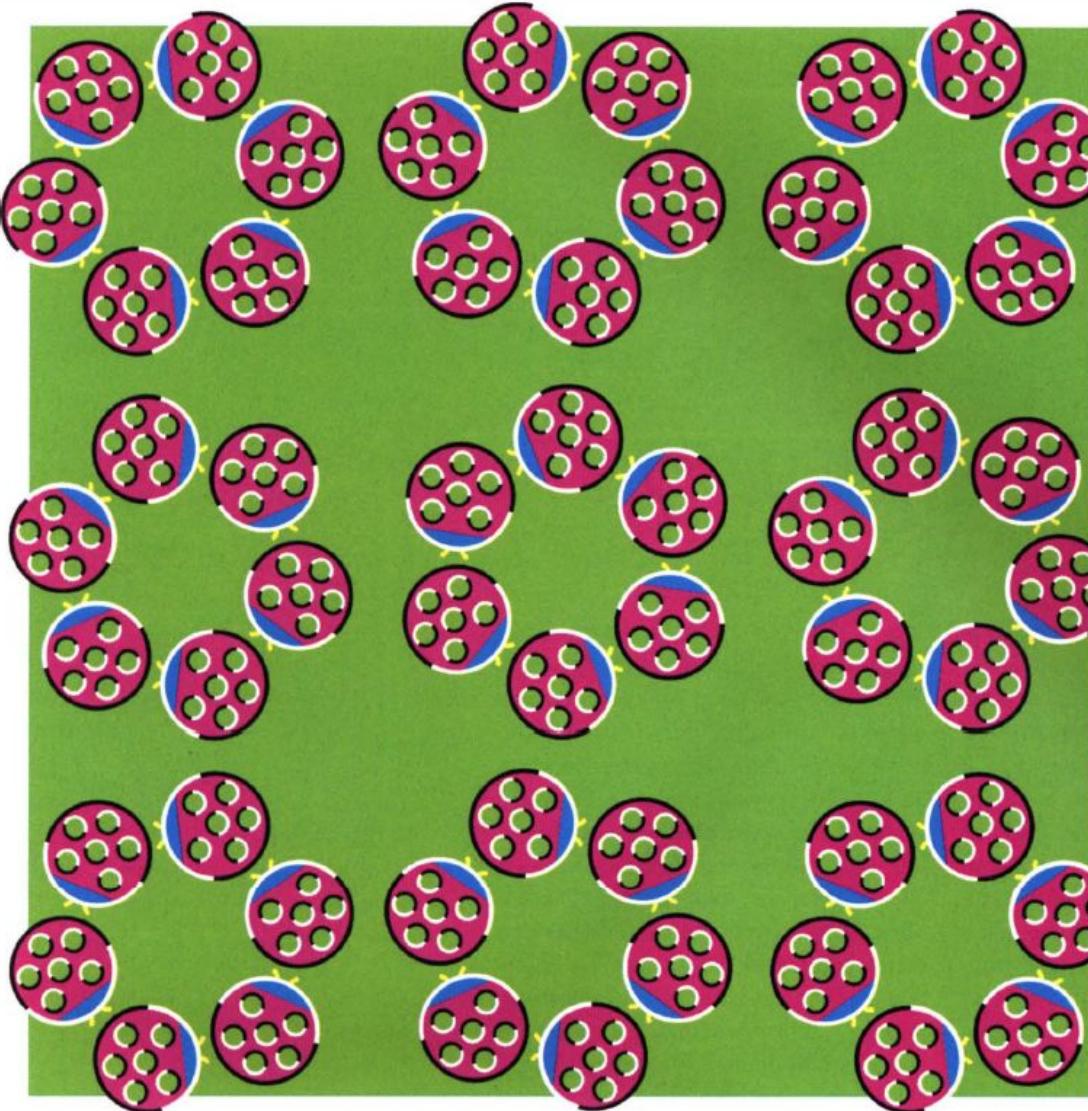
# Adversarial examples



## Test des rayures

**Illusion:** les rangées de briques apparaissent en pente alors qu'elles sont pourtant parfaitement parallèles !

**Explication:** le contraste entre les carrés blancs et noirs avec le léger décalage des carrés noirs d'une rangée à l'autre et la finesse des lignes grises créeraient l'illusion.



## Test des coccinelles

**Illusion:** si on fixe les coccinelles, d'autant plus sur le bord de cette image, elles se mettent à tourner.

**Explication:** cet effet découle de l'excitation des neurones sensibles au mouvement, majoritaires dans la vision périphérique. Il serait en partie dû au contraste entre des couleurs claires à l'avant de la coccinelle et des couleurs sombres à l'arrière, car ces informations ne parviennent pas simultanément aux aires visuelles. Le cerveau interpréterait alors le mouvement comme un déroulement temporel. L'effet est accentué par les "microsaccades" naturelles des yeux.

# “Adversarial” examples for the human brain

#thedress

#whiteandgold

#blackandblue





# Adversarial examples

An adversarial example is an instance with small, intentional feature perturbations that cause a machine learning model (**!not only CNNs!**) to make a false prediction.



$\mathbf{x}$   
“panda”  
57.7% confidence

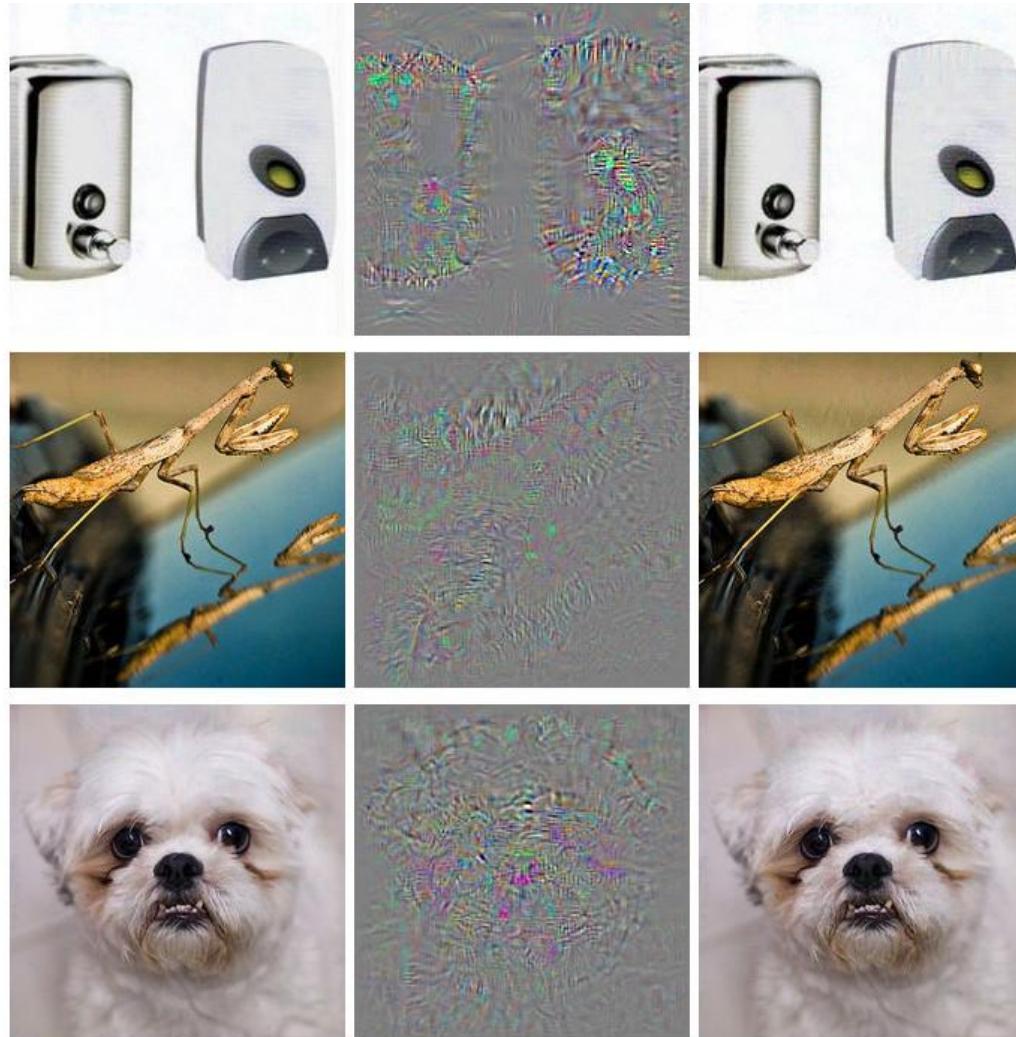


$+ .007 \times$   
 $\text{sign}(\nabla_{\mathbf{x}} J(\theta, \mathbf{x}, y))$   
“nematode”  
8.2% confidence

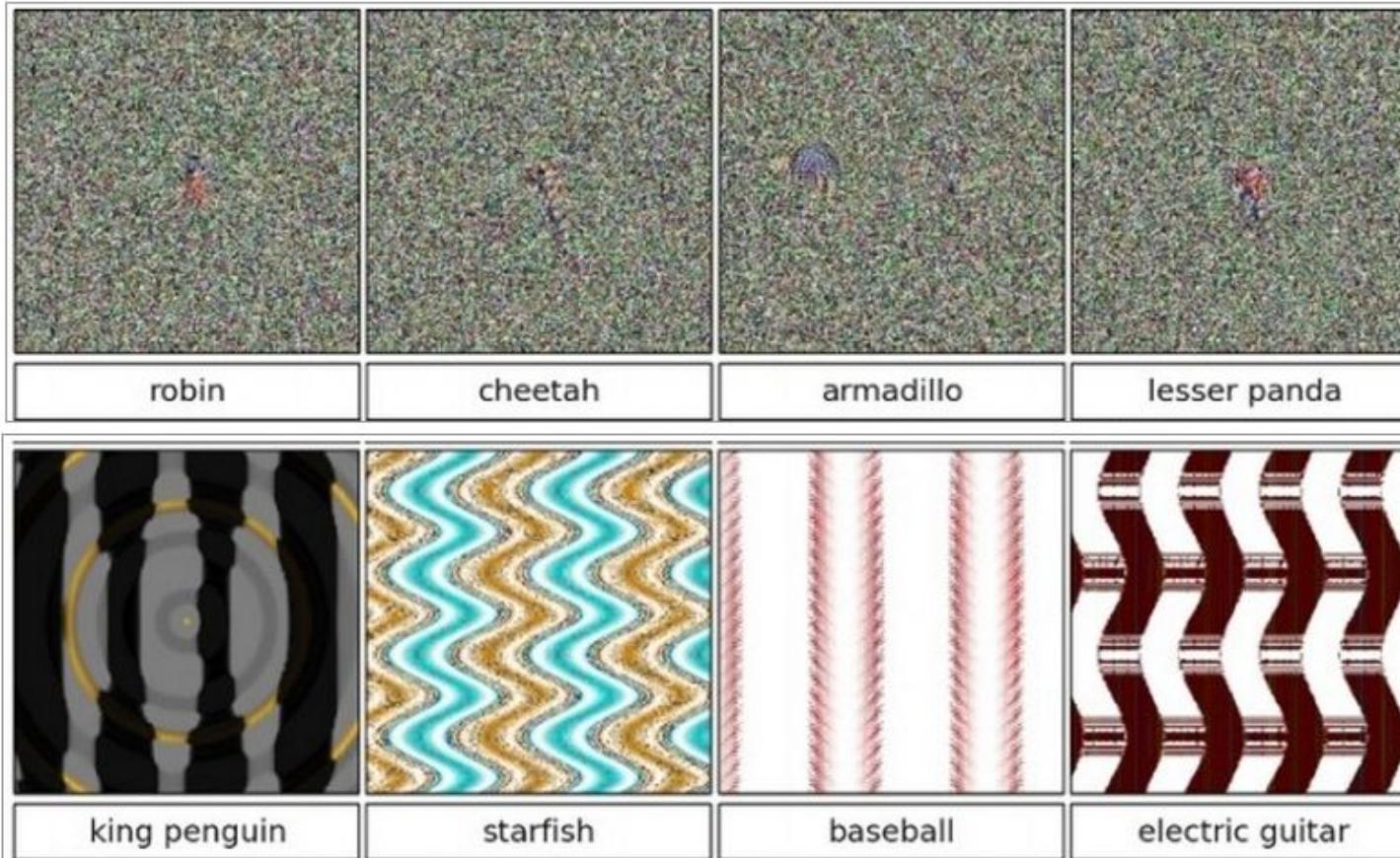


$=$   
 $\mathbf{x} +$   
 $\epsilon \text{sign}(\nabla_{\mathbf{x}} J(\theta, \mathbf{x}, y))$   
“gibbon”  
99.3 % confidence

# Make everything an ostrich



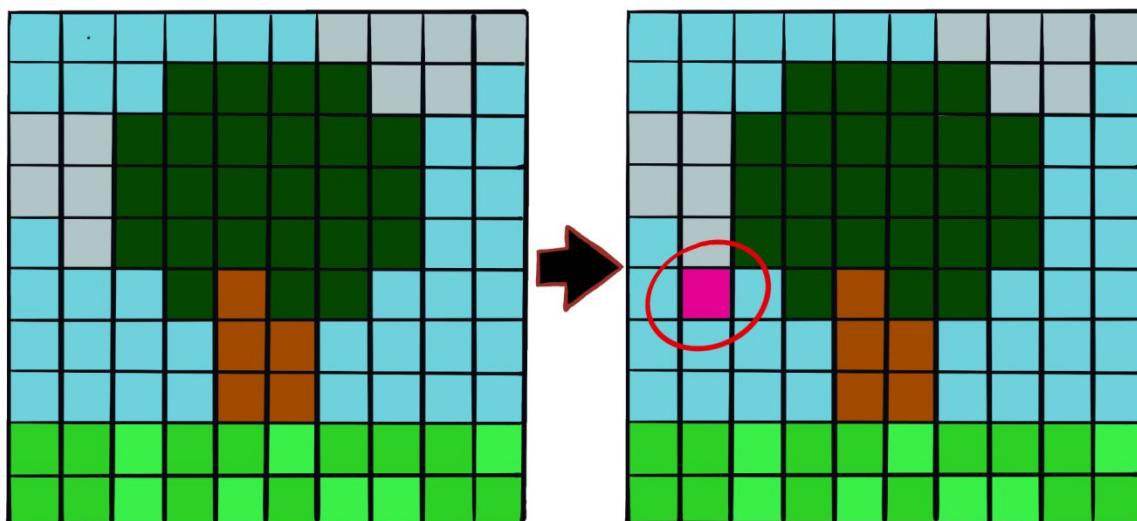
# Classifying noise



Nguyen, Anh, Jason Yosinski, and Jeff Clune. "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.

# 1-pixel attacks

Constraint: when designing the adversarial example only one pixel may change



Cup(16.48%)  
Soup Bowl(16.74%)



Bassinet(16.59%)  
Paper Towel(16.21%)



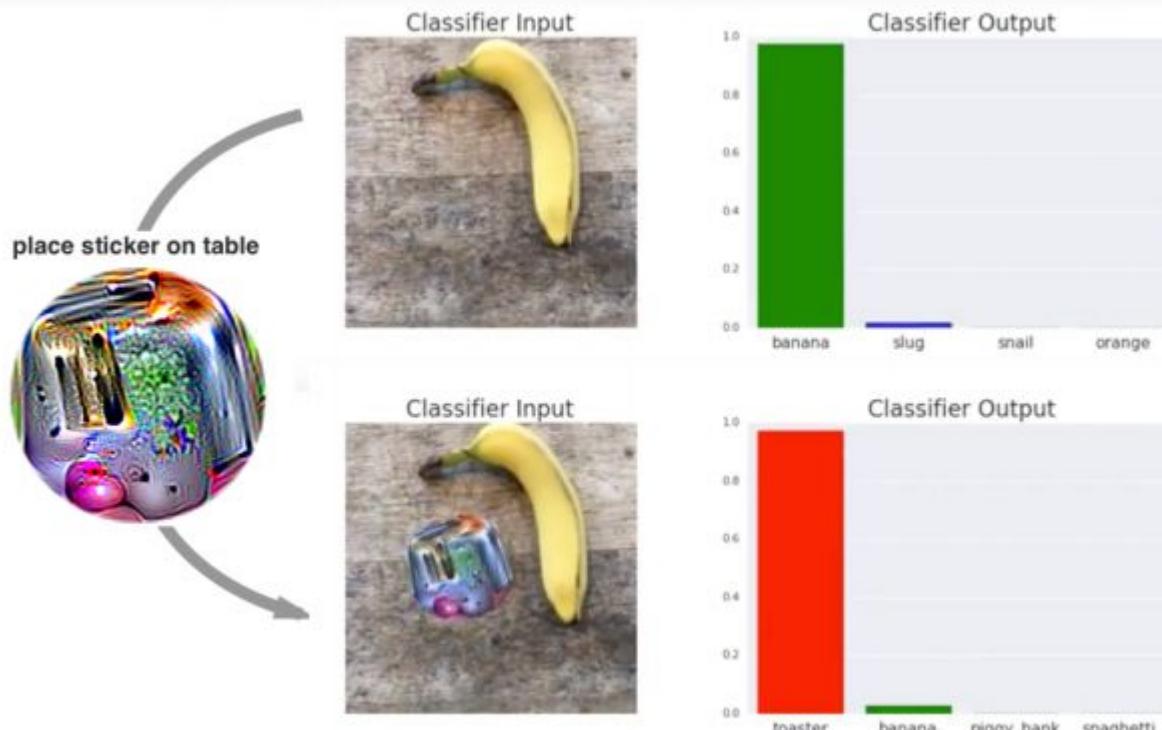
Teapot(24.99%)  
Joystick(37.39%)



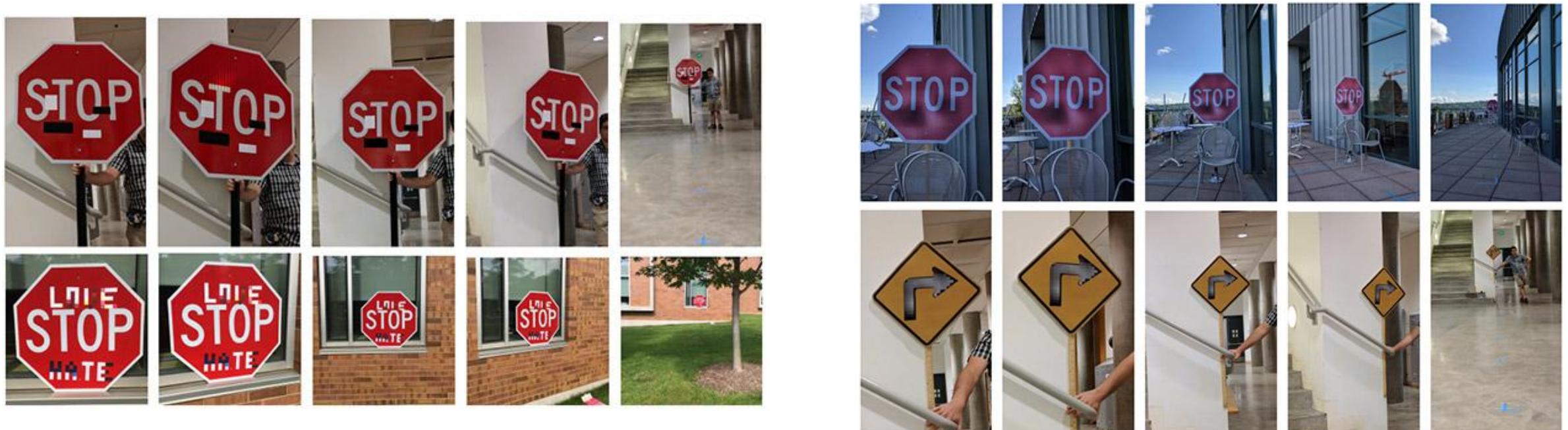
Hamster(35.79%)  
Nipple(42.36%)

# Adversarial patch

Make everything a toaster



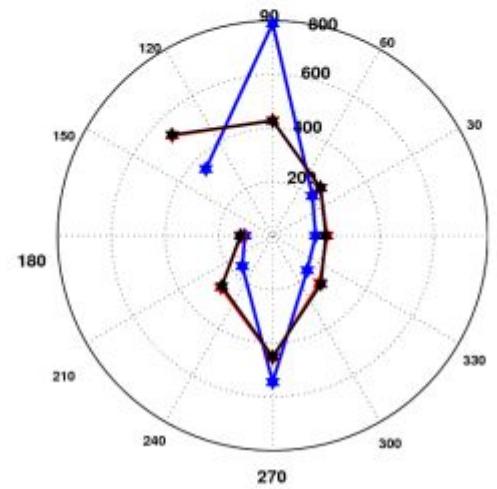
# Slight alterations in the physical world



# Adversarial examples

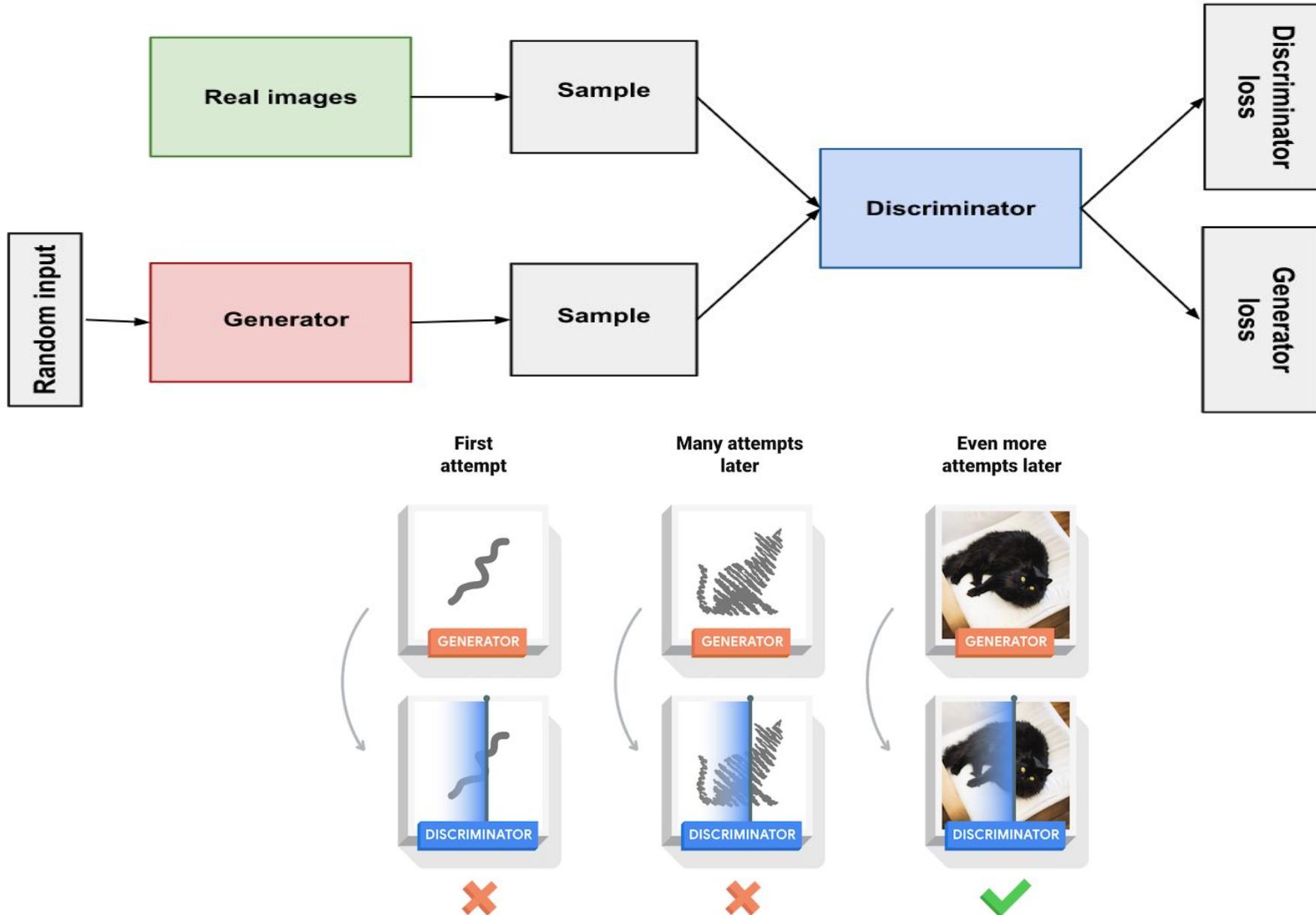
occur not only for CNNs but also for handcrafted features

HOG descriptor



# Generative adversarial networks

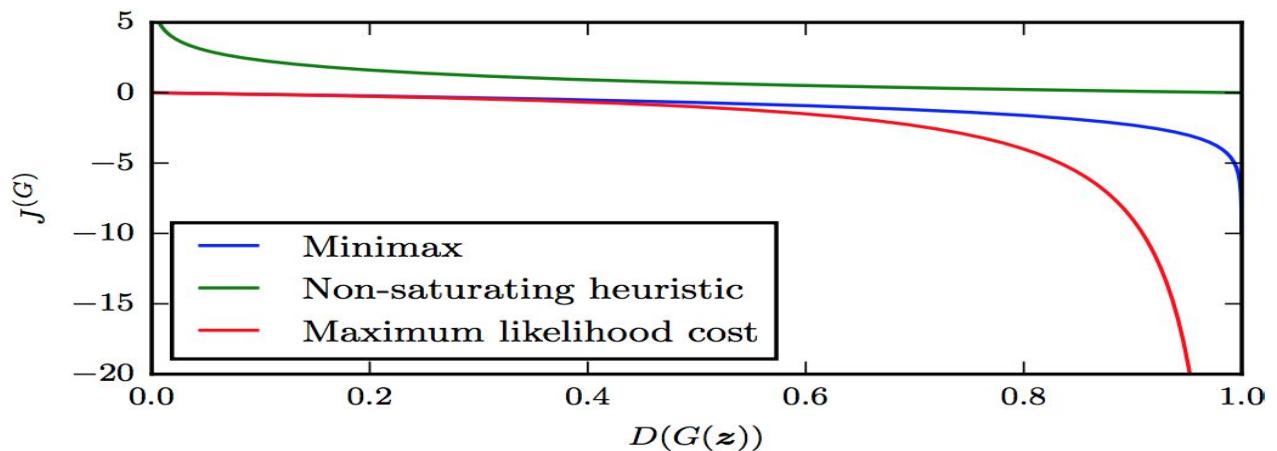
# GANs



# GANs

Training is hard – **Vanishing gradients**

- If the discriminator is too good, the generator training can fail due to vanishing gradients
- A perfect discriminator won't allow the generator to converge

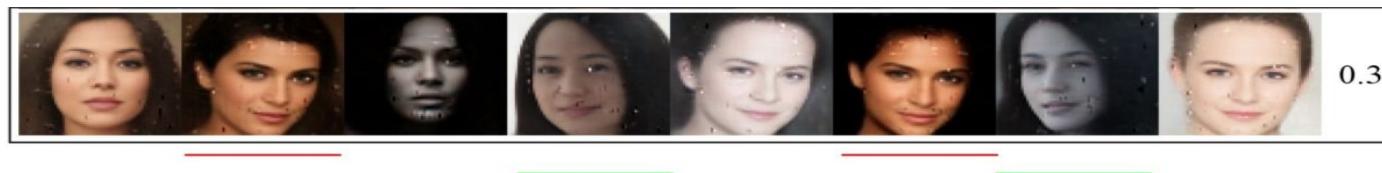
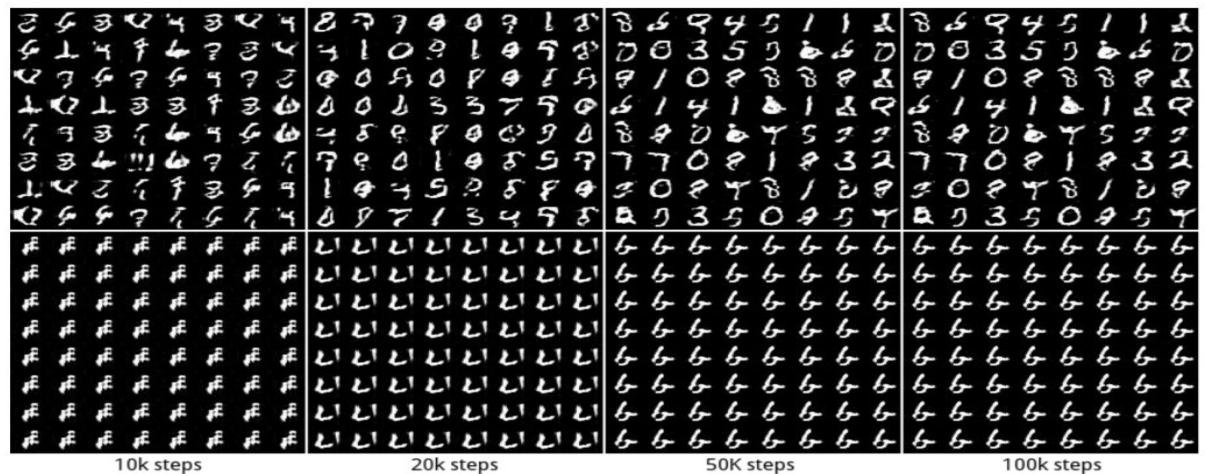


# GANs

Training is hard – **Mode collapse**

- Generator starts producing a single output (that is most common to fool the discriminator)

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log \left( 1 - D \left( G \left( z^{(i)} \right) \right) \right)$$



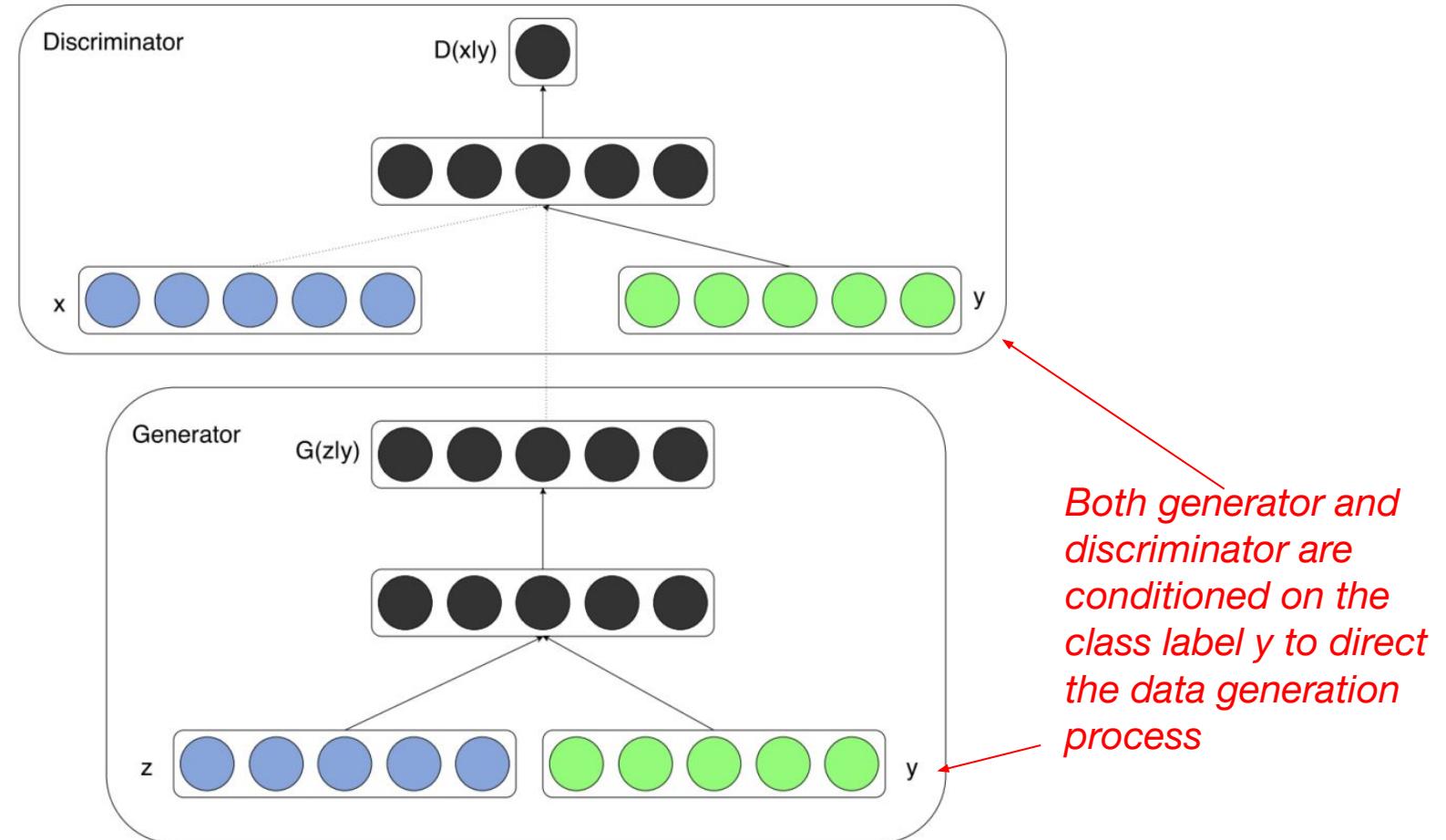
# GANs

Training is hard – **Mode collapse**

- Discriminator learns to always reject the same output generated by the generator; local minimum
- In the generation, the generator finds and generates the sample that is most plausible for the current generator; it's a trap!

# Conditional Generative Adversarial Networks (cGAN), 2014

- Use the class label info when training the GAN
  - Generate targeted images



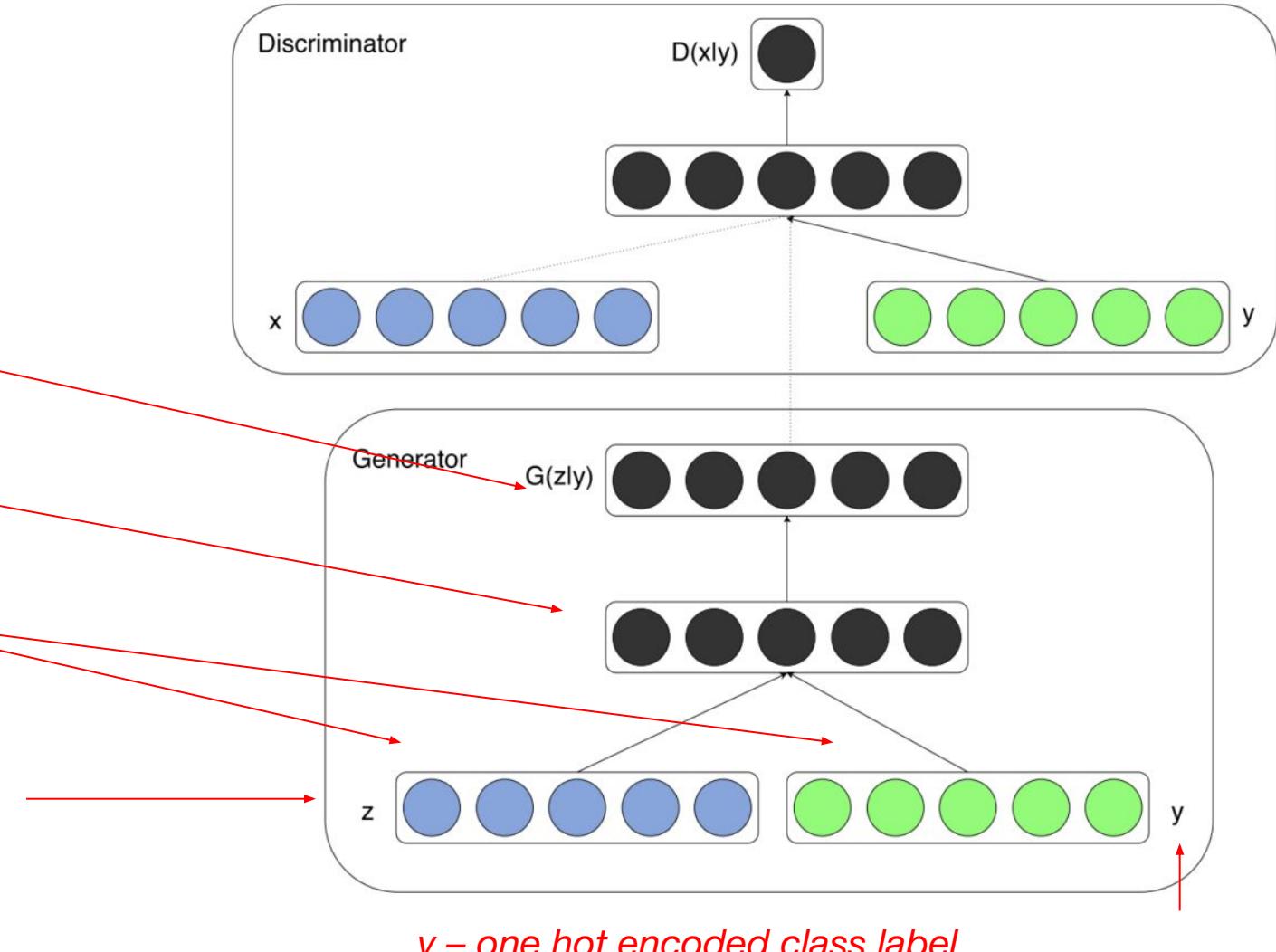
# Conditional Generative Adversarial Networks (cGAN), 2014

**Finally:** final sigmoid unit layer as our output for generating the 784-dimensional MNIST samples.

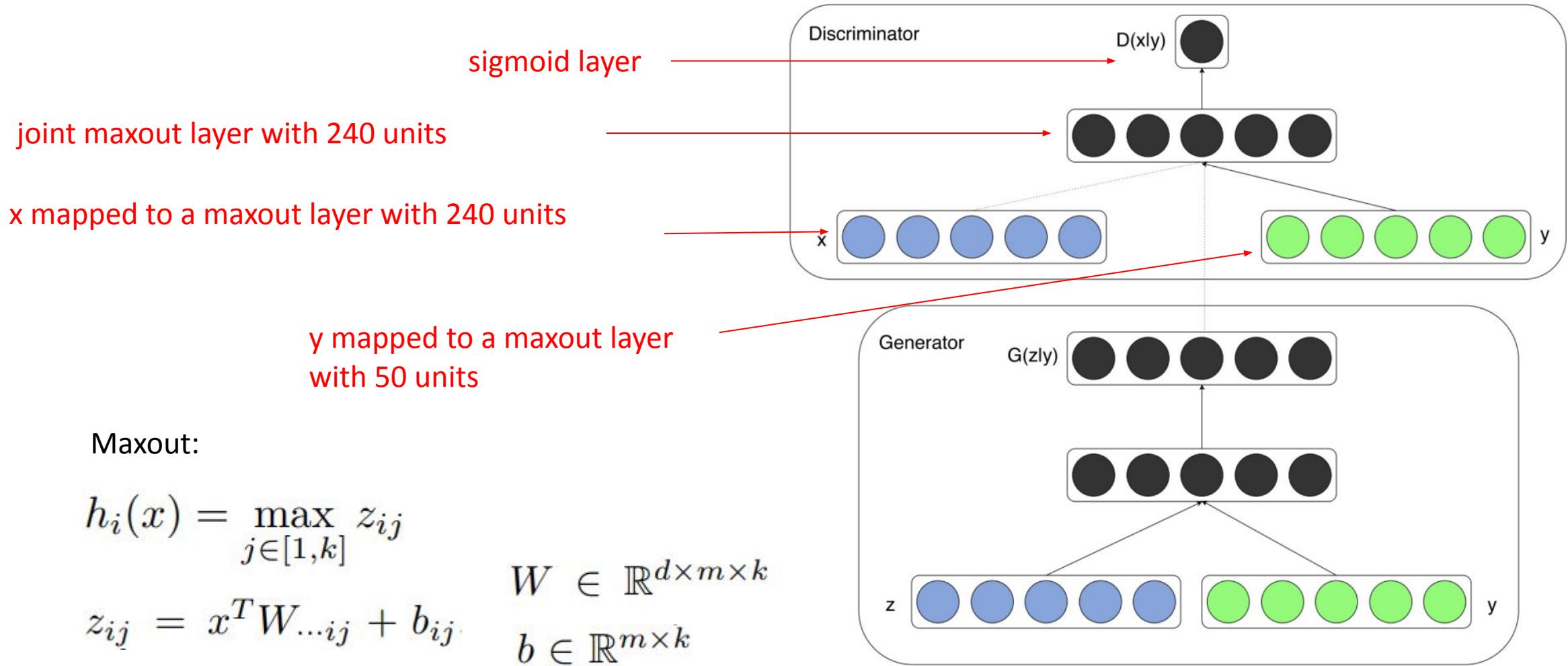
*combined to a hidden ReLu layer of dimensionality 1200*

*hidden layers with ReLu activation with layer sizes 200 and 1000*

*z - noise prior (100 D) drawn from a uniform distribution within the unit hypercube*



# Conditional Generative Adversarial Networks (cGAN), 2014



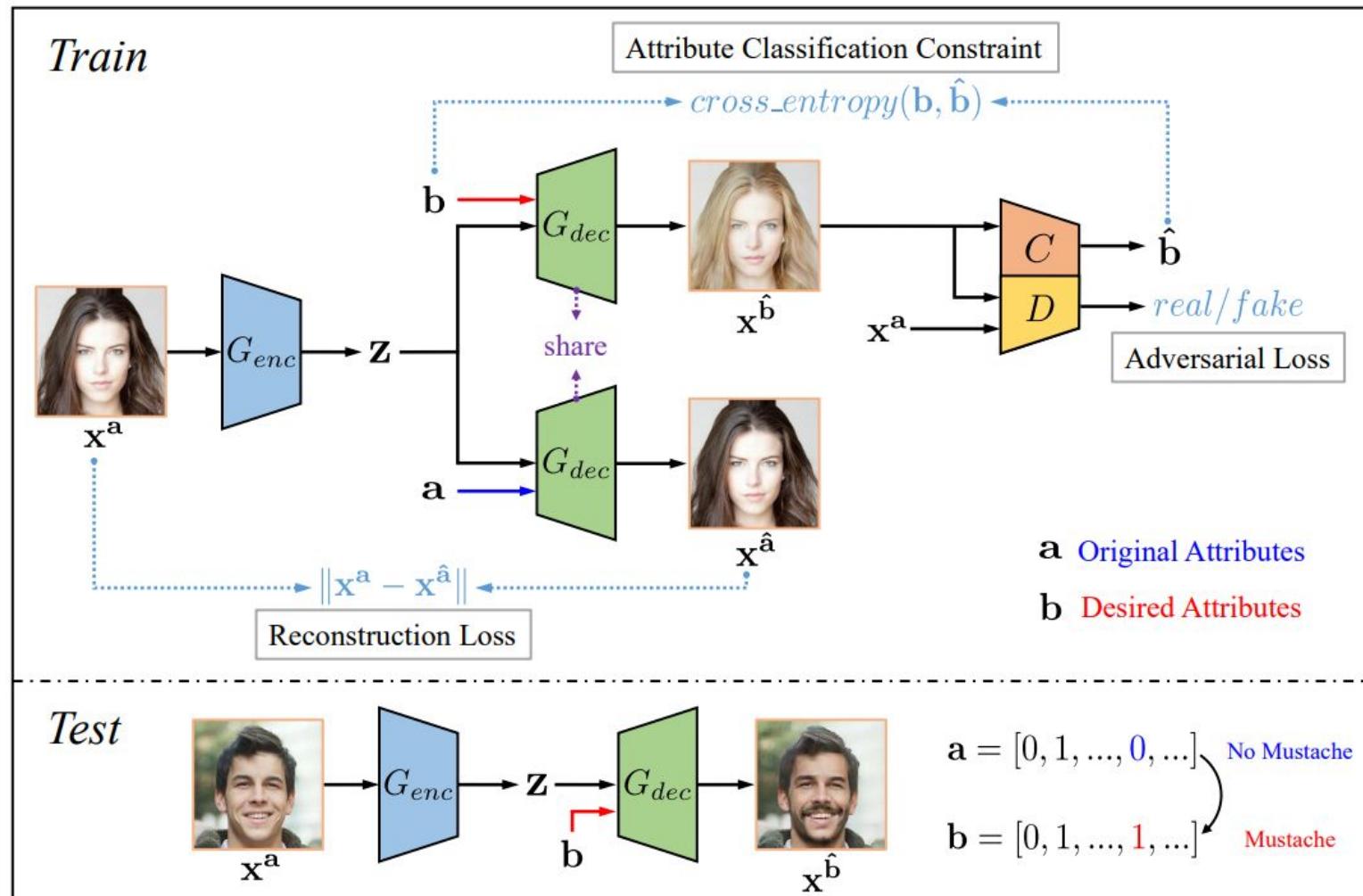
# Conditional Generative Adversarial Networks (cGAN), 2014



# Adding discrete variables in Conditional GANs

- Use an Embedding layer to encode the discrete variables
- Add as additional channels to images
- Keep embedding dimensionality low and up-sample to match image channel size

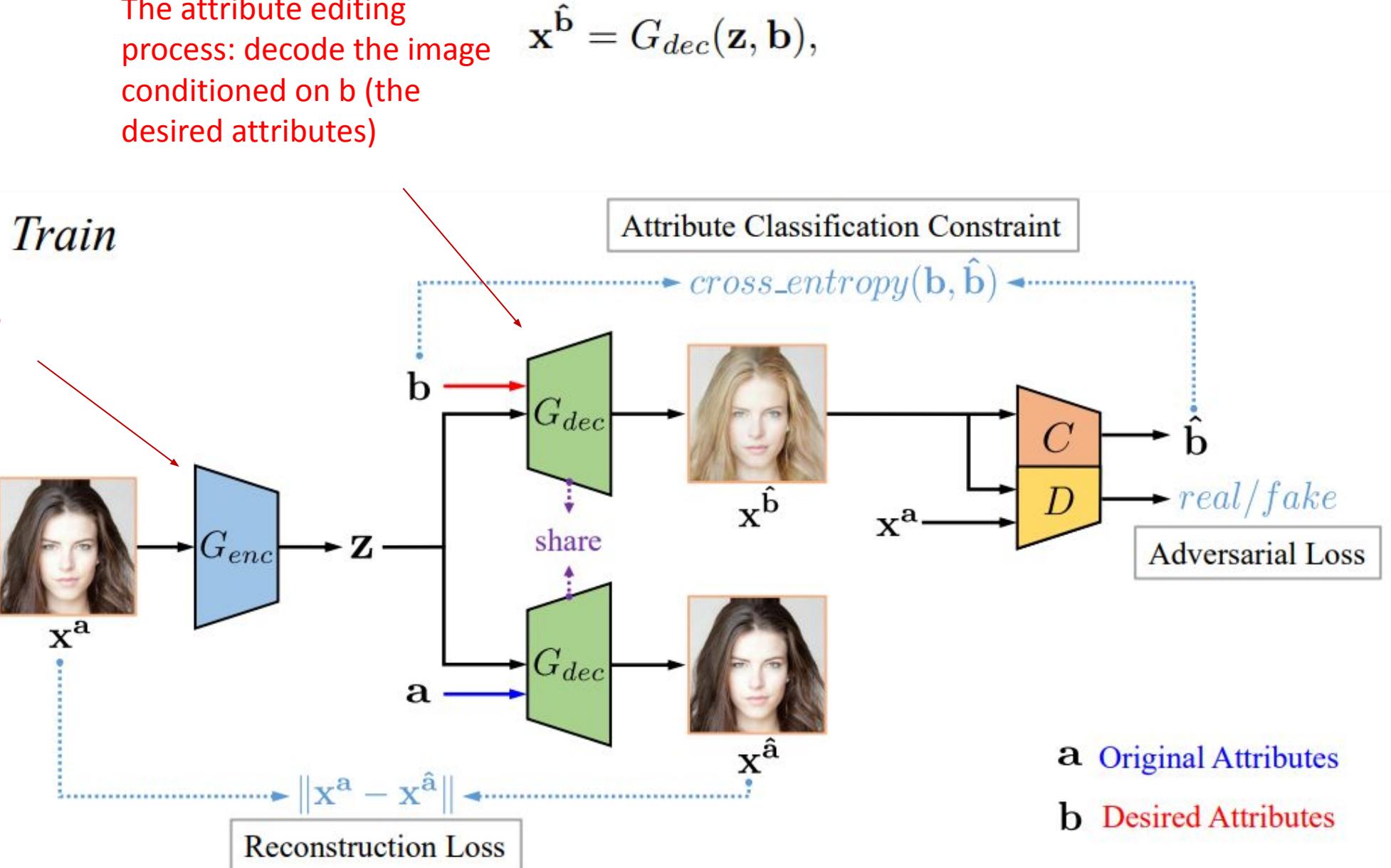
# AttGAN



# AttGAN

The attribute editing process: decode the image conditioned on b (the desired attributes)

$G_{enc}$ : encode the image into a latent representation, z



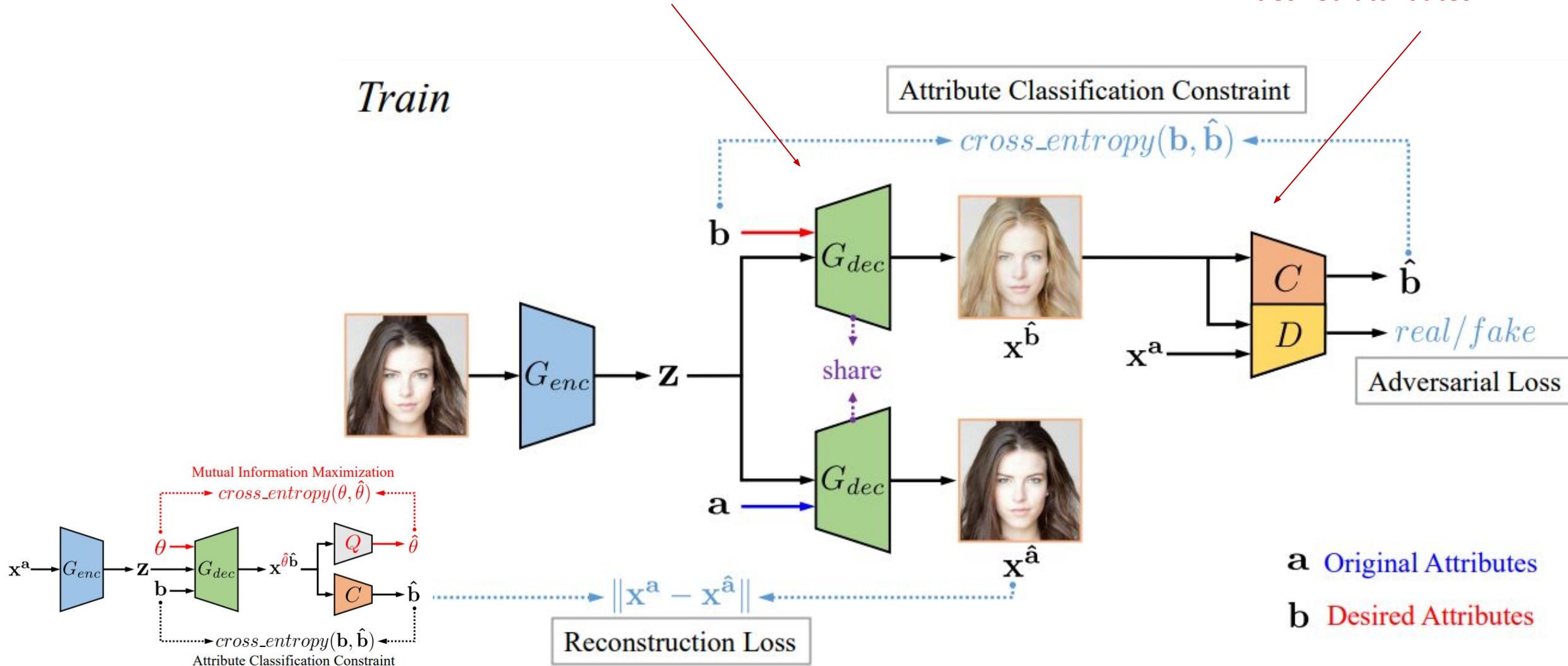
# AttGAN

Unsupervised problem:  
the ground truth of the  
editing image is not  
available

$$\mathbf{x}^{\hat{\mathbf{b}}} = G_{dec}(\mathbf{z}, \mathbf{b}),$$

Use an attribute classifier  
to constrain the generated  
image to contain the  
desired attributes

*Train*



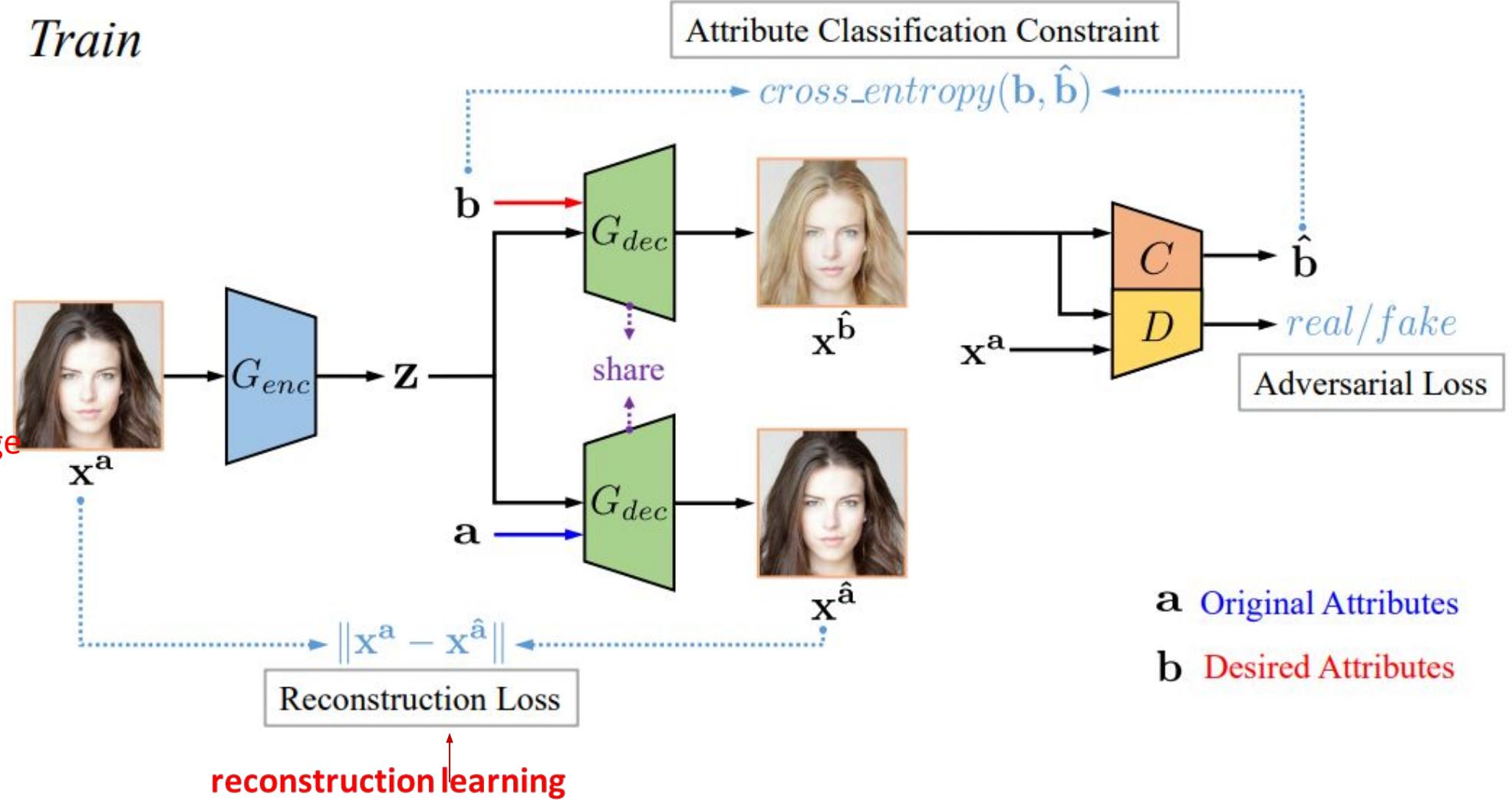
**a** Original Attributes

**b** Desired Attributes

# AttGAN

*Train*

We want to only change those desired attributes, while keeping the other details unchanged

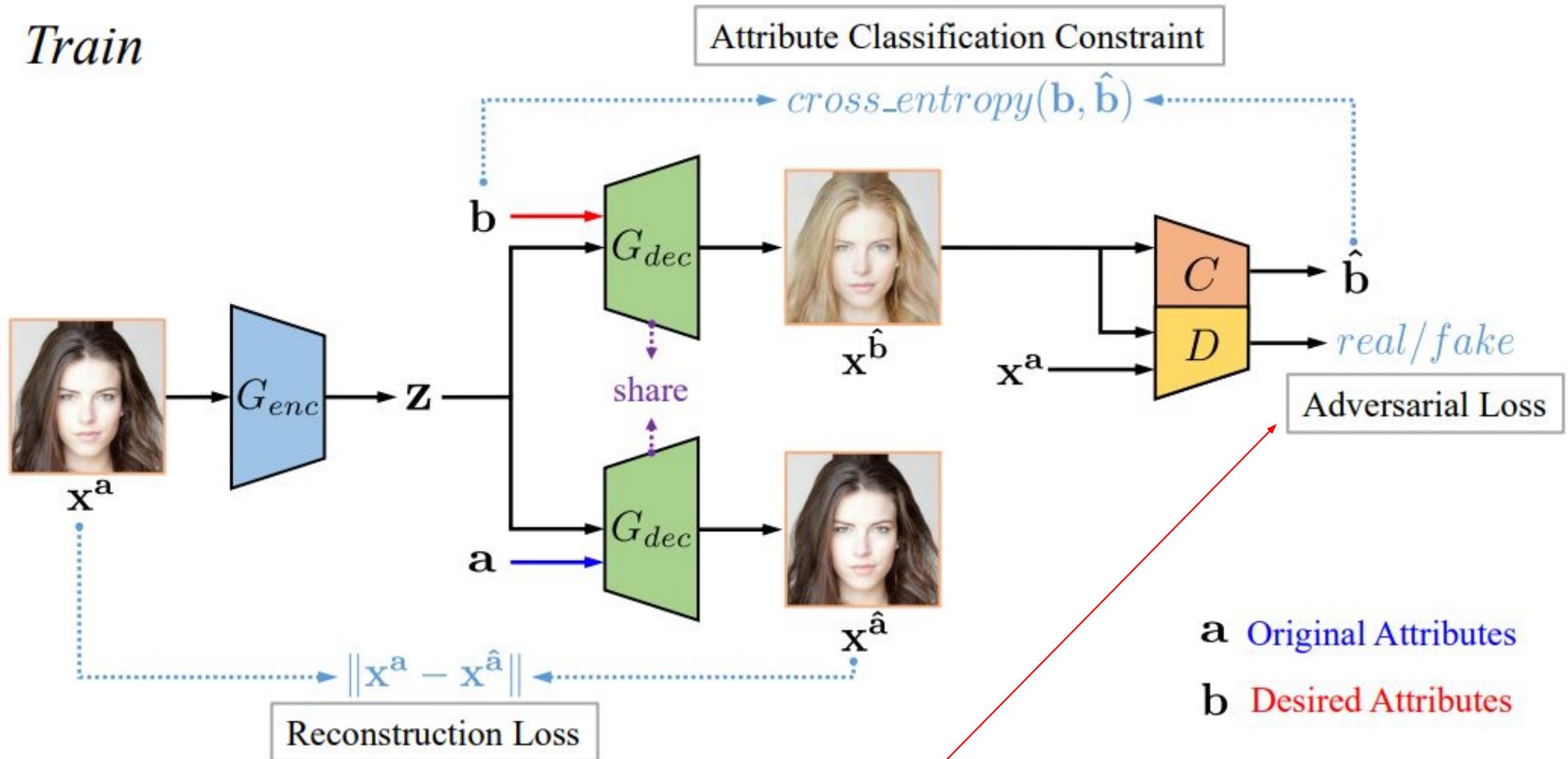


reconstruction learning

- $z$  should conserve enough information for the later recovery of the attribute-excluding details
- $G_{dec}$  should be able to restore the attribute excluding details from  $z$

# AttGAN

*Train*



adversarial learning between the generator (including the encoder and decoder) and the discriminator D – to make the generated image visually realistic

# AttGAN

## Generator (encoder, decoder)

$$\min_{G_{enc}, G_{dec}} \mathcal{L}_{enc, dec} = \lambda_1 \mathcal{L}_{rec} + \lambda_2 \mathcal{L}_{cls_g} + \mathcal{L}_{adv_g},$$



$$\min_{G_{enc}, G_{dec}} \mathcal{L}_{rec} = \mathbb{E}_{\mathbf{x}^a \sim p_{data}} [\|\mathbf{x}^a - \hat{\mathbf{x}}^a\|_1],$$

**Reconstruction** loss: satisfactory preservation of attribute excluding details

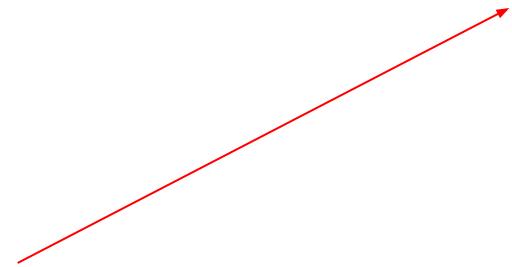
The decoder should be able to reconstruct the input image conditioned on the original attributes  $a$

# AttGAN

## Generator (encoder, decoder)

$$\min_{G_{enc}, G_{dec}} \mathcal{L}_{enc, dec} = \lambda_1 \mathcal{L}_{rec} + \lambda_2 \mathcal{L}_{cls_g} + \mathcal{L}_{adv_g},$$

The generated image should correctly own the new attributes b



$$\min_{G_{enc}, G_{dec}} \mathcal{L}_{cls_g} = \mathbb{E}_{\mathbf{x}^a \sim p_{data}, \mathbf{b} \sim p_{attr}} [\ell_g(\mathbf{x}^a, \mathbf{b})],$$

$$\ell_g(\mathbf{x}^a, \mathbf{b}) = \sum_{i=1}^n -b_i \log C_i(\mathbf{x}^{\hat{b}}) - (1 - b_i) \log(1 - C_i(\mathbf{x}^{\hat{b}})),$$

# AttGAN

Generator (encoder, decoder)

$$\min_{G_{enc}, G_{dec}} \mathcal{L}_{enc, dec} = \lambda_1 \mathcal{L}_{rec} + \lambda_2 \mathcal{L}_{cls_g} + \mathcal{L}_{adv_g},$$



Generated image should be visually realistic

$$\min_{\|D\|_L \leq 1} \mathcal{L}_{adv_d} = -\mathbb{E}_{\mathbf{x}^a \sim p_{data}} D(\mathbf{x}^a) + \mathbb{E}_{\mathbf{x}^a \sim p_{data}, \mathbf{b} \sim p_{attr}} D(\mathbf{x}^{\hat{b}}),$$

$$\min_{G_{enc}, G_{dec}} \mathcal{L}_{adv_g} = -\mathbb{E}_{\mathbf{x}^a \sim p_{data}, \mathbf{b} \sim p_{attr}} [D(\mathbf{x}^{\hat{b}})],$$

# AttGAN

## Discriminator, classification

$$\min_{D,C} \mathcal{L}_{dis,cls} = \lambda_3 \mathcal{L}_{cls_c} + \mathcal{L}_{adv_d},$$

$$\min_{G_{enc}, G_{dec}} \mathcal{L}_{cls_g} = \mathbb{E}_{\mathbf{x}^a \sim p_{data}, \mathbf{b} \sim p_{attr}} [\ell_g(\mathbf{x}^a, \mathbf{b})],$$

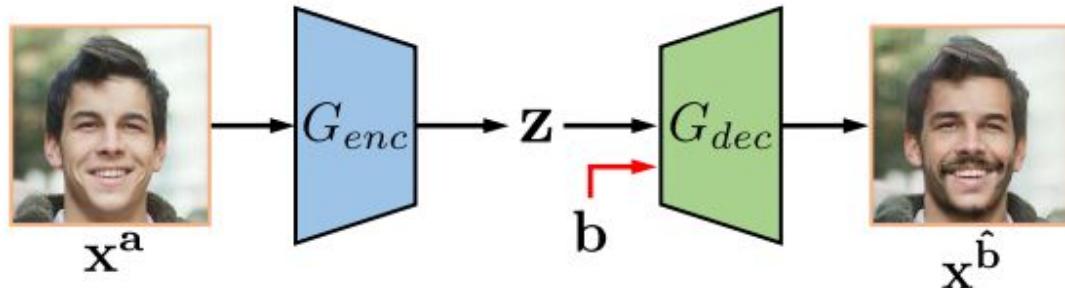
$$\ell_g(\mathbf{x}^a, \mathbf{b}) = \sum_{i=1}^n -b_i \log C_i(\mathbf{x}^{\hat{b}}) - (1 - b_i) \log(1 - C_i(\mathbf{x}^{\hat{b}})),$$

$$\min_{\|D\|_L \leq 1} \mathcal{L}_{adv_d} = -\mathbb{E}_{\mathbf{x}^a \sim p_{data}} D(\mathbf{x}^a) + \mathbb{E}_{\mathbf{x}^a \sim p_{data}, \mathbf{b} \sim p_{attr}} D(\mathbf{x}^{\hat{b}}),$$

$$\min_{G_{enc}, G_{dec}} \mathcal{L}_{adv_g} = -\mathbb{E}_{\mathbf{x}^a \sim p_{data}, \mathbf{b} \sim p_{attr}} [D(\mathbf{x}^{\hat{b}})],$$

# AttGAN

*Test*



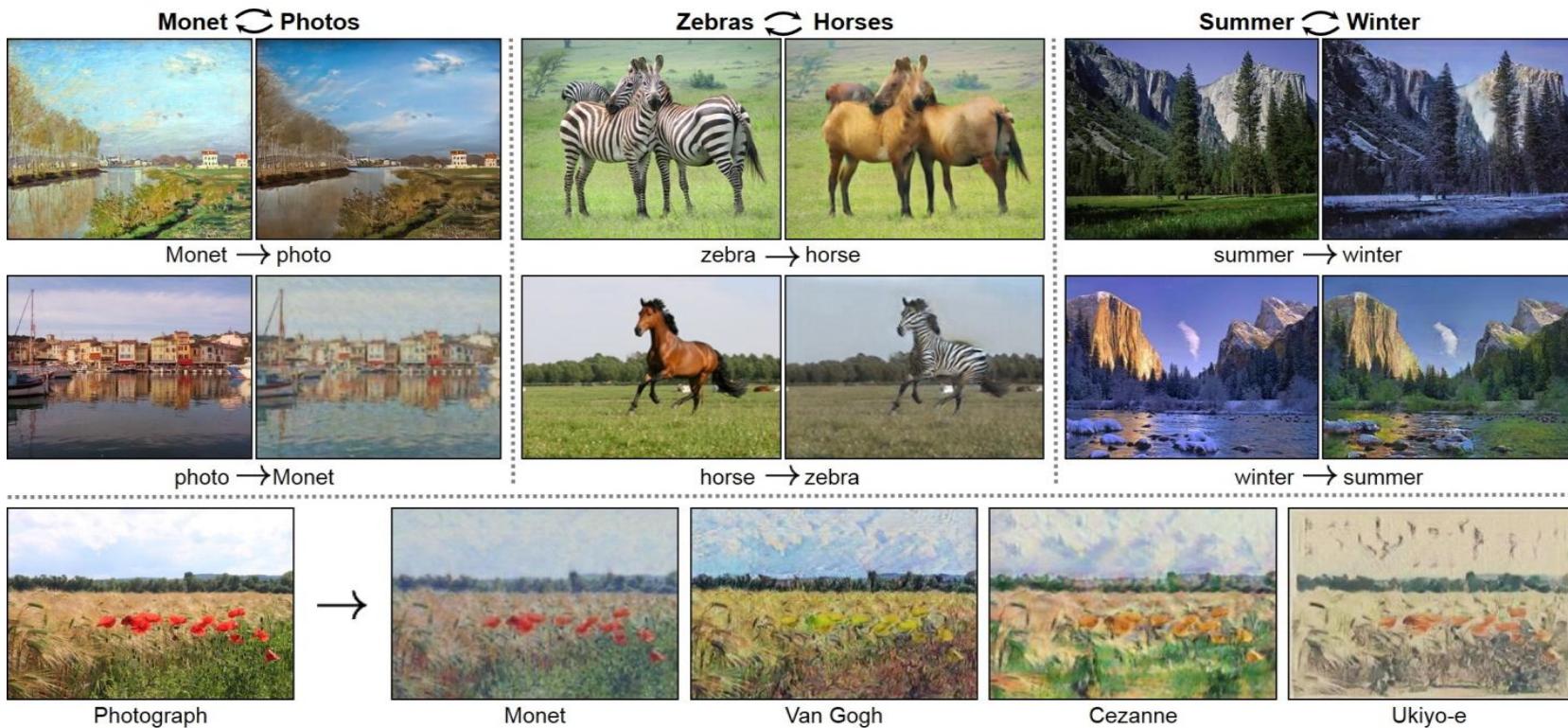
$$\mathbf{a} = [0, 1, \dots, \textcolor{blue}{0}, \dots] \quad \text{No Mustache}$$

$$\mathbf{b} = [0, 1, \dots, \textcolor{red}{1}, \dots] \quad \text{Mustache}$$



# Cycle GAN

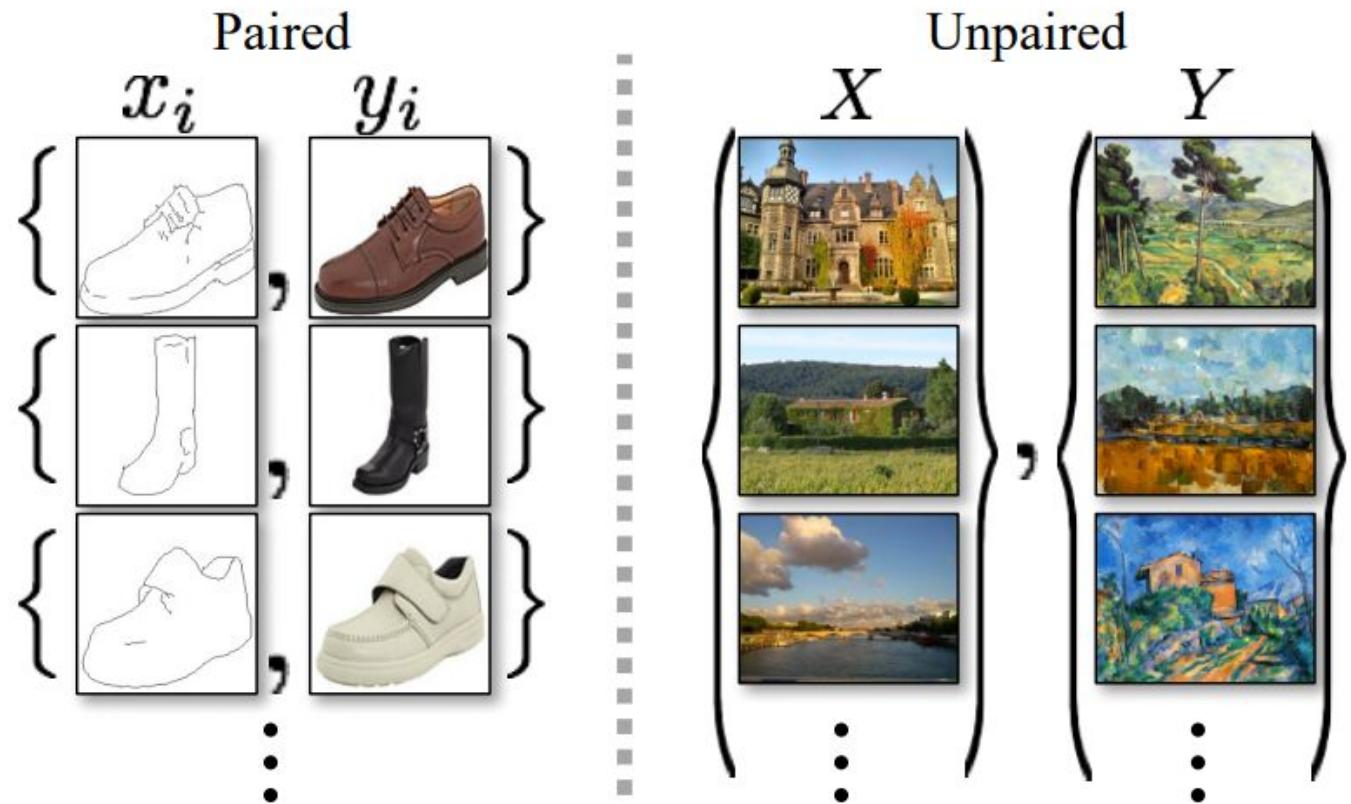
[https://www.youtube.com/watch?v=D4C1dB9UheQ&ab\\_channel=TwoMinutePapers](https://www.youtube.com/watch?v=D4C1dB9UheQ&ab_channel=TwoMinutePapers)



# Cycle GAN

**Goal:** capturing special characteristics of one image collection and figuring out how these characteristics could be translated into the other image collection, all in the *absence of any paired training example*

**supervision** is exploited at the level of sets: we are given one set of images in domain X and a different set in domain Y



### Original text:

There was a feller here once by the name of Jim Smiley, in the winter of '49 or may be it was the spring of '50 I don't recollect exactly, somehow; though what makes me think it was one or the other is because I remember the big flume wasn't finished when he first came to the camp; but any way, he was the curiosest man about always betting on any thing that turned up you ever see, if he could get any body to bet on the other side; and if he couldn't, he'd change sides. Any way that suited the other man would suit him any way just so's he got a bet, he was satisfied. But still he was lucky, uncommon lucky; he most always come out winner.

### Back Translation



### Text translated into French:

Il y avait une fois ici un individu connu sous le nom de Jim Smiley: c'était dans l'hiver de 49, peut-être bien au printemps de 50, je ne me rappelle pas exactement. Ce qui me fait croire que c'était l'un ou l'autre, c'est que je me souviens que le grand bief n'était pas achevé lorsqu'il arriva au camp pour la première fois, mais de toutes façons il était l'homme le plus friand de paris qui se put voir, pariant sur tout ce qui se présentait, quand il pouvait trouver un adversaire, et, quand il n'en trouvait pas, il passait du côté opposé. Tout ce qui convenait à l'autre lui convenait;

Et il avait une chance! une chance inouïe: presque toujours il gagnait.

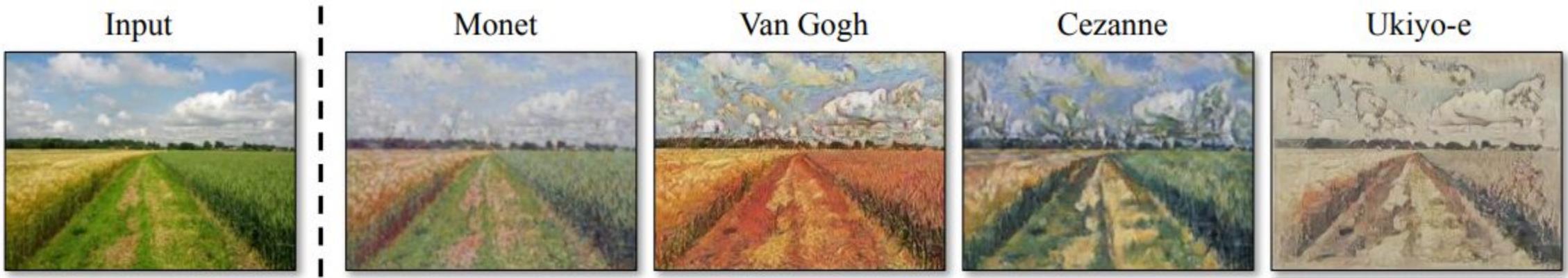
### Text re-translated into English by Mark Twain:

It there was one time here an individual known under the name of Jim Smiley; it was in the winter of '49, possibly well at the spring of '50, I no me recollect not exactly. This which me makes to believe that it was the one or the other, it is that I shall remember that the grand flume is not achieved when he arrives at the camp for the first time, but of all sides he was the man the most fond of to bet which one have seen, betting upon all that which is presented, when he could find an adversary; and when he not of it could not, he passed to the side opposed. All that which convenience to the other to him convenience also; seeing that he had a bet Smiley was satisfied. And he had a chance! a chance even worthless; nearly always he gained.

*The Jumping Frog: in English, then in French, and then Clawed Back into a Civilized Language Once More by Patient, Unremunerated Toil, Mark Twain*

# Cycle GAN

Unpaired image to image translation

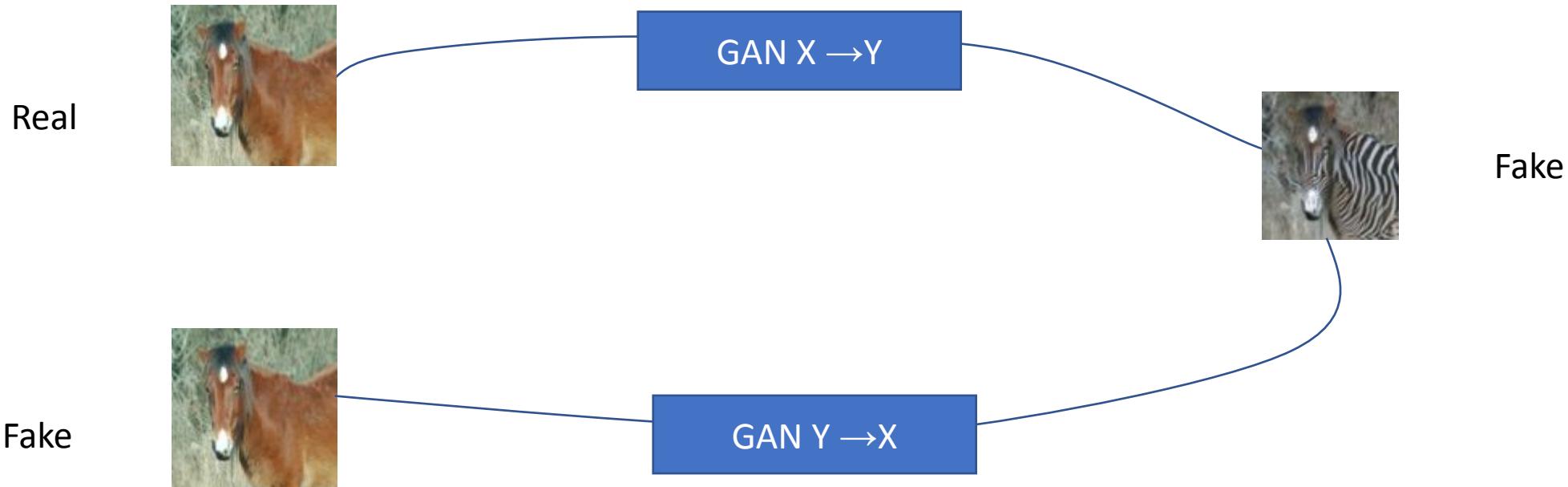


- Content - common elements, must be kept in the generated image
- Style – unique elements, must be transferred to the generated image

# Cycle GAN

Unpaired image to image translation

- Use two GANs
- Cycle consistency: getting the content to be preserved while only changing the styles



# Cycle GAN

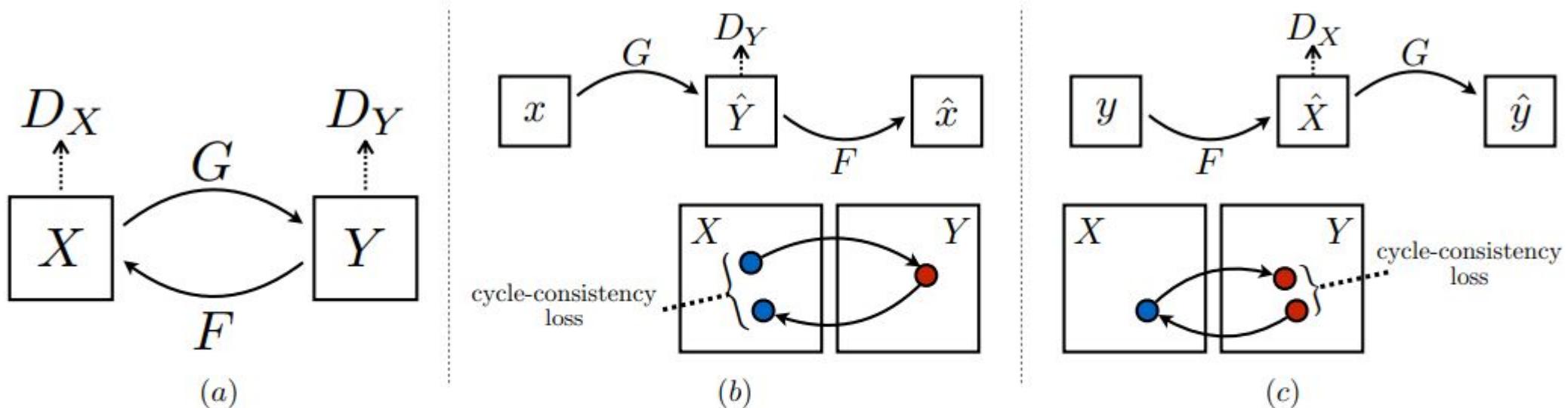
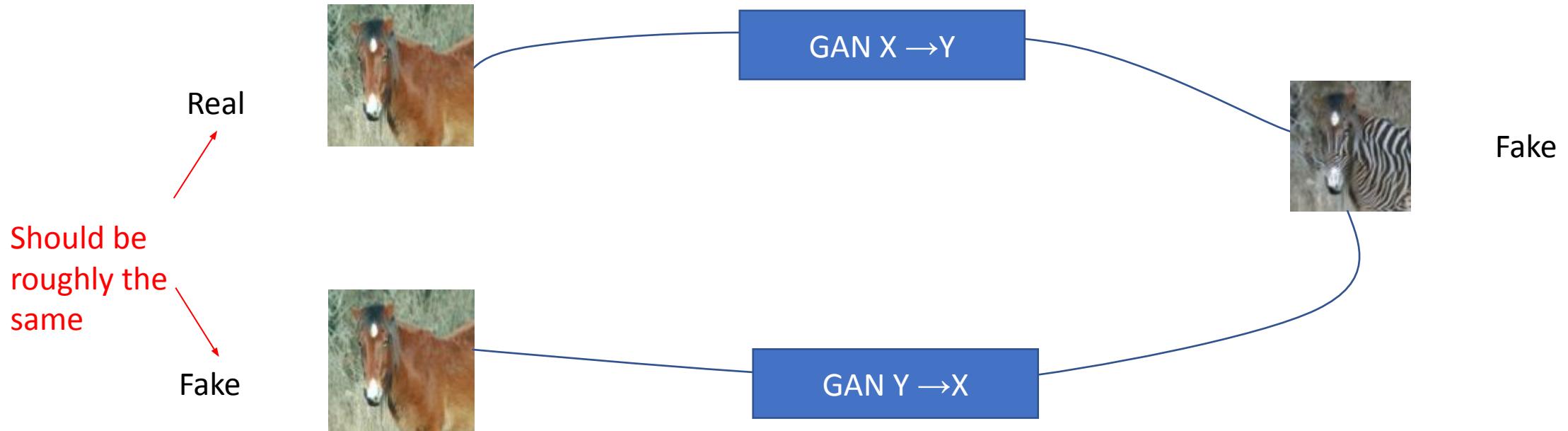


Figure 3: (a) Our model contains two mapping functions  $G : X \rightarrow Y$  and  $F : Y \rightarrow X$ , and associated adversarial discriminators  $D_Y$  and  $D_X$ .  $D_Y$  encourages  $G$  to translate  $X$  into outputs indistinguishable from domain  $Y$ , and vice versa for  $D_X$  and  $F$ . To further regularize the mappings, we introduce two *cycle consistency losses* that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss:  $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$ , and (c) backward cycle-consistency loss:  $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

# Cycle GAN

## Cycle consistency loss



$$\begin{aligned}\mathcal{L}_{\text{cyc}}(G, F) = & \mathbb{E}_{x \sim p_{\text{data}}(x)} [\|F(G(x)) - x\|_1] \\ & + \mathbb{E}_{y \sim p_{\text{data}}(y)} [\|G(F(y)) - y\|_1].\end{aligned}$$

# Cycle GAN

- Full loss function:

$$\begin{aligned}\mathcal{L}(G, F, D_X, D_Y) = & \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) \\ & + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) \\ & + \lambda \mathcal{L}_{\text{cyc}}(G, F),\end{aligned}$$

# FSGAN

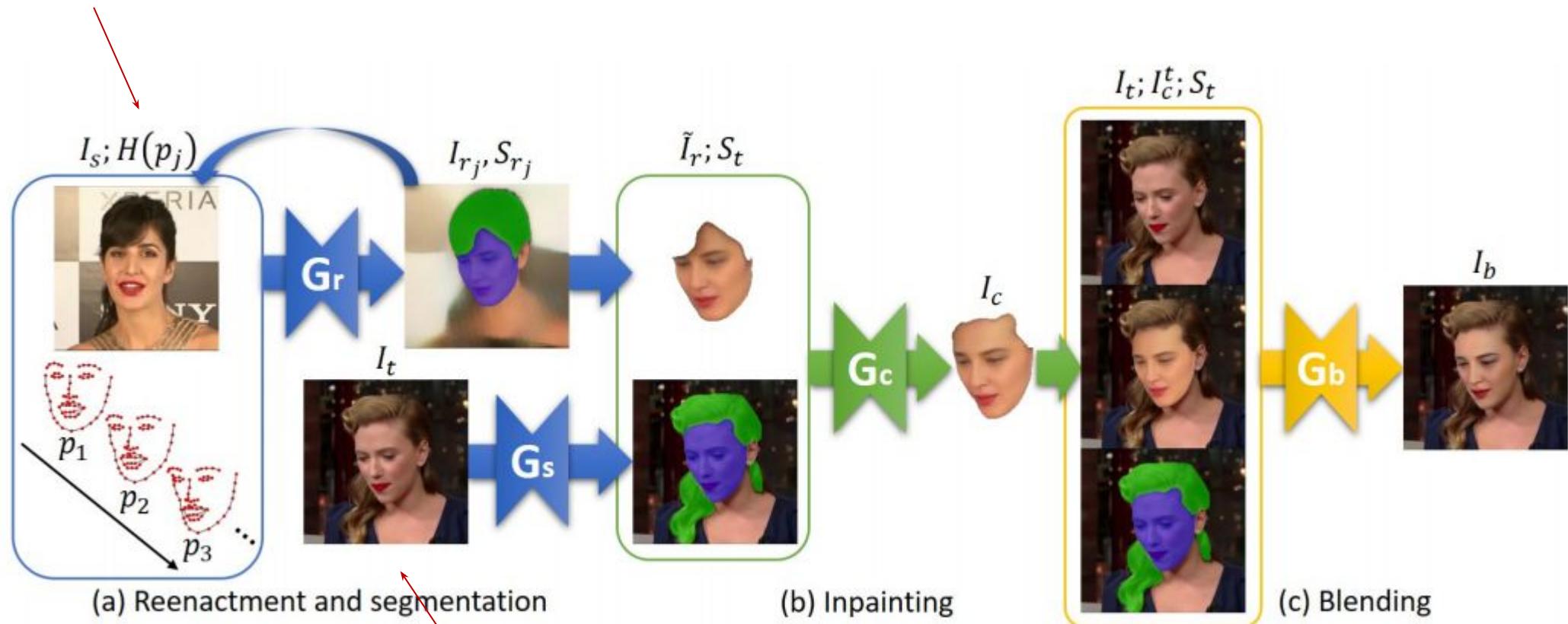
- Deep learning–based approach to face swapping and reenactment in images and videos
- Subject agnostic: applied to faces of different subjects **without requiring subject specific training**

<https://youtu.be/BsITEVX6hkE?t=7>

# FSGAN

Source image

$I_s$  – face in the source image



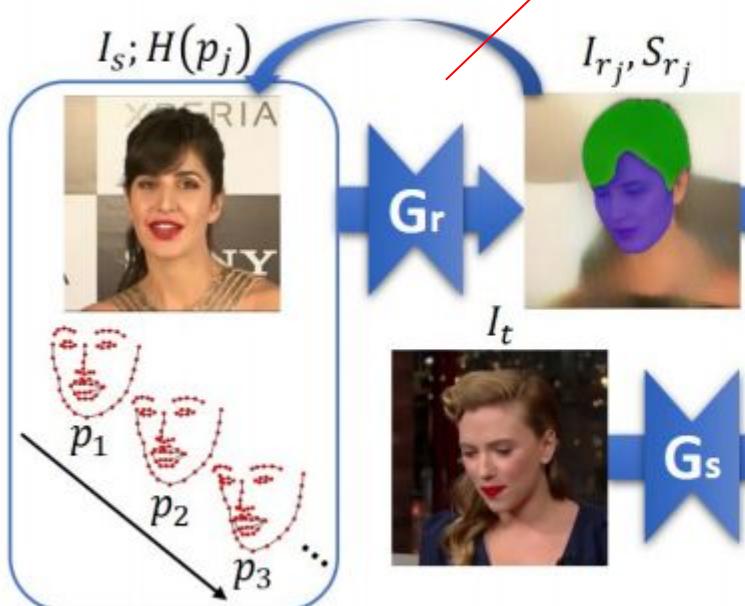
Goal: create a new image based on the target image, such that the face in this image is replaced by the face in the source image, while maintaining the pose and expression

Given an image  $I \in \mathbb{R}^{3 \times H \times W}$  and a heatmap representation  $H(p) \in \mathbb{R}^{70 \times H \times W}$  of facial landmarks,  $p \in \mathbb{R}^{70 \times 2}$ , we define the face reenactment generator,  $G_r$ , as the mapping  $G_r : \{\mathbb{R}^{3 \times H \times W}, \mathbb{R}^{70 \times H \times W}\} \rightarrow \mathbb{R}^{3 \times H \times W}$ .

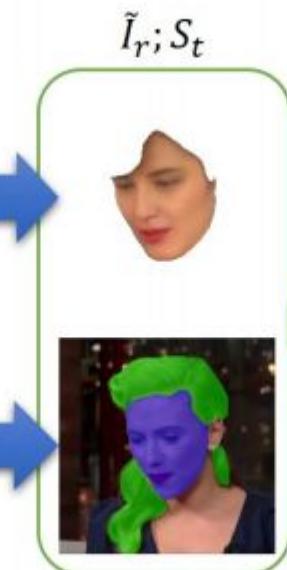
Let  $v_s, v_t \in \mathbb{R}^{70 \times 3}$  and  $e_s, e_t \in \mathbb{R}^3$ , be the 3D landmarks and Euler angles corresponding to  $F_s$  and  $F_t$ . We generate intermediate 2D landmark positions  $p_j$  by interpolating between  $e_s$  and  $e_t$ , and the centroids of  $v_s$  and  $v_t$ , using intermediate points for which we project  $v_s$  back to  $I_s$ . We define the reenactment output recursively for each iteration  $1 \leq j \leq n$  as

$$I_{r_j}, S_{r_j} = G_r(I_{r_{j-1}}; H(p_j)), \quad (6)$$

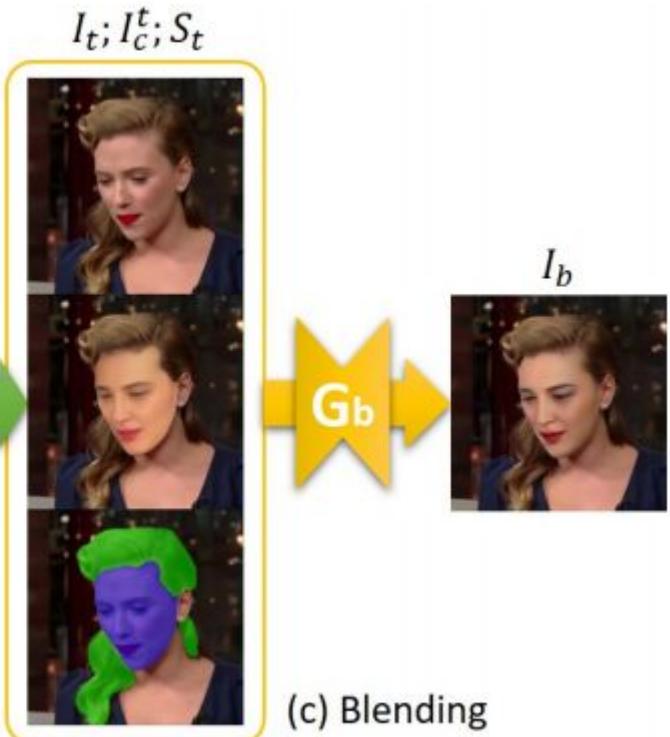
$$I_{r_0} = I_s.$$



(a) Reenactment and segmentation



(b) Inpainting



### Reenactment generator

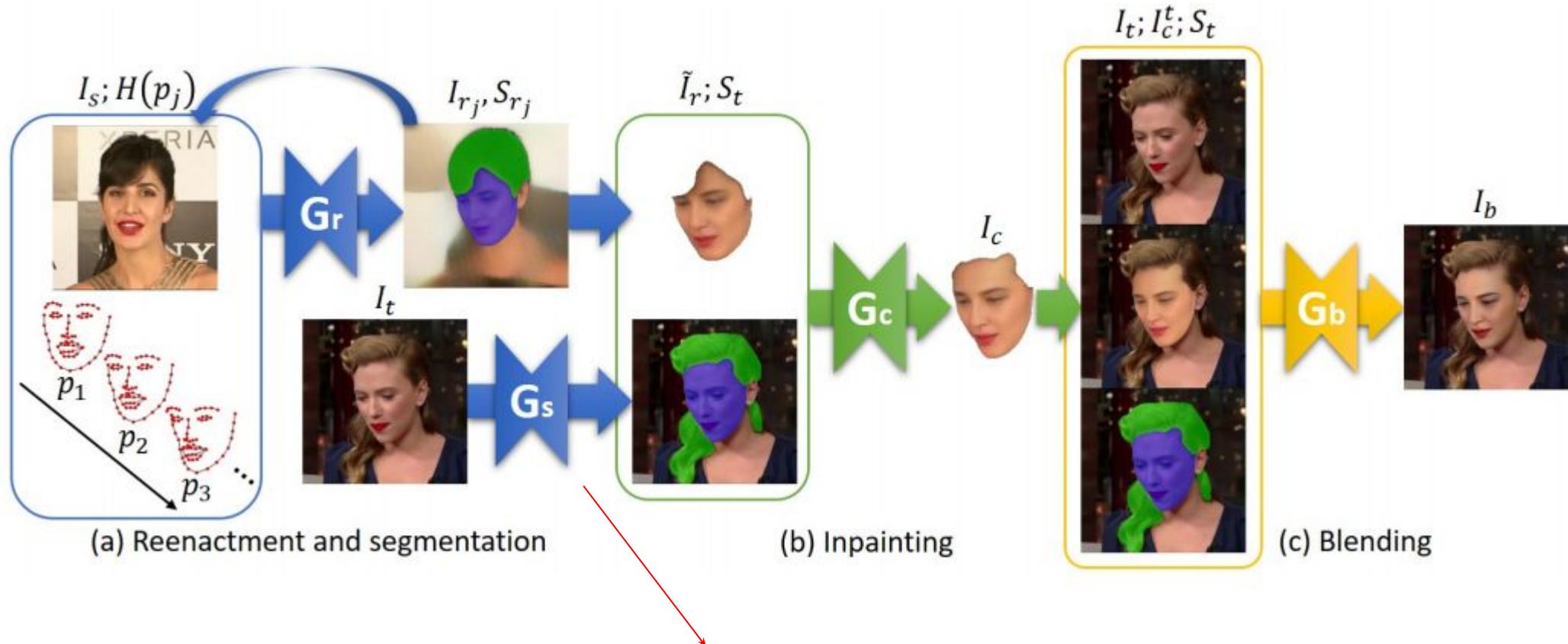
**Input:** heatmaps encoding the face landmarks in the target image

RNN

**Output:**

- $I_r$  – reenacted image, such that the face in this image ( $F_r$ ) depicts  $F_s$  at the same pose and expression as  $F_t$
- $S_r$  – segmentation map of  $F_r$  – face and hair

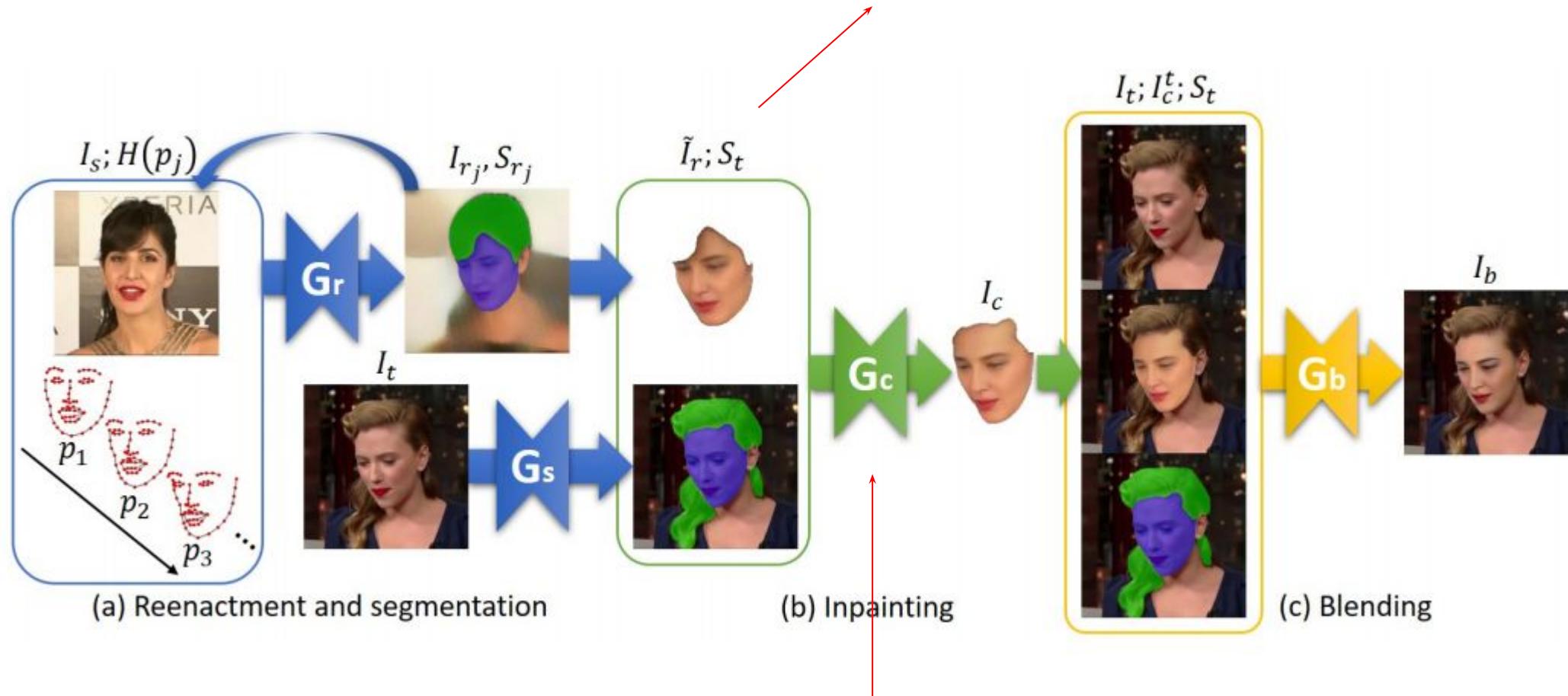
# FSGAN



Gs: compute the source image face-hair segmentation  
U-Net

# FSGAN

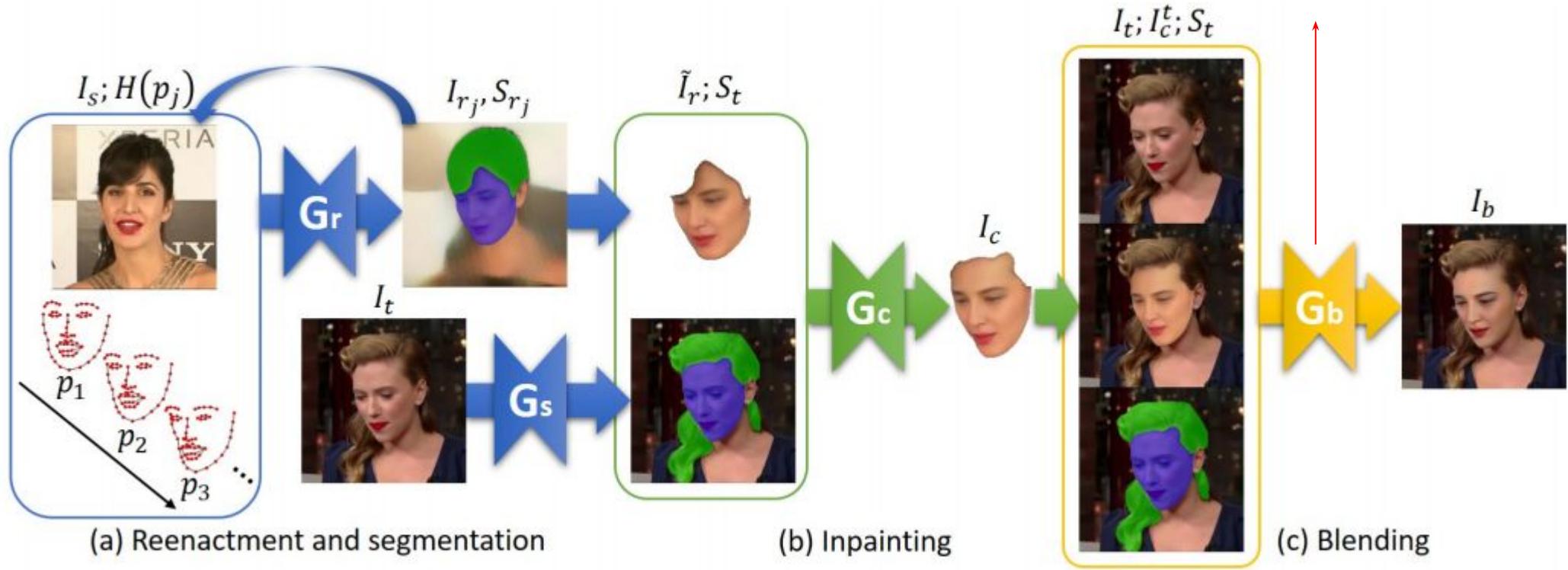
Re-enacted image might contain missing parts



Use a face inpainting network

# FSGAN

Blend the completed face  $F_c$  to the target face, accounting for different skin tones and lighting conditions



# Computer Vision and Deep Learning

Lecture 13

# Final exam

[CVDL \(coggle.it\)](#)

# EEML summer school

<https://www.eeml.eu/home>



Next week: virtual conference  
Don't forget to finish your teaser  
videos by 15<sup>th</sup> of Jan

CLIP

# CLIP

## FOOD101

**guacamole (90.1%)** Ranked 1 out of 101 labels



- ✓ a photo of **guacamole**, a type of food.
- ✗ a photo of **ceviche**, a type of food.
- ✗ a photo of **edamame**, a type of food.
- ✗ a photo of **tuna tartare**, a type of food.
- ✗ a photo of **hummus**, a type of food.

## SUN397

**television studio (90.2%)** Ranked 1 out of 397



- ✓ a photo of a **television studio**.
- ✗ a photo of a **podium indoor**.
- ✗ a photo of a **conference room**.
- ✗ a photo of a **lecture room**.
- ✗ a photo of a **control room**.

## YOUTUBE-BB

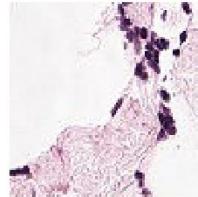
**airplane, person (89.0%)** Ranked 1 out of 23



- ✓ a photo of a **airplane**.
- ✗ a photo of a **bird**.
- ✗ a photo of a **bear**.
- ✗ a photo of a **giraffe**.
- ✗ a photo of a **car**.

## PATCHCAMELYON (PCAM)

**healthy lymph node tissue (22.8%)** Ranked 2 out of 2



- ✗ this is a photo of **lymph node tumor tissue**
- ✓ this is a photo of **healthy lymph node tissue**

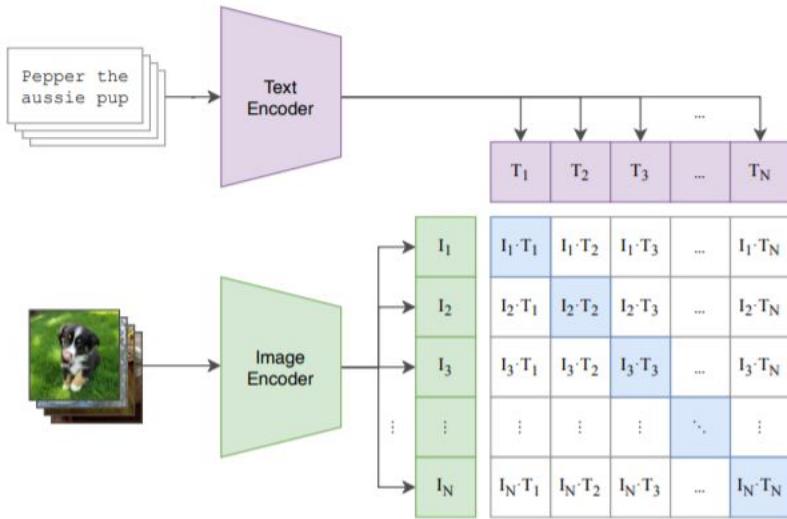
# Learning Transferable Visual Models From Natural Language Supervision

CLIP

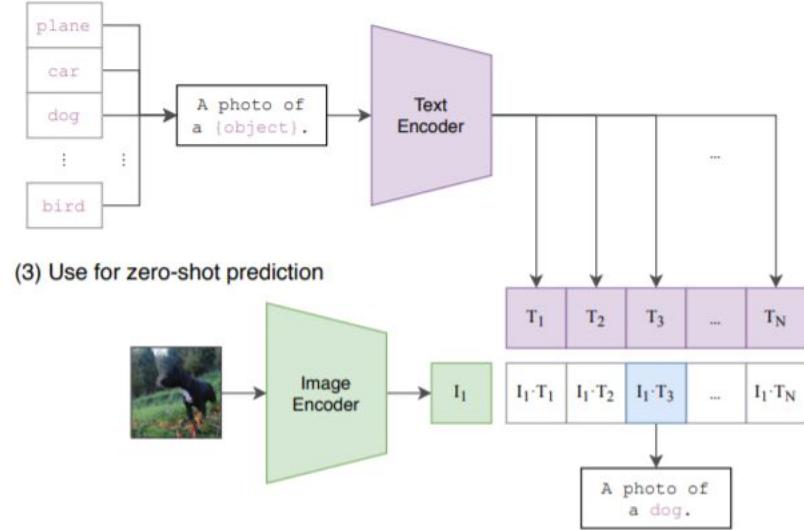
<https://arxiv.org/pdf/2103.00020.pdf>

# CLIP

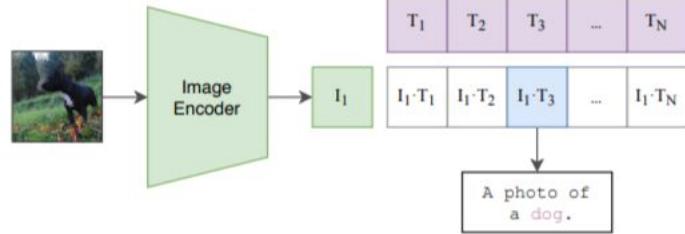
(1) Contrastive pre-training



(2) Create dataset classifier from label text



(3) Use for zero-shot prediction



*Figure 1.* Summary of our approach. While standard image models jointly train an image feature extractor and a linear classifier to predict some label, CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples. At test time the learned text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the target dataset's classes.

<https://www.youtube.com/watch?v=u0HG77RNhPE>

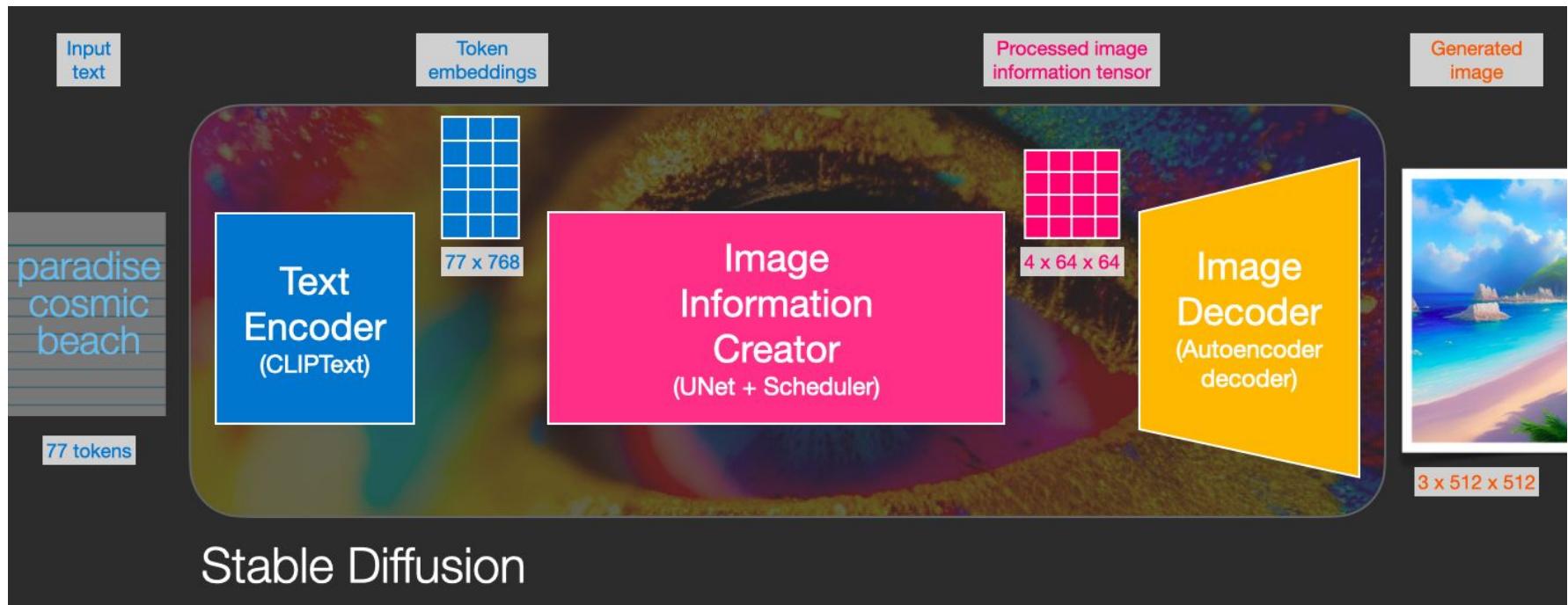
# STABLE DIFUSSION



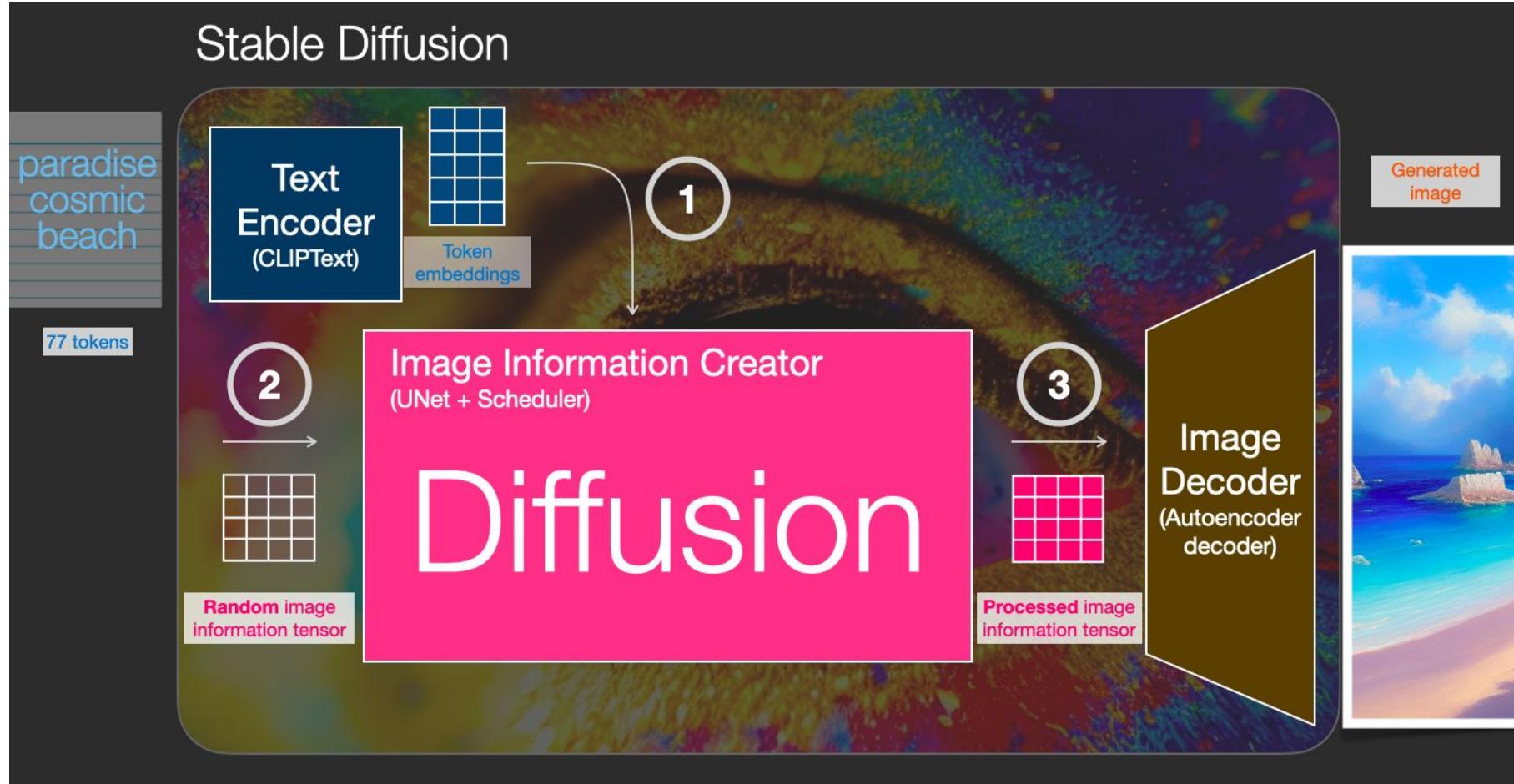
# Stable diffusion

- [https://www.youtube.com/watch?v=kVpDARqZdrQ&t=113s&ab\\_channel=AndrejKarpathy](https://www.youtube.com/watch?v=kVpDARqZdrQ&t=113s&ab_channel=AndrejKarpathy)
- [https://www.youtube.com/watch?v=sM9bozW295Q&ab\\_channel=AndrejKarpathy](https://www.youtube.com/watch?v=sM9bozW295Q&ab_channel=AndrejKarpathy)
- [https://www.youtube.com/watch?v=Jv1ayv-04H4&ab\\_channel=AndrejKarpathy](https://www.youtube.com/watch?v=Jv1ayv-04H4&ab_channel=AndrejKarpathy)

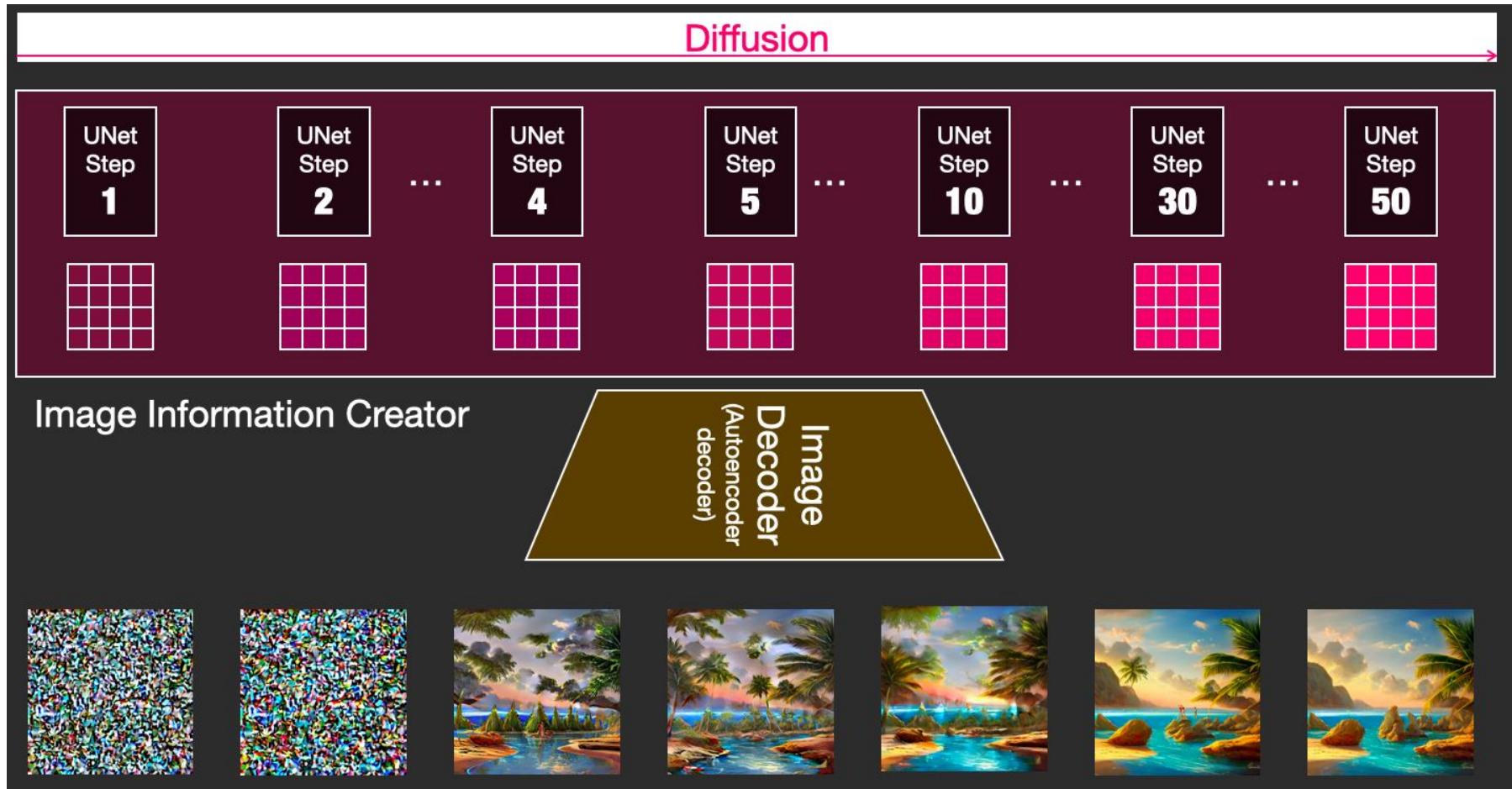
# Stable diffusion



# Stable diffusion



# Stable diffusion



# Stable diffusion

Training examples are created by generating **noise** and adding an **amount** of it to the images in the training dataset (forward diffusion)

- 1  
Pick an image

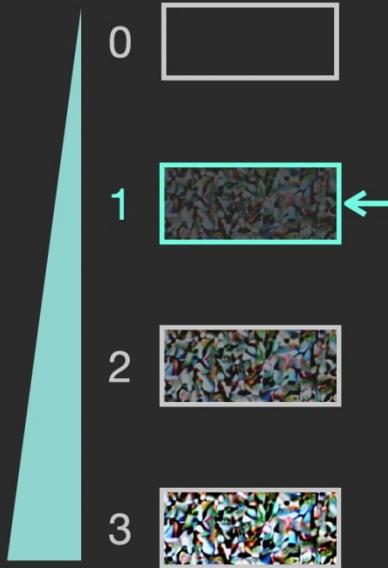


- 2  
Generate some random **noise**



Noise sample 1

- 3  
Pick an amount of **noise**

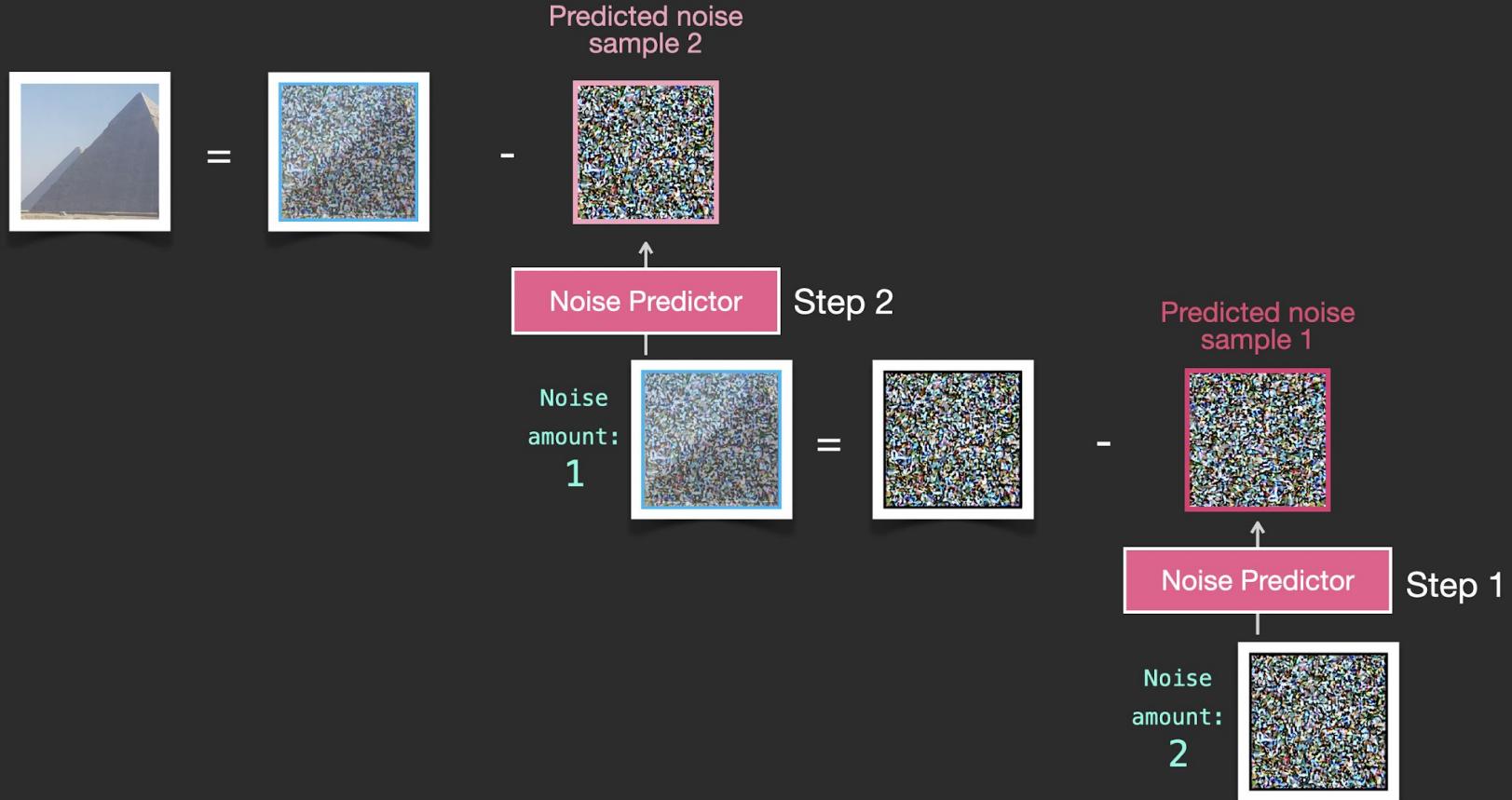


- 4  
Add **noise** to the image in that **amount**

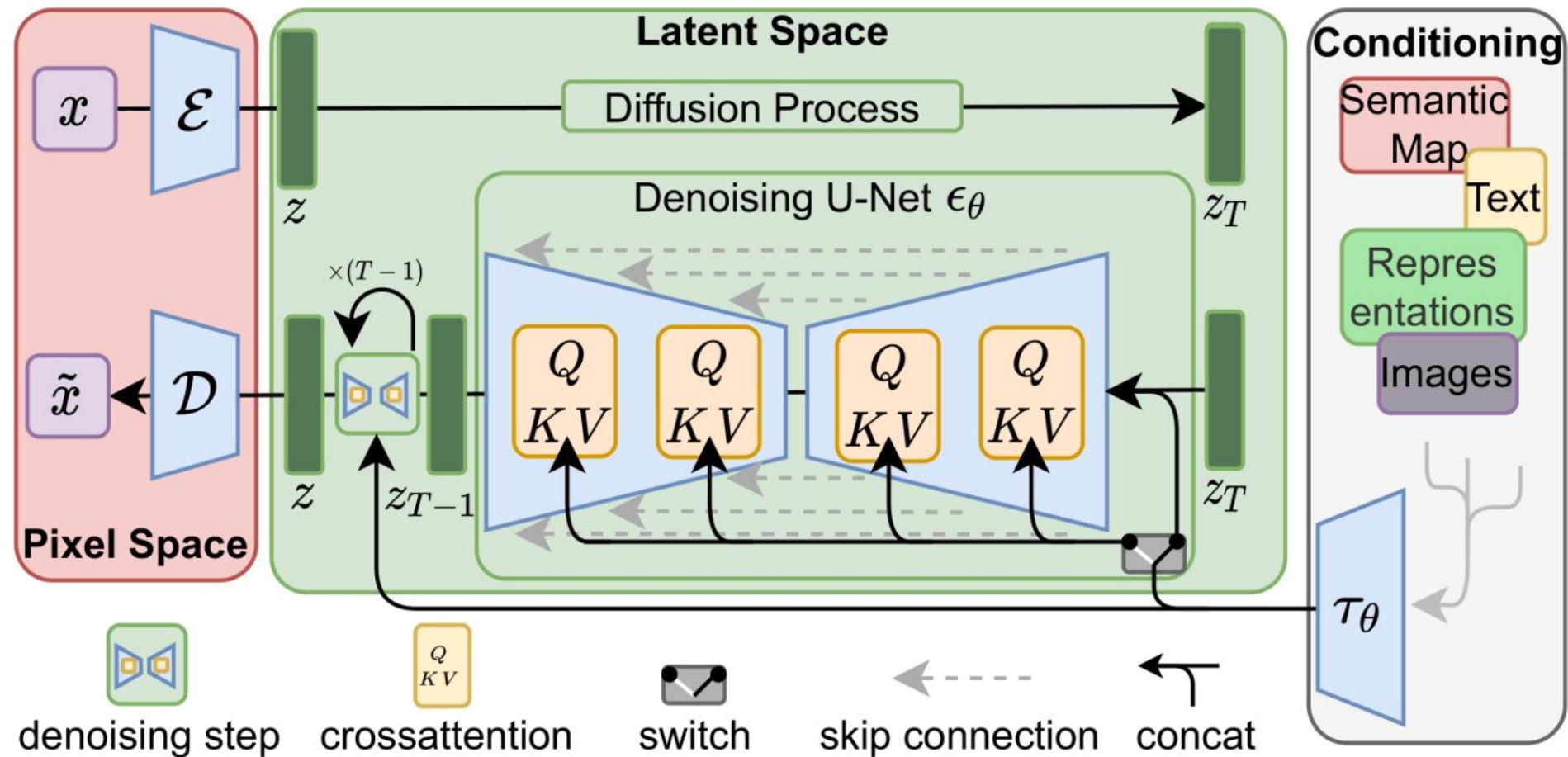


# Stable diffusion

## Image Generation by Reverse Diffusion (Denoising)



# Stable diffusion



# Deep fake detection

<https://www.youtube.com/watch?v=poSd2CyDpyA>