

Transformers in Vision

Giacomo Boracchi,
DEIB, Politecnico di Milano
January, 27th, 2023

giacomo.boracchi@polimi.it
<https://boracchi.faculty.polimi.it/>

Outline

Part1: CNN for Visual Recognition

- Convolutional Neural Networks
- Latent Representation: Transfer Learning
- Advanced Visual Recognition Problems

Part2: Attention and Transformers in Images

- Non-locality Principles in Imaging
- Attention mechanism for images: Non-Local Networks
- Visual Transformers

Part3: Combining Text and Images

- CLIP, Lseg, Dall-e

Part4: Eugenio's Show on ViT

CNN for Visual Recognition

Part1 Deep Learning Fundamentals

The First CNN

LeNet



Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

Abstract—

Multilayer Neural Networks trained with the backpropagation algorithm constitute the best example of a successful Gradient-Based Learning technique. Given an appropriate network architecture, Gradient-Based Learning algorithms can be used to synthesize a complex decision surface that can classify high-dimensional patterns such as handwritten characters, with minimal preprocessing. This paper reviews various methods applied to handwritten character recognition and compares them on a standard handwritten digit recognition task. Convolutional Neural Networks, that are specifically designed to deal with the variability of 2D shapes, are shown to outperform all other techniques.

I. INTRODUCTION

Over the last several years, machine learning techniques, particularly when applied to neural networks, have played an increasingly important role in the design of pattern recognition systems. In fact, it could be argued that the availability of learning techniques has been a crucial factor in the recent success of pattern recognition applications such as continuous speech recognition and handwriting recognition.

Gradient-Based Learning Applied to Document Recognition

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner

Abstract—

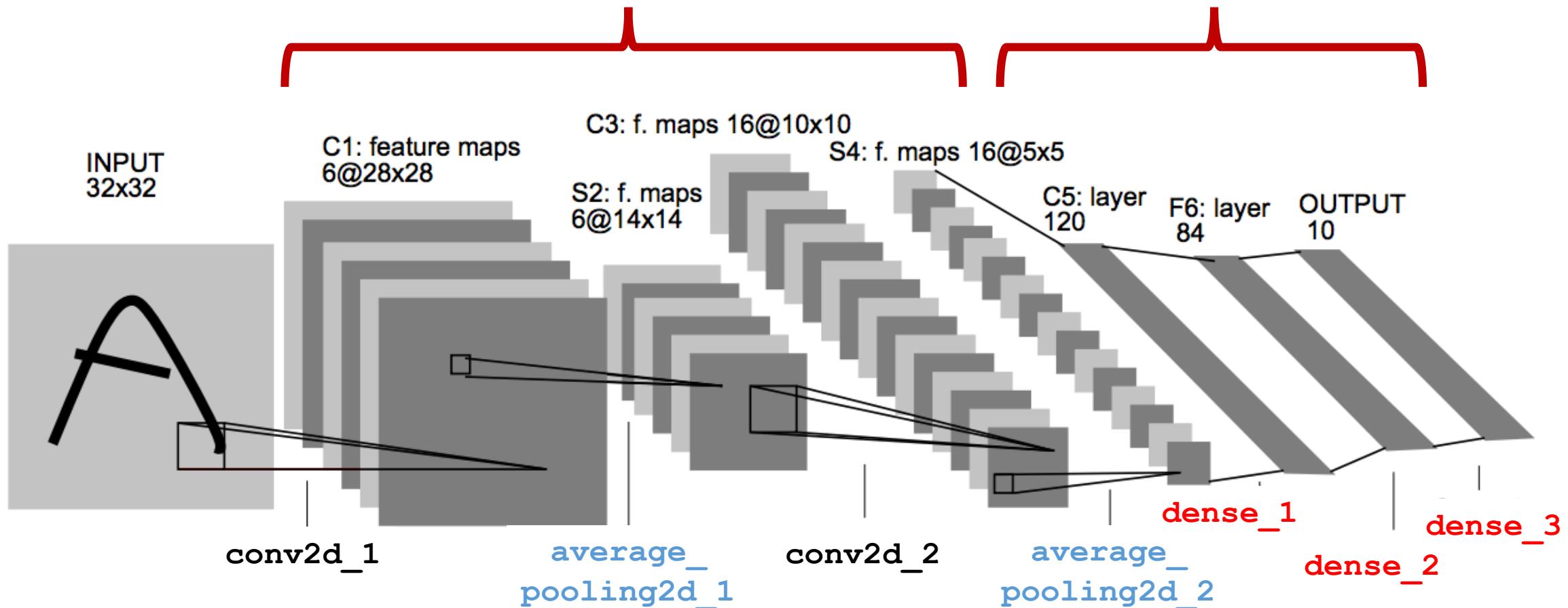
Multilayer Neural Networks trained with the backpropagation algorithm constitute the best example of a successful Gradient-Based Learning technique. Given an appropriate network architecture, Gradient-Based Learning algorithms can be used to synthesize a complex decision surface that can classify high-dimensional patterns such as handwritten characters, with minimal preprocessing. This paper reviews various methods applied to handwritten character recognition and compares them on a standard handwritten digit recognition task. Convolutional Neural Networks, that are specifically designed to deal with the variability of 2D shapes, are shown to outperform all other techniques.

I. INTRODUCTION

Over the last several years, machine learning techniques, particularly when applied to neural networks, have played an increasingly important role in the design of pattern recognition systems. In fact, it could be argued that the availability of learning techniques has been a crucial factor in the recent success of pattern recognition applications such as continuous speech recognition and handwriting recognition.

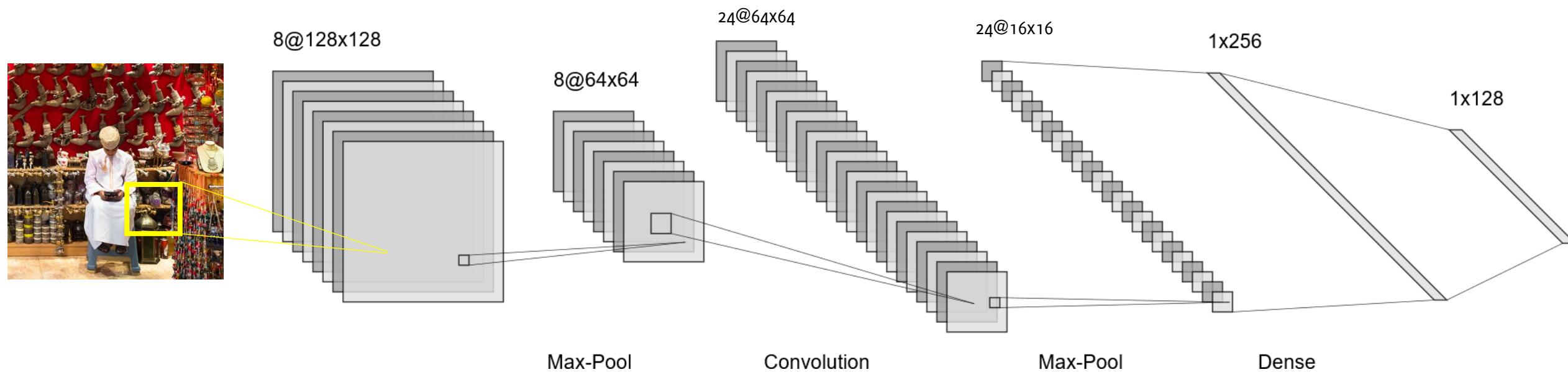
LeNet-5 (1998)

Stack of Conv2D + RELU + AVG-POOLING A TRADITIONAL MLP



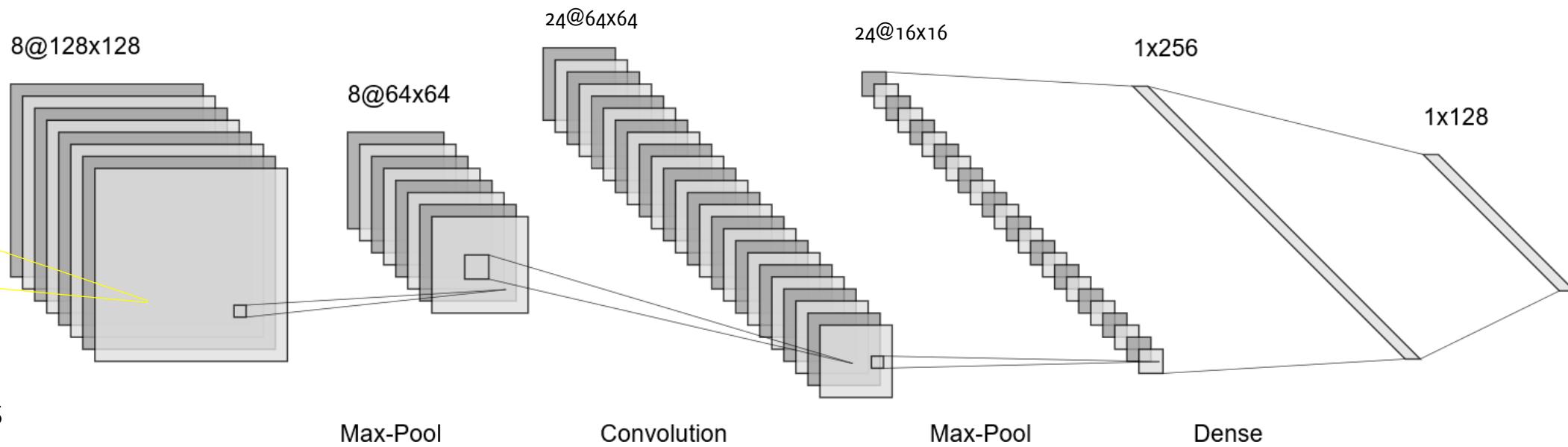
CNN Architecture...
just a recap

The typical architecture of a CNN



The typical architecture of a CNN

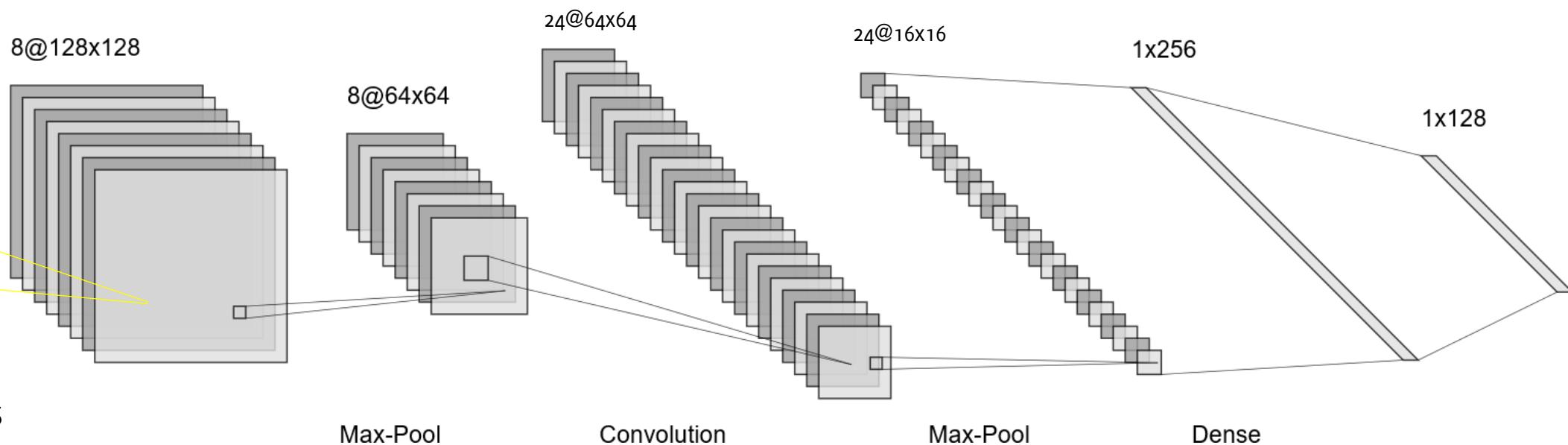
The input of a CNN
is an entire image



The image gets
convolved against
many filters

The typical architecture of a CNN

The input of a CNN
is an entire image

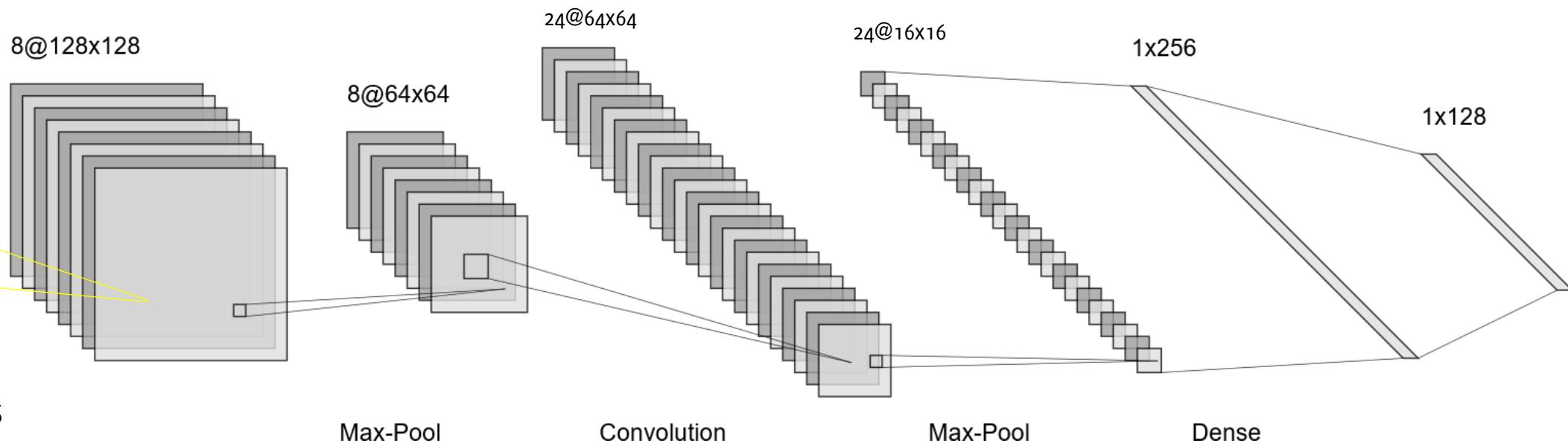


The image gets
convolved against
many filters

When progressing along the network, the
«number of images» or the «number of
channels in the images» increases, while
the image size decreases

The typical architecture of a CNN

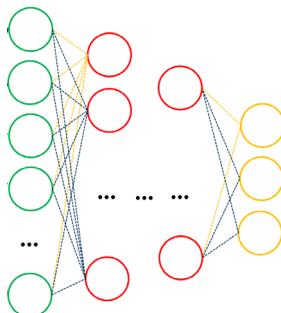
The input of a CNN
is an entire image



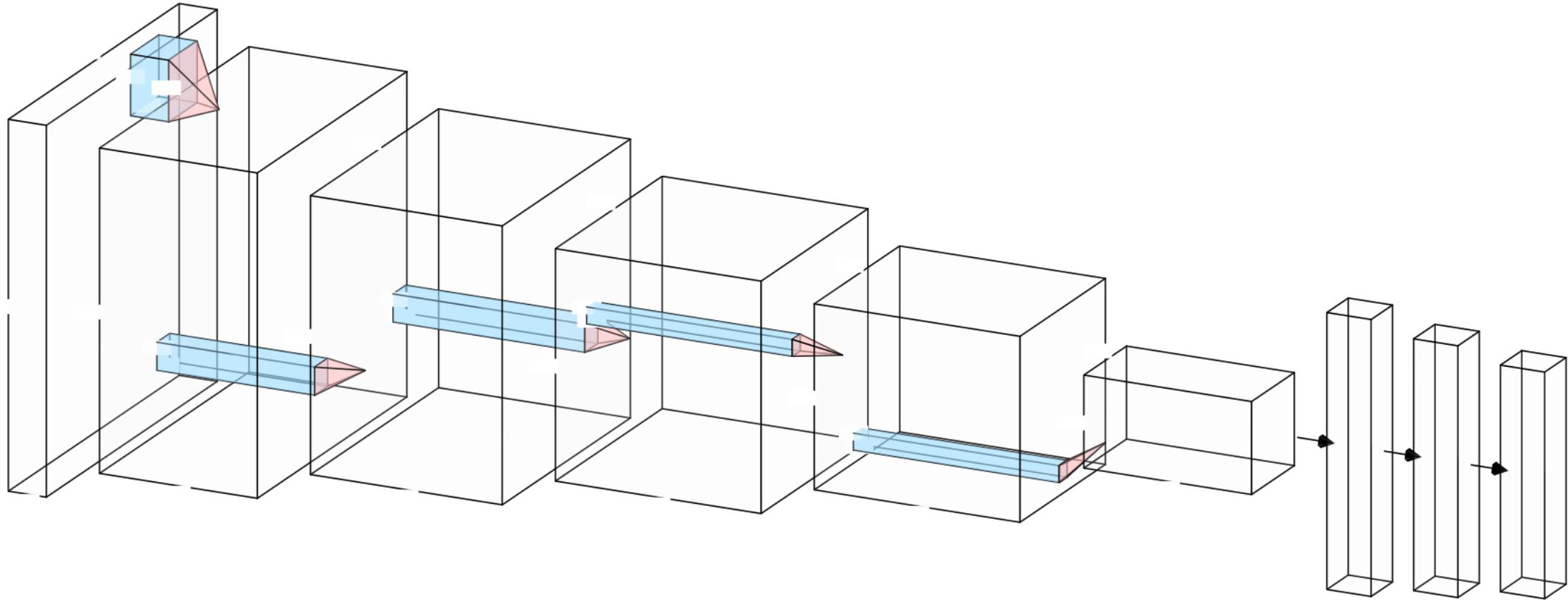
The image gets
convolved against
many filters

When progressing along the network, the
«number of images» or the «number of
channels in the images» increases, while
the image size decreases

Once the image
gets to a vector,
this is fed to a
traditional
neural network



Another Popular Illustration

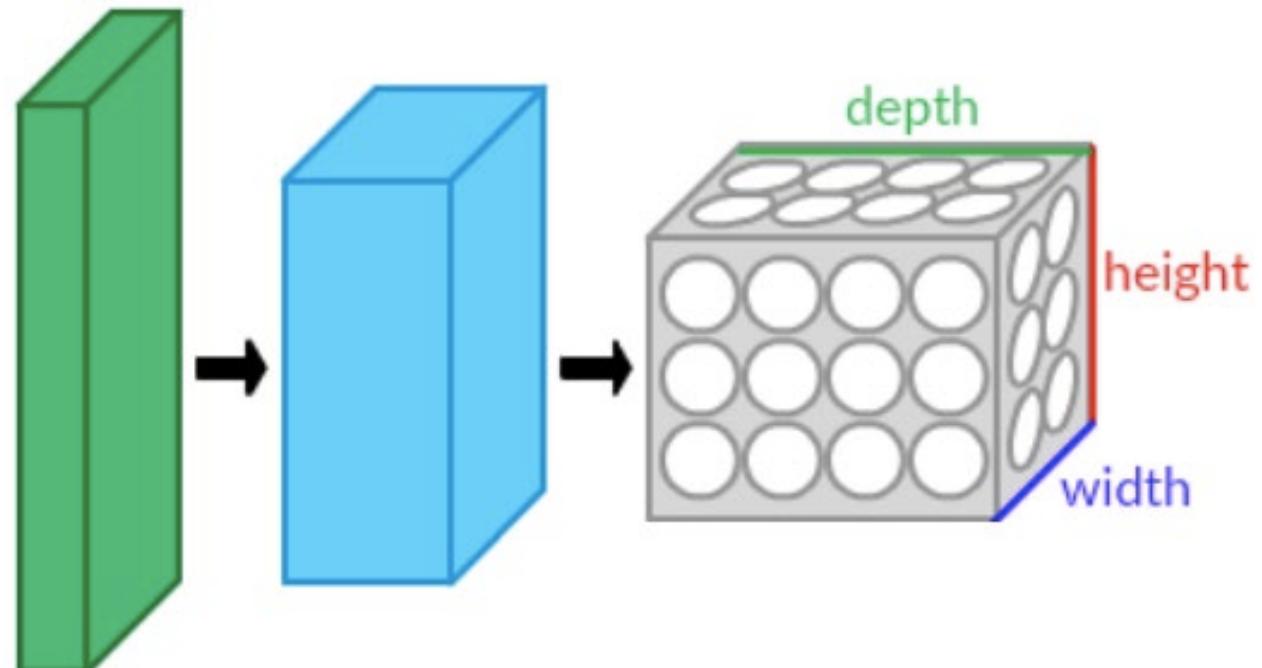


Convolutional Neural Networks (CNN)

CNN are typically made of blocks that include:

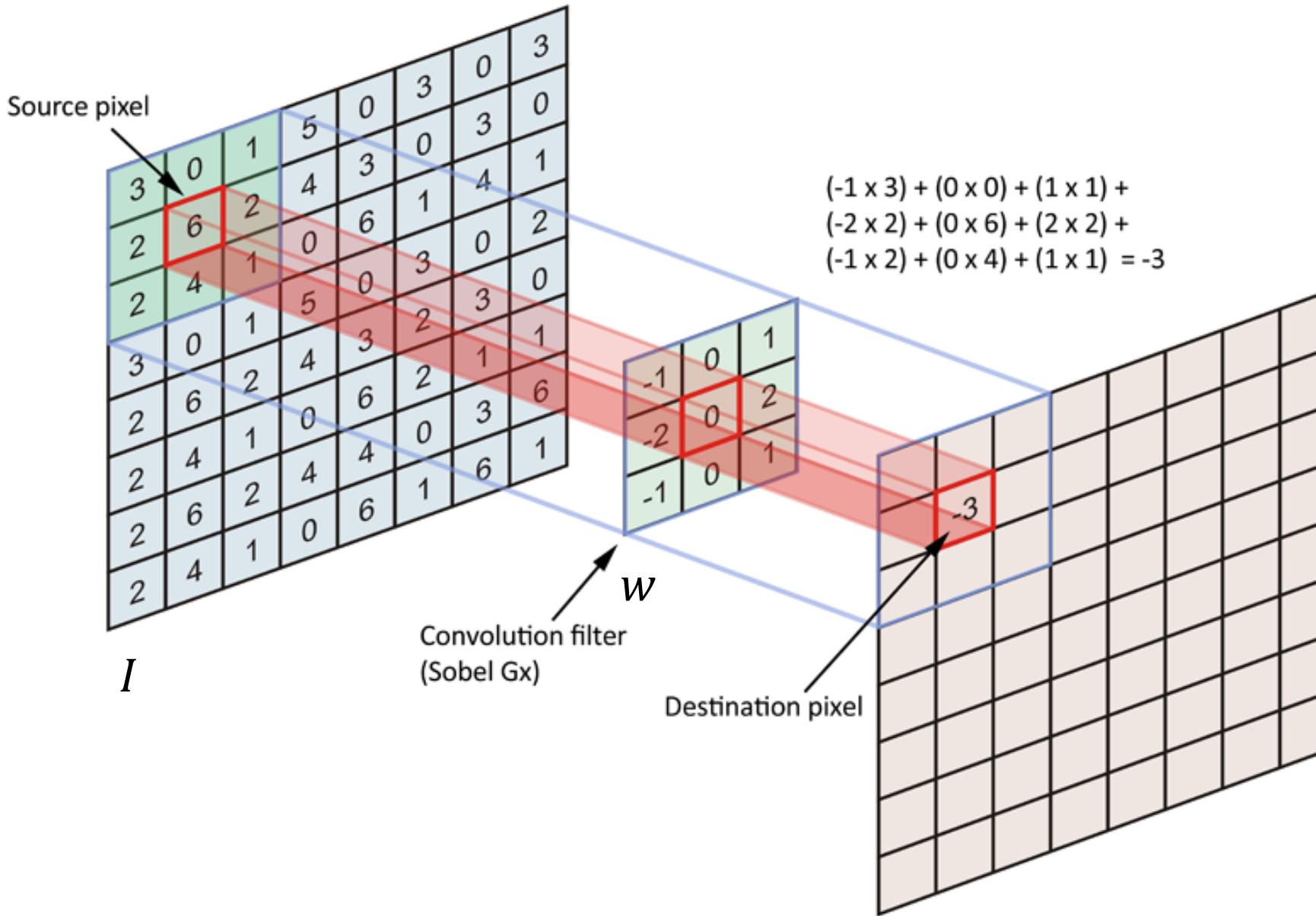
- Convolutional layers
- Nonlinearities (activation functions)
- Pooling Layers (Subsampling / maxpooling)

Each layer takes as input
and returns a volume



Convolutional Layers

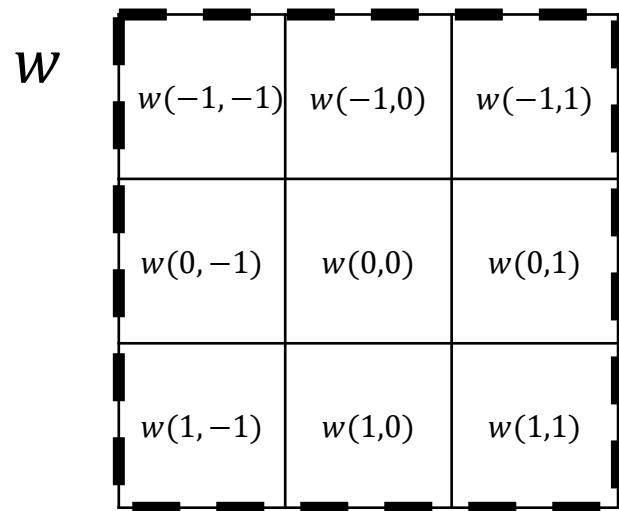
What is a convolution?



Convolution in ConvLayers (aka Correlation)

Convolution is a linear transformation. Linearity implies that

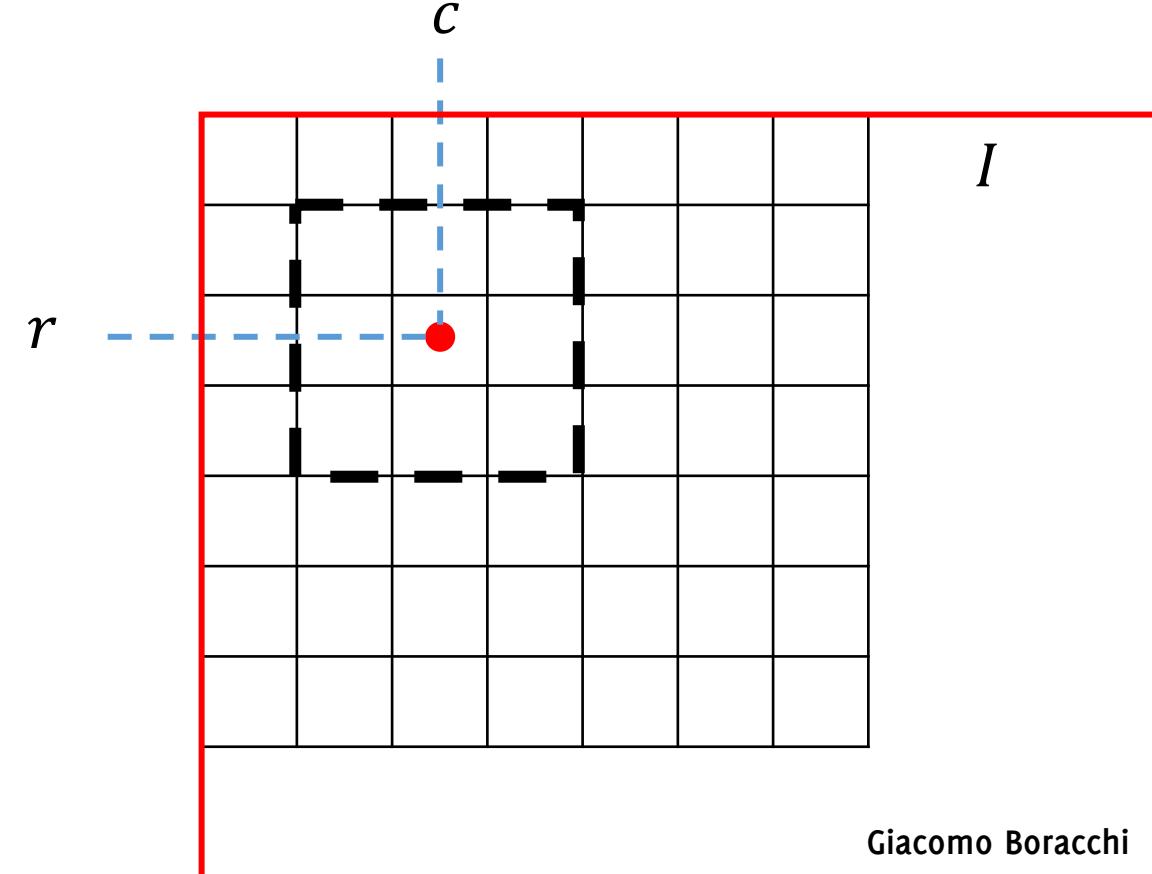
$$(I \circledast w)(r, c) = \sum_{(u,v) \in U} w(u, v) I(r + u, c + v)$$



Rmk: indexes
have been shifted
in the filter w

We can consider weights as a filter h that
entirely defines the convolution.

Convolution operates the same in each pixel

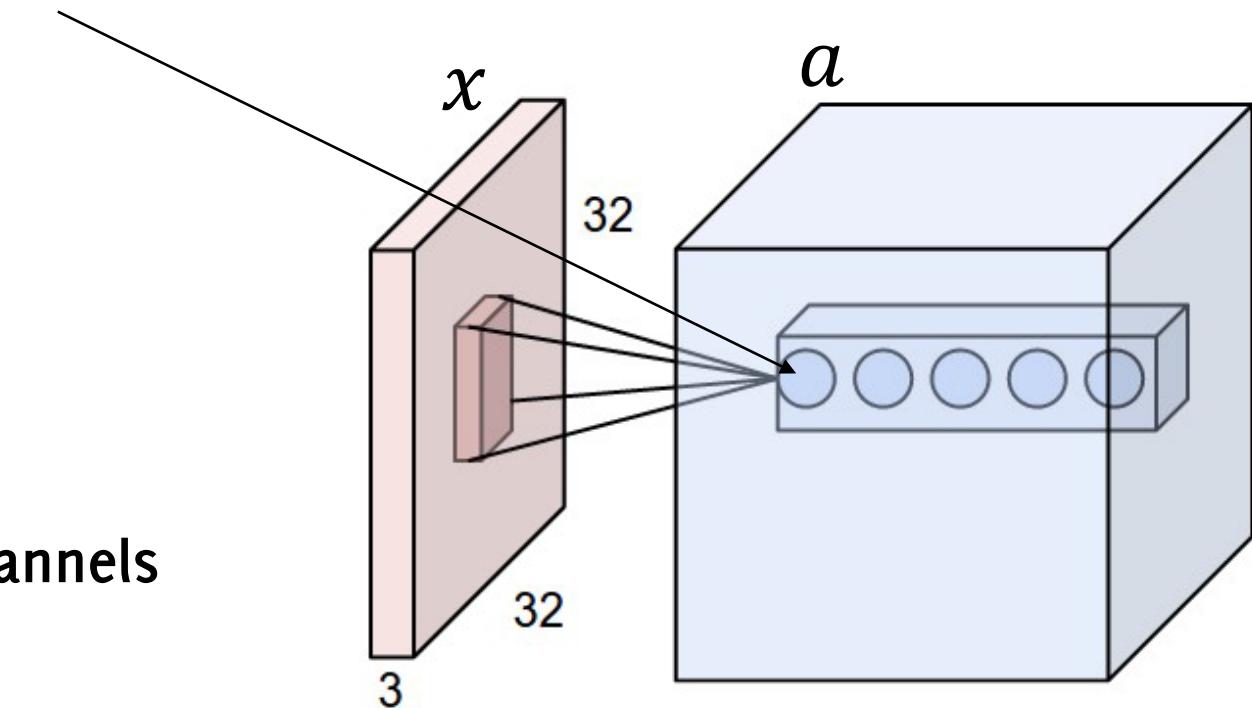


Convolutional Layers

Convolutional layers "mix" all the input components

The output is a linear combination of all the values in a region of the input, considering all the channels

$$a(r, c, 1) = \sum_{i,j,k} w^1(i, j, k) x(r + i, c + j, k) + b^1$$



Filters need to have **the same number of channels** as the input, to process all the values from the input layer

By Aphex34 - Own work, CC BY-SA 4.0,
<https://commons.wikimedia.org/w/index.php?curid=45659236>

Giacomo Boracchi

Convolutional Layers

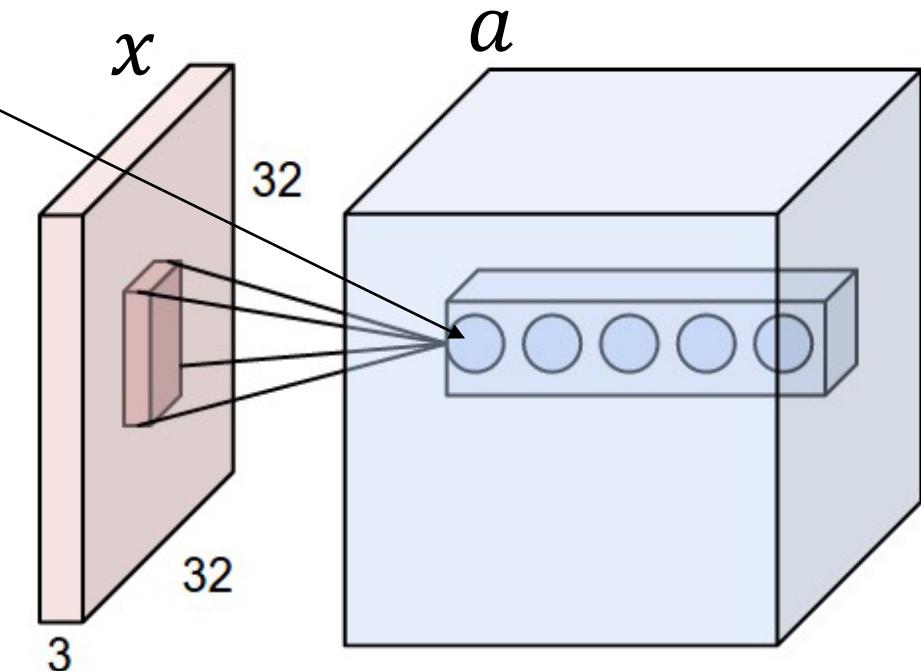
Convolutional layers "mix" all the input components

The output is a linear combination of all the values in a region of the input, considering all the channels

$$a(r, c, 1) = \sum_{i,j,k} w^1(i, j, k) x(r + i, c + j, k) + b^1$$

The parameters of this layer are called filters.

The same filter is used through the whole spatial extent of the input



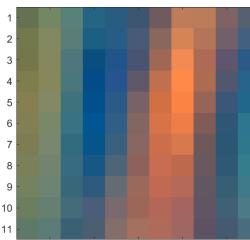


Convolutional Layers

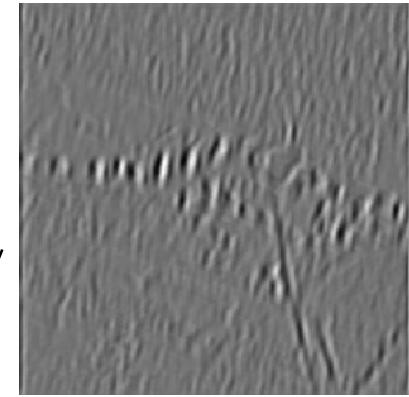
I



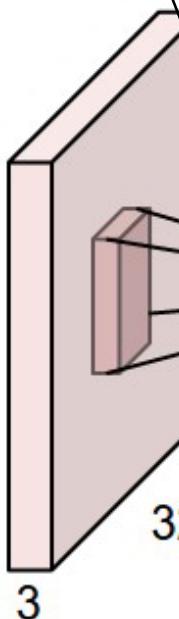
w^1



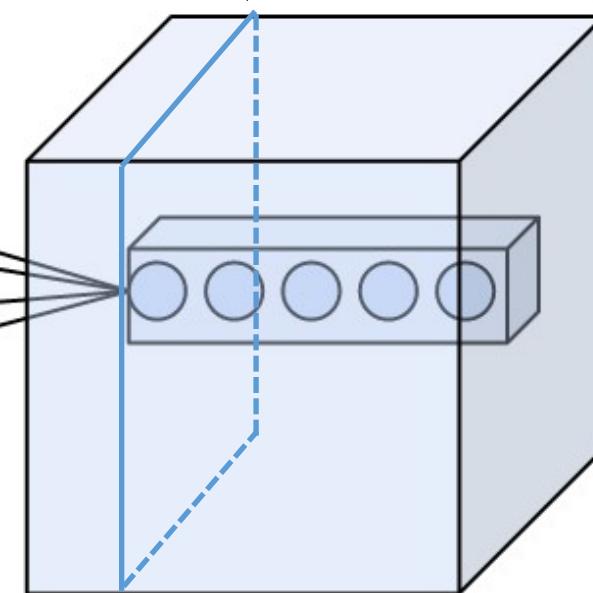
b^1



32



32



$a(:,:,1)$

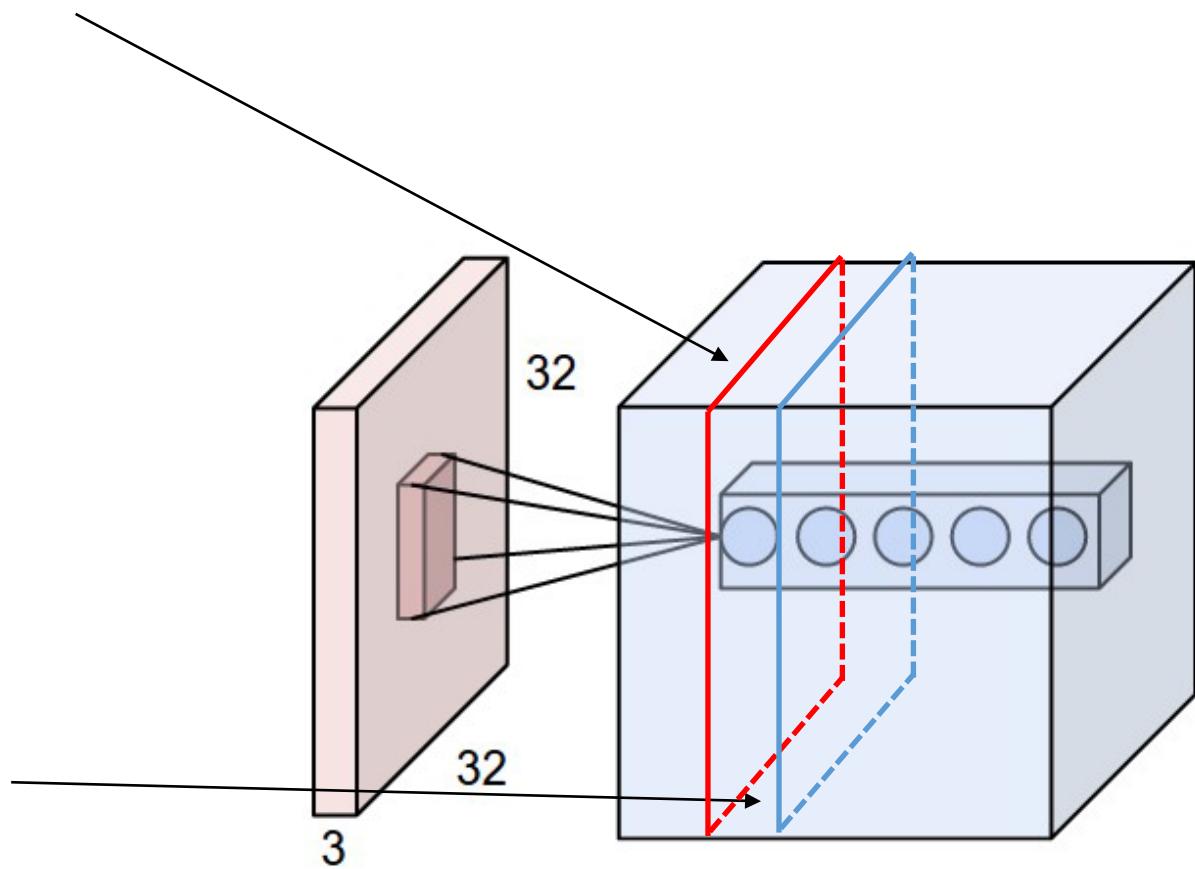
By Aphex34 - Own work, CC BY-SA 4.0,
<https://commons.wikimedia.org/w/index.php?curid=45659236>

Convolutional Layers

Different filters yield different layers in the output

$$a(r, c, 1) = \sum_{i,j,k} w^1(i, j, k) x(r + i, c + j, k) + b^1$$

$$a(r, c, 2) = \sum_{i,j,k} w^2(i, j, k) x(r + i, c + j, k) + b^2$$



By Aphex34 - Own work, CC BY-SA 4.0,
<https://commons.wikimedia.org/w/index.php?curid=45659236>

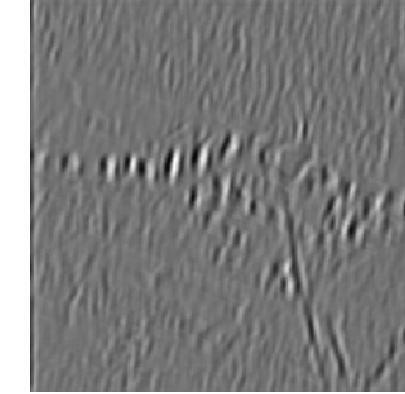
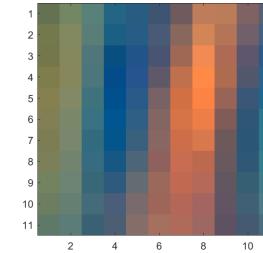
Giacomo Boracchi



Convolutional Layers

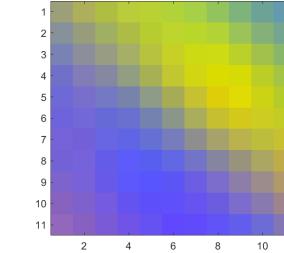


w^1, b^1

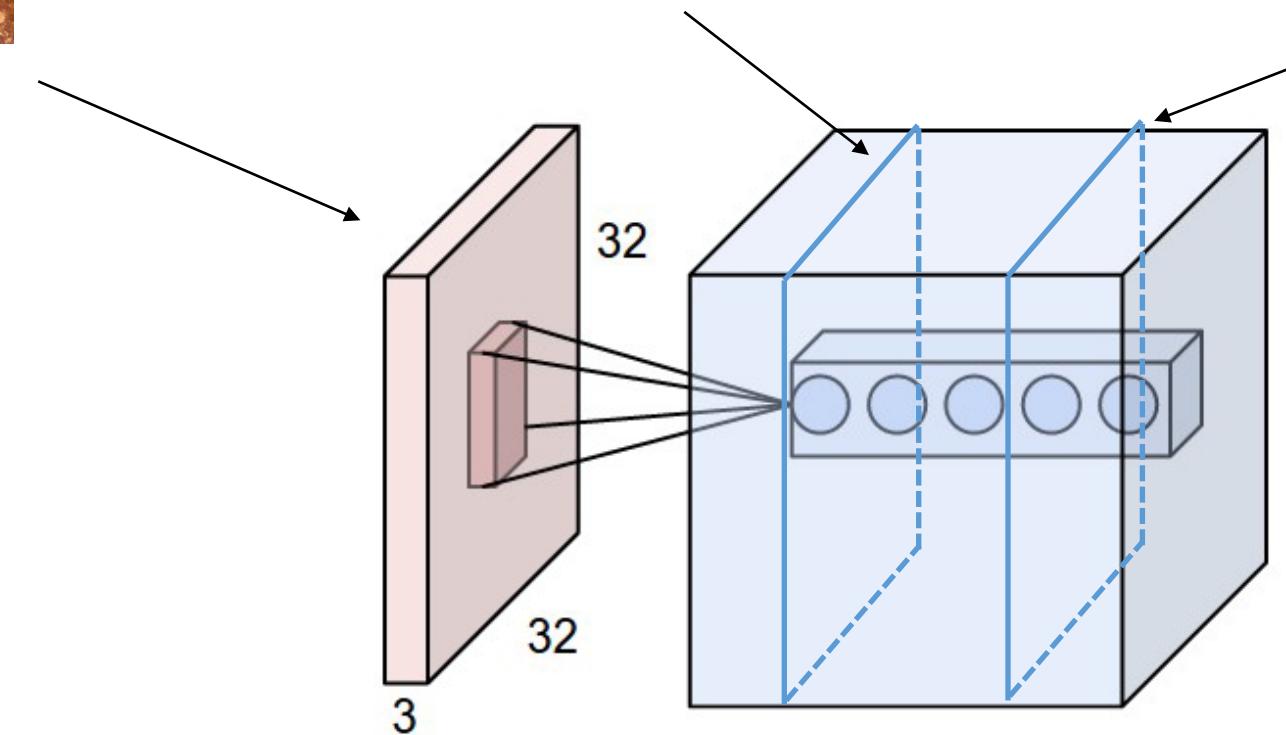


$a(:,:,1)$

w^n, b^n



$a(:,:,n)$



By Aphex34 - Own work, CC BY-SA 4.0,
<https://commons.wikimedia.org/w/index.php?curid=45659236>

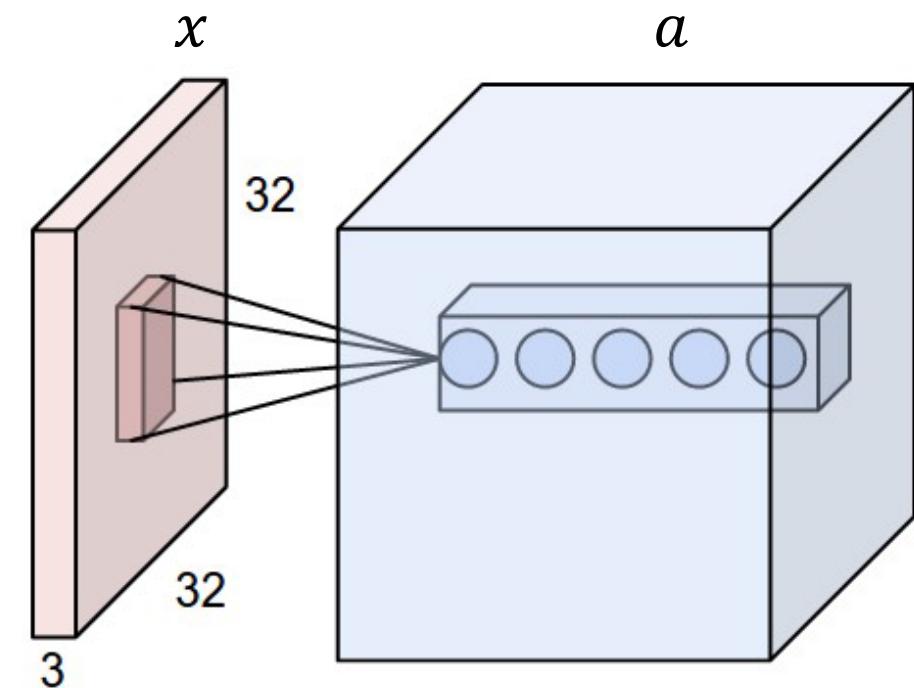
Giacomo Boracchi

Recap: Convolutional Layers

Convolutional layers "mix" all the input components

- The **output** is also called **volume** or **activation maps**
- Each **filter** yields a **different slice** of the output volume
- Each filter has **depth equal to the depth of the input volume** and typically rather **small spatial extent**
- Filters represent the **parameters**

The output of the convolution against a filter becomes a slice in the volume feed to the next layer



By Aphex34 - Own work, CC BY-SA 4.0,
<https://commons.wikimedia.org/w/index.php?curid=45659236>

Giacomo Boracchi

Other Layers

Activation and Pooling

Activation Layers

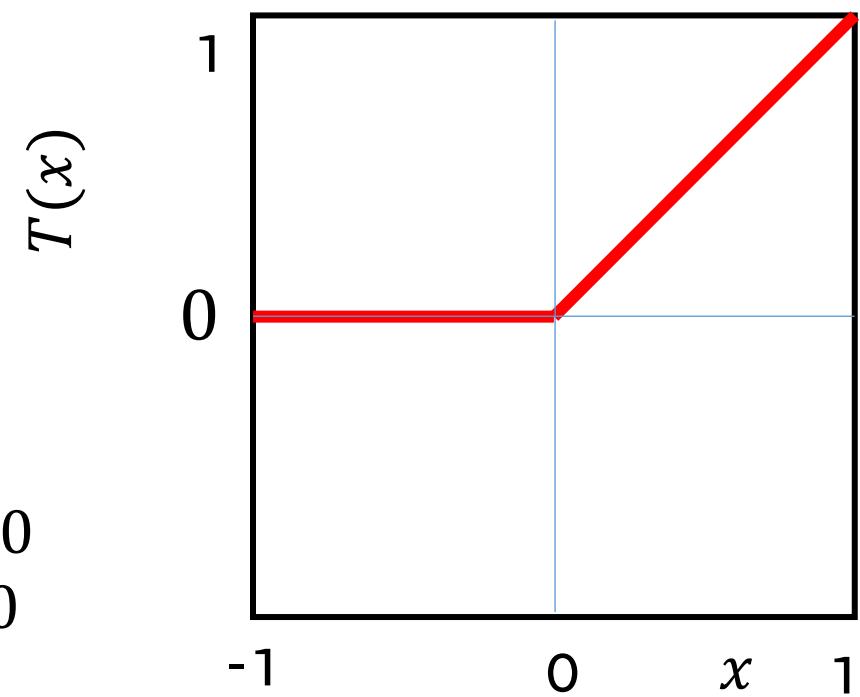
Introduce nonlinearities in the network, otherwise the CNN might be equivalent to a linear classifier...

Activation functions are scalar functions, namely they operate on each single value of the volume. Activations don't change volume size

RELU (Rectifier Linear Units): it's a thresholding on the feature maps, i.e., a $\max(0, \cdot)$ operator.

- By far the most popular activation function in deep NN (since when it has been used in AlexNet)
- Dying neuron problem: a few neurons become insensitive to the input (vanishing gradient problem)

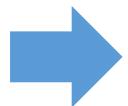
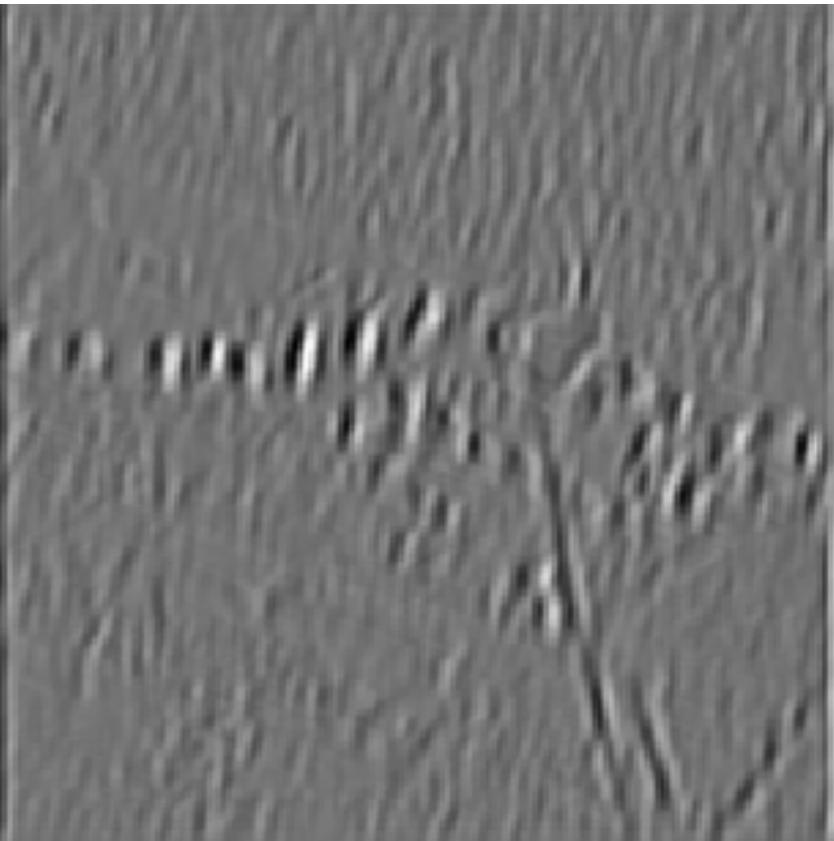
$$T(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$$



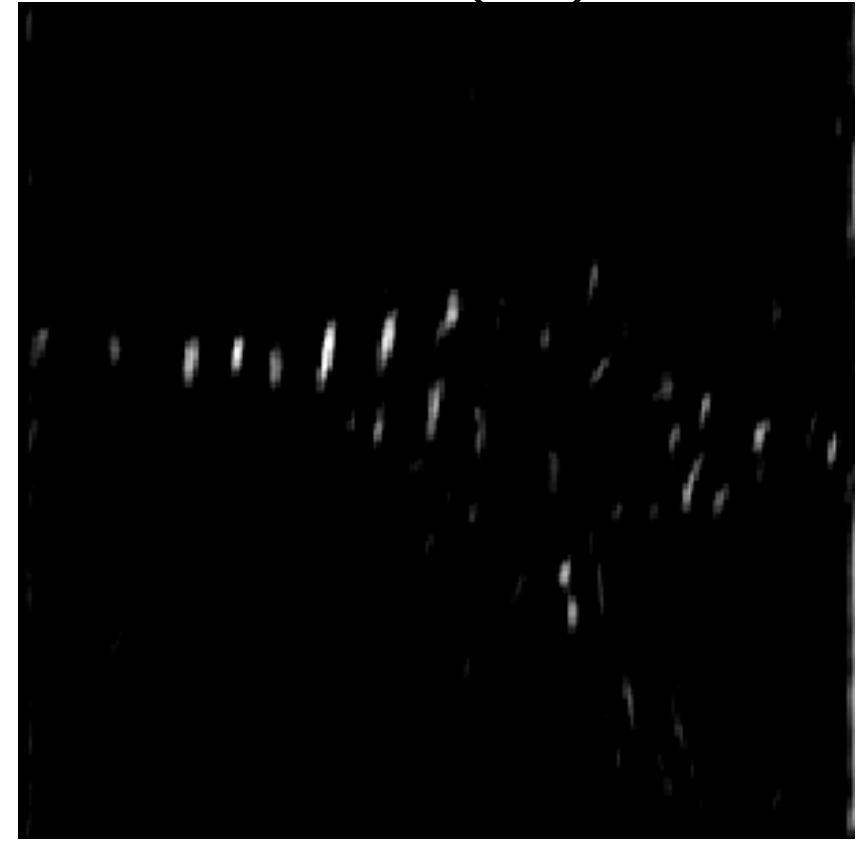
ReLU

Acts separately on each layer

$$a^1$$



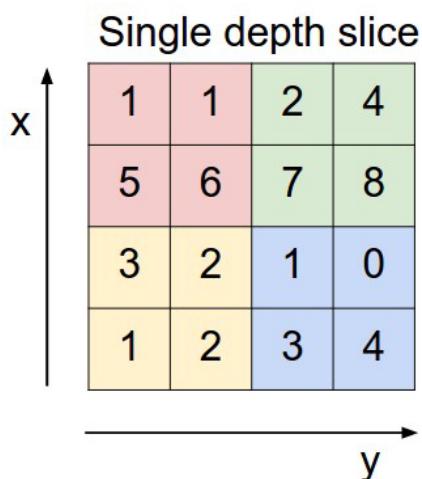
$$\text{ReLU}(a^1)$$



Pooling Layers

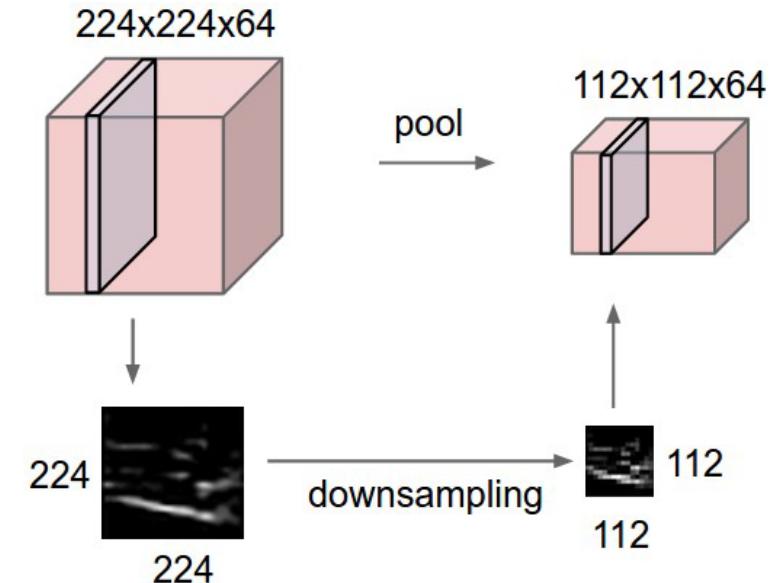
Pooling Layers reduce the spatial size of the volume.

The Pooling Layer operates **independently** on every depth slice of the input and **resizes it spatially**, often using the **MAX operation**.



max pool with 2x2 filters and stride 2

6	8
3	4

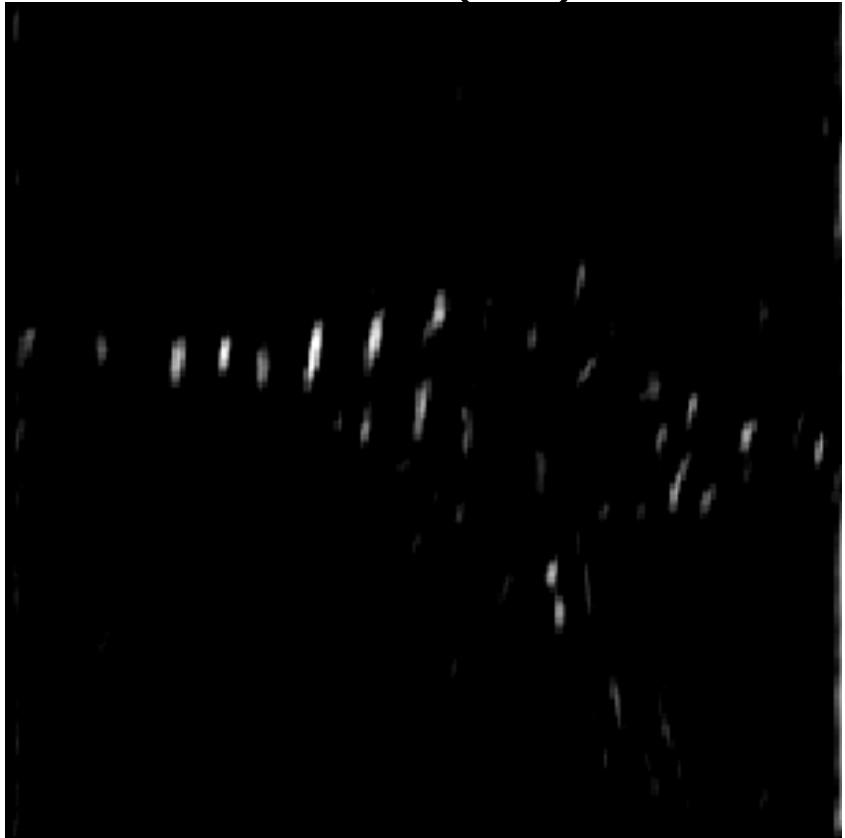


In a 2x2 support it discards 75% of samples in a volume

Max-Pooling (MP)

Acts separately on each layer

$$ReLU(a^1)$$

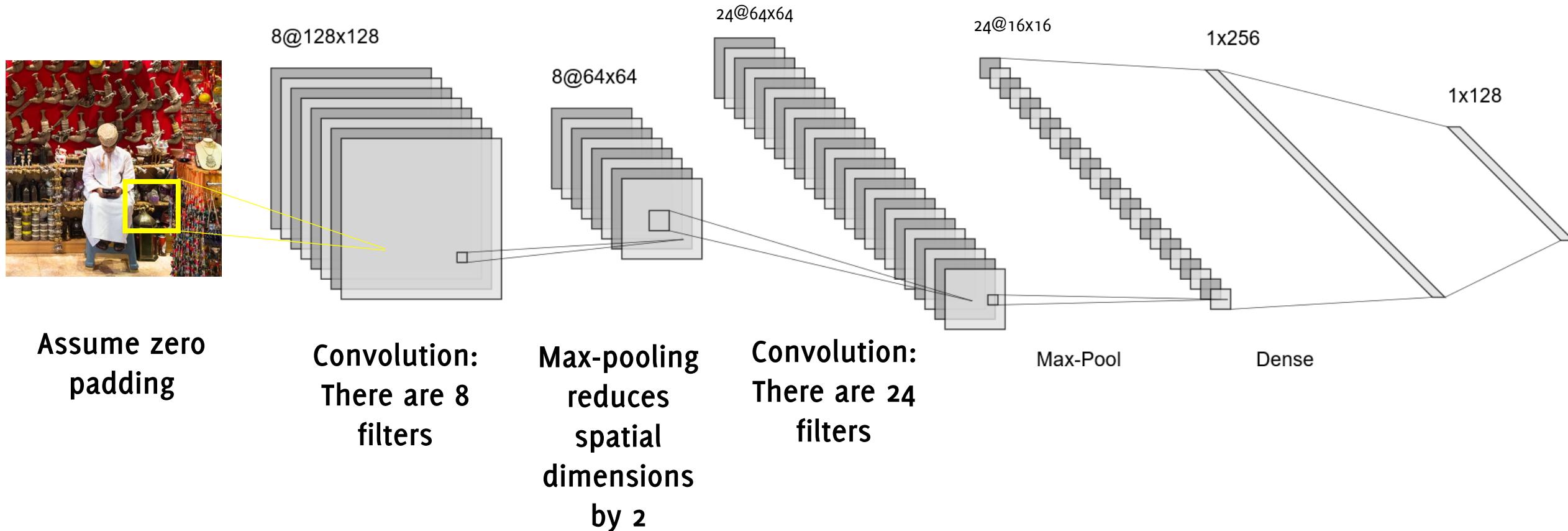


$$\text{MP}(ReLU(a^1))$$



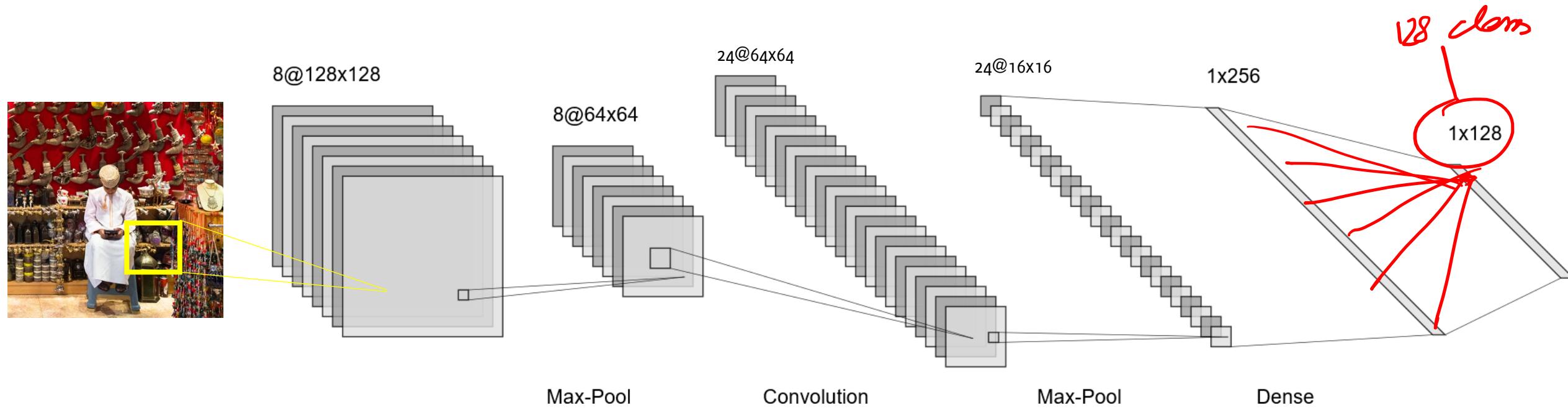
Fully Connected Layers

The typical architecture of a convolutional neural network



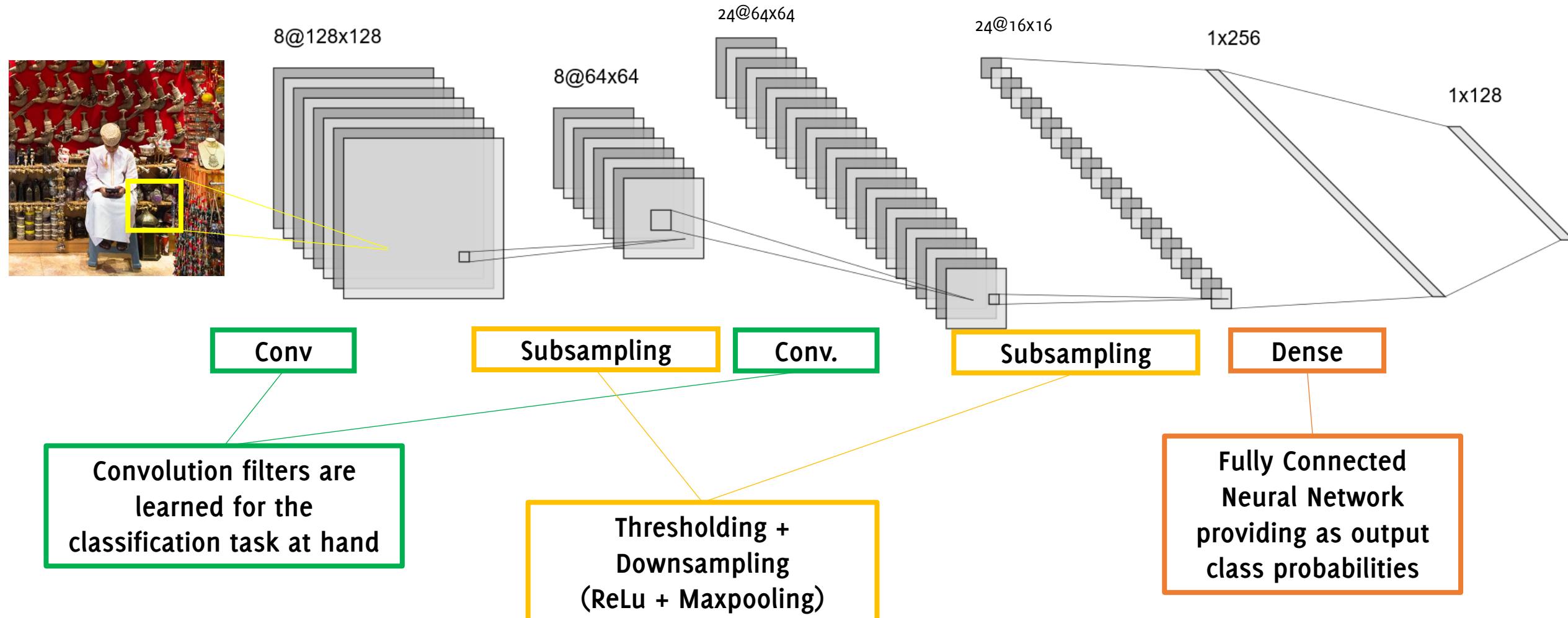
Fully Connected Layers

The typical architecture of a convolutional neural network



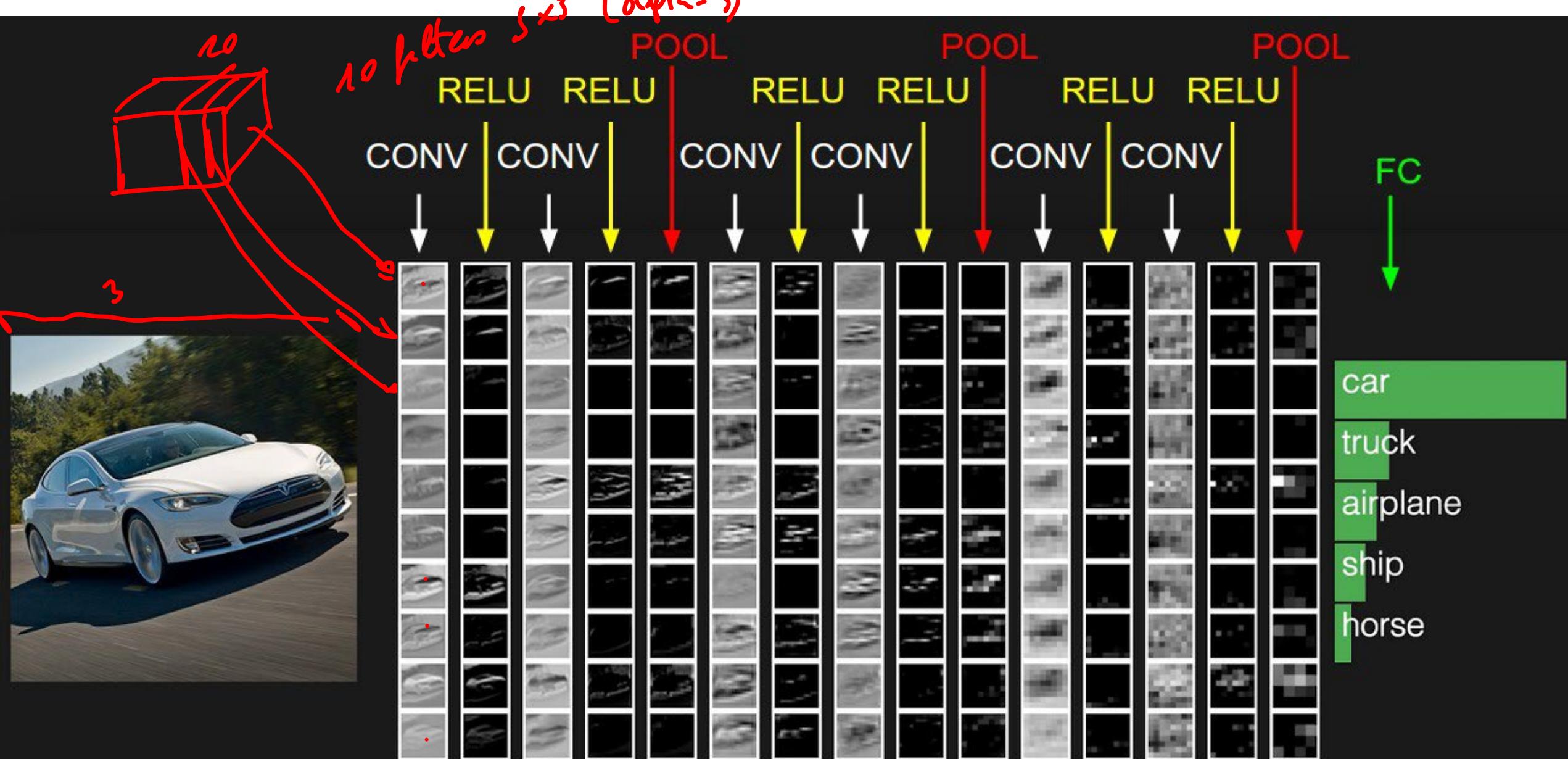
The output of the **fully connected (FC) layer has the same size as the number of classes**, and provides a score for the input image to belong to each class

Convolutional Neural Networks (CNN)



CNN «in action»

Activations in a convolutional network

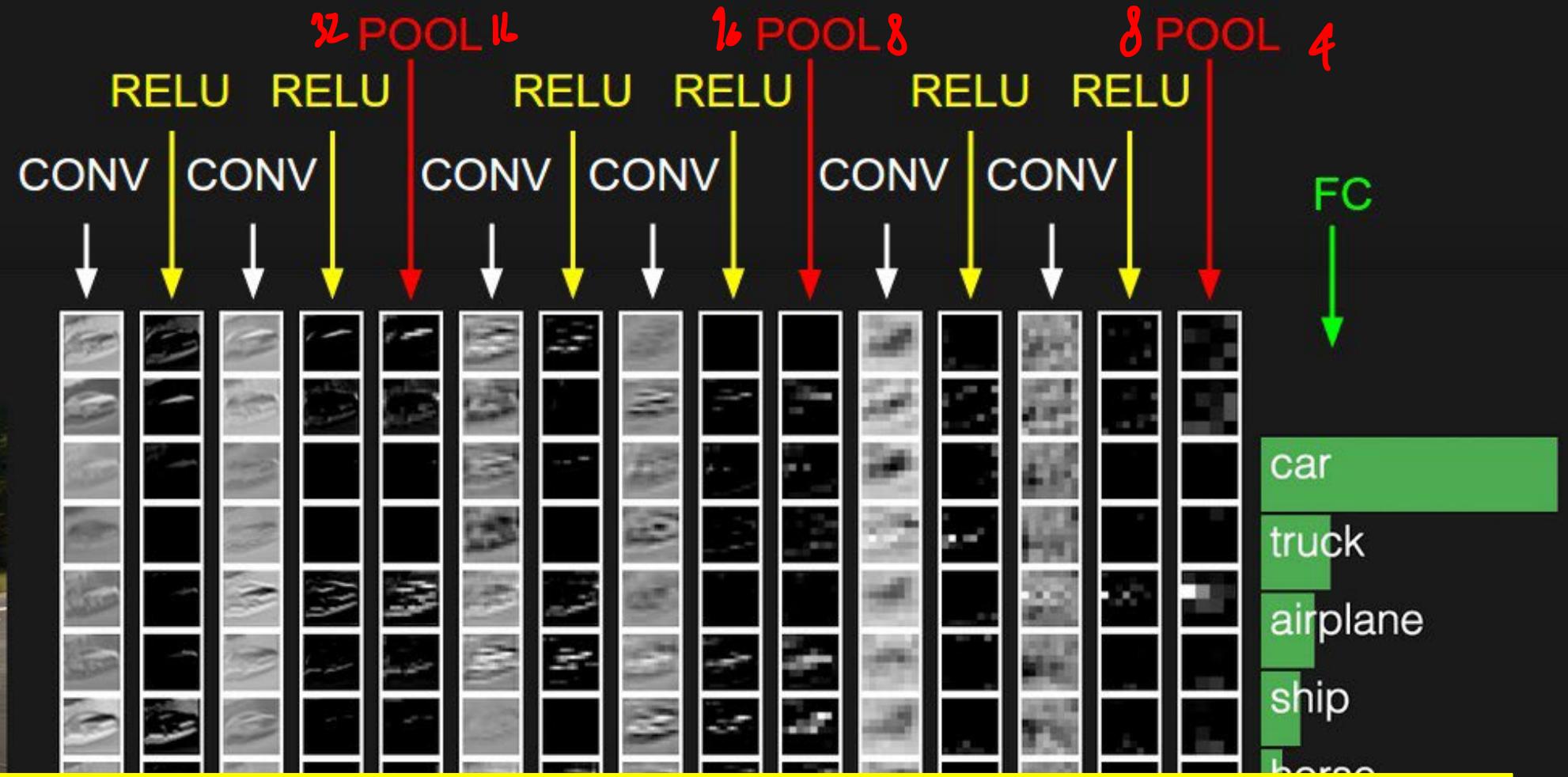




Activations in a convolutional network

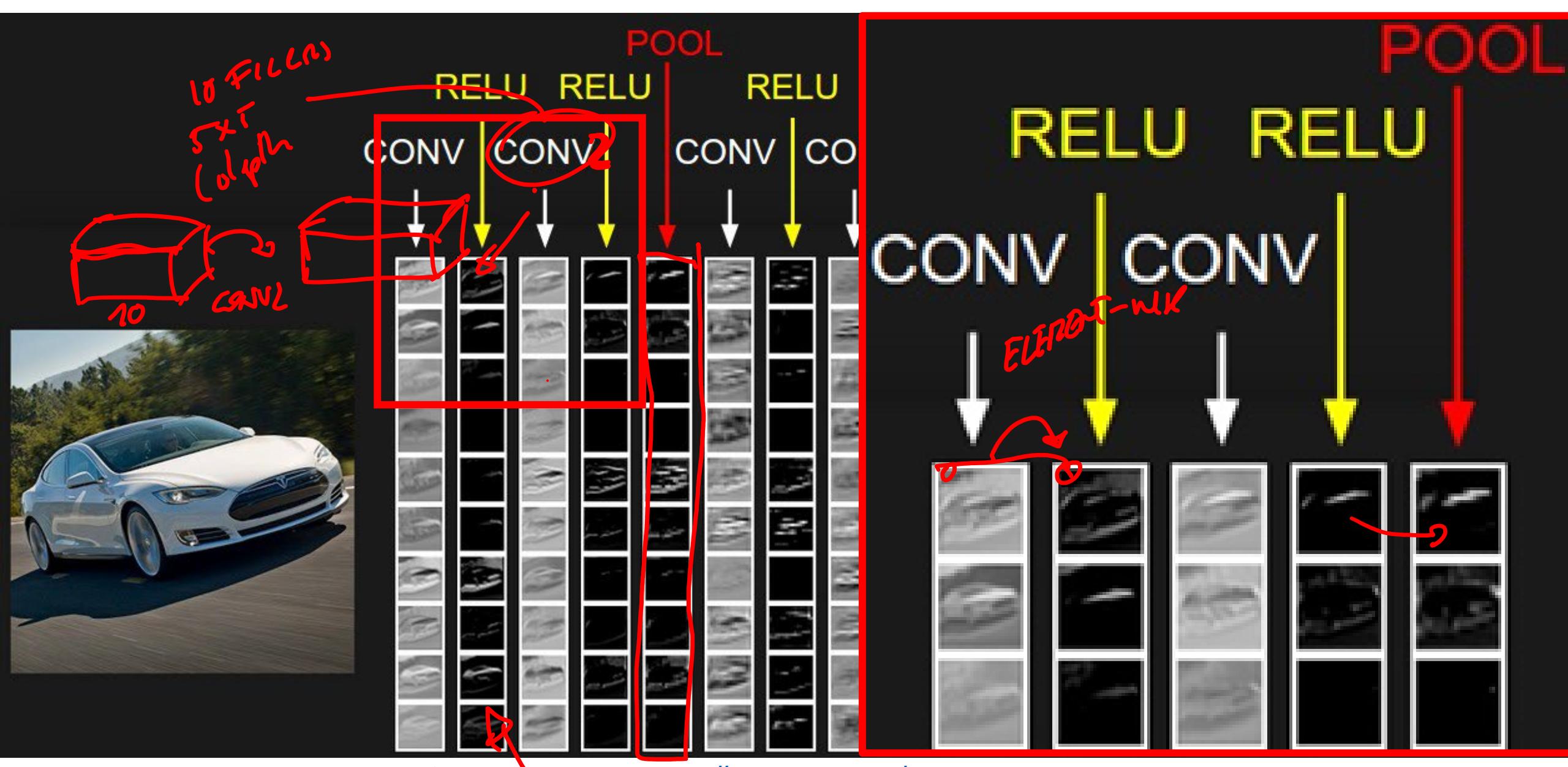


32x32

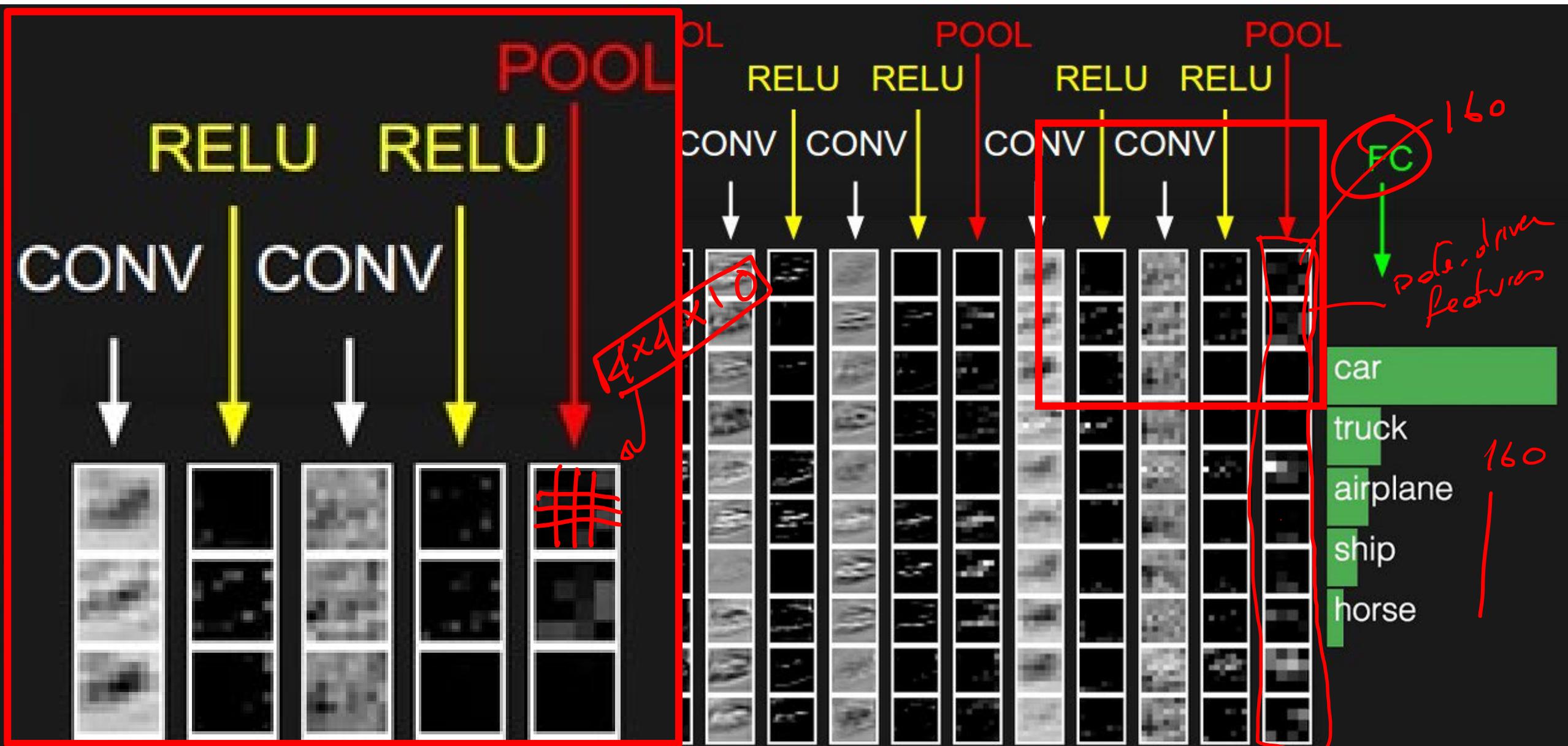


Each layer in the volume is represented as an image here
(using the same size but different resolution for visualization sake)

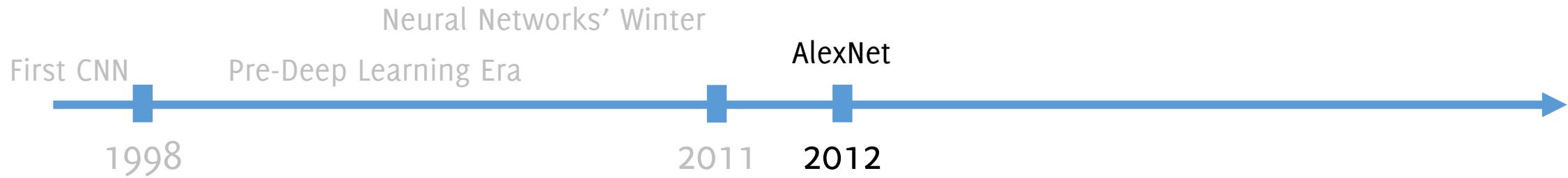
Activations in a convolutional network



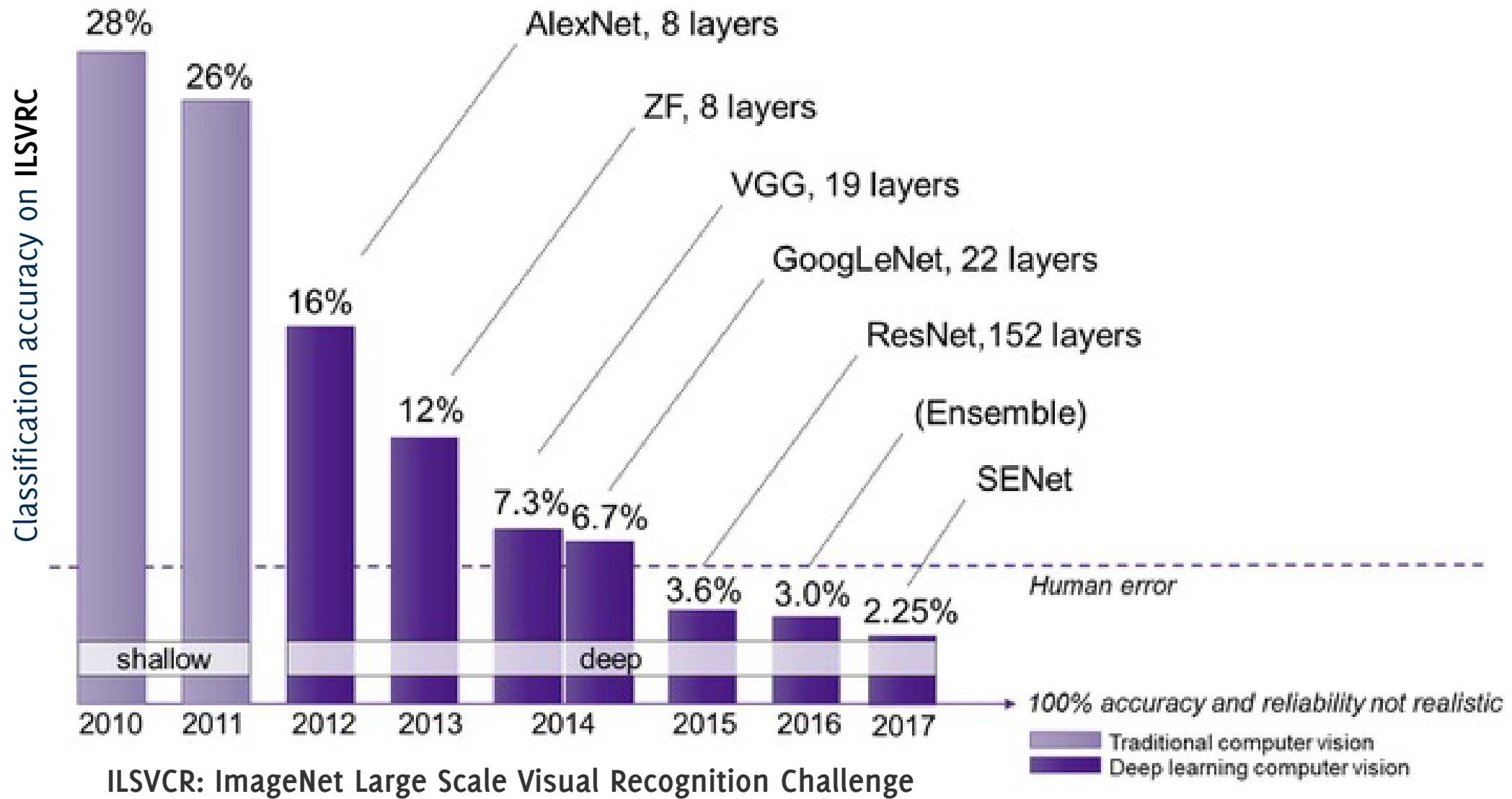
Activations in a convolutional network



The Return of CNNs



The impact of Deep Learning in Visual Recognition



ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky

University of Toronto

kriz@cs.utoronto.ca

Ilya Sutskever

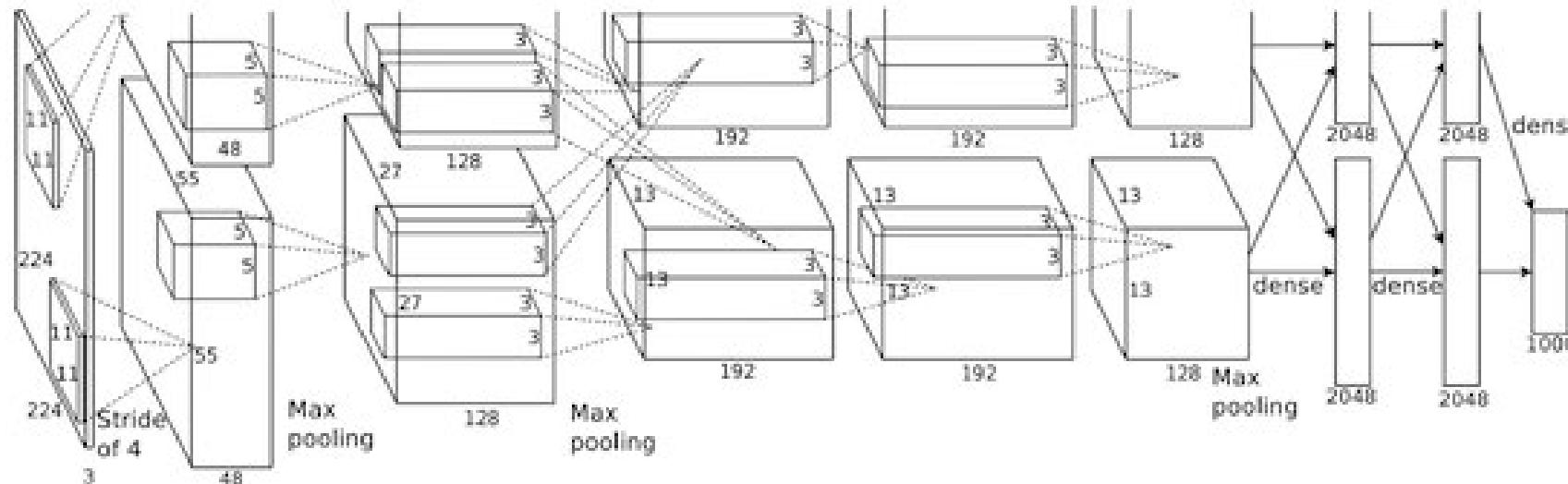
University of Toronto

ilya@cs.utoronto.ca

Geoffrey E. Hinton

University of Toronto

hinton@cs.utoronto.ca



Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012.

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky

University of Toronto

kriz@cs.utoronto.ca

Ilya Sutskever

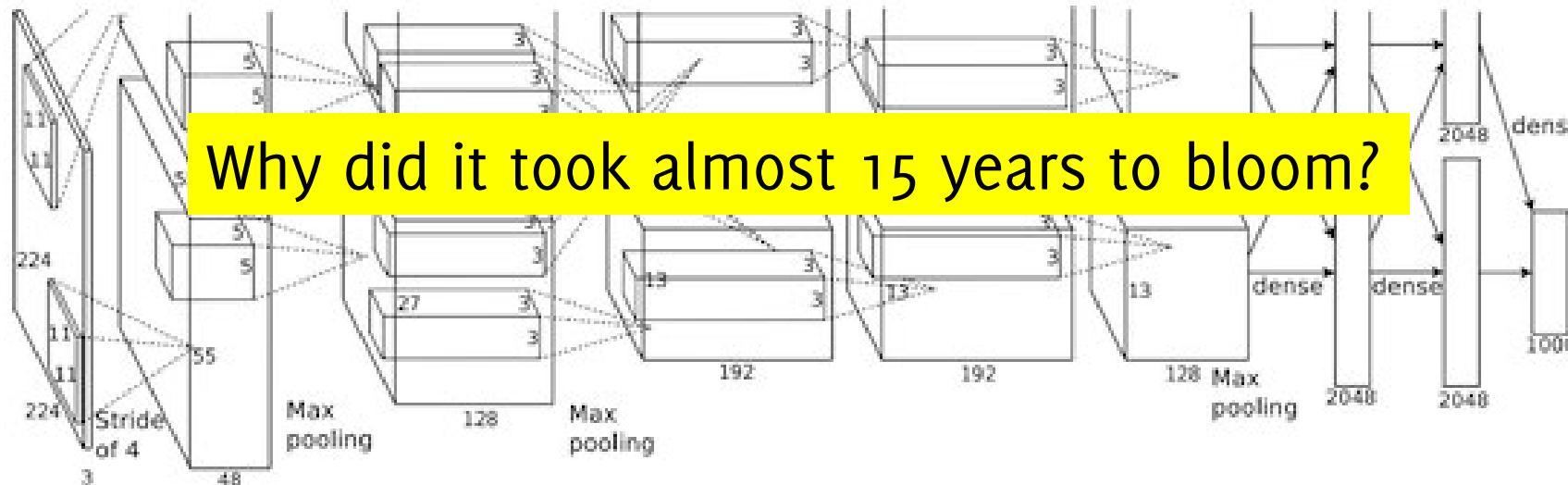
University of Toronto

ilya@cs.utoronto.ca

Geoffrey E. Hinton

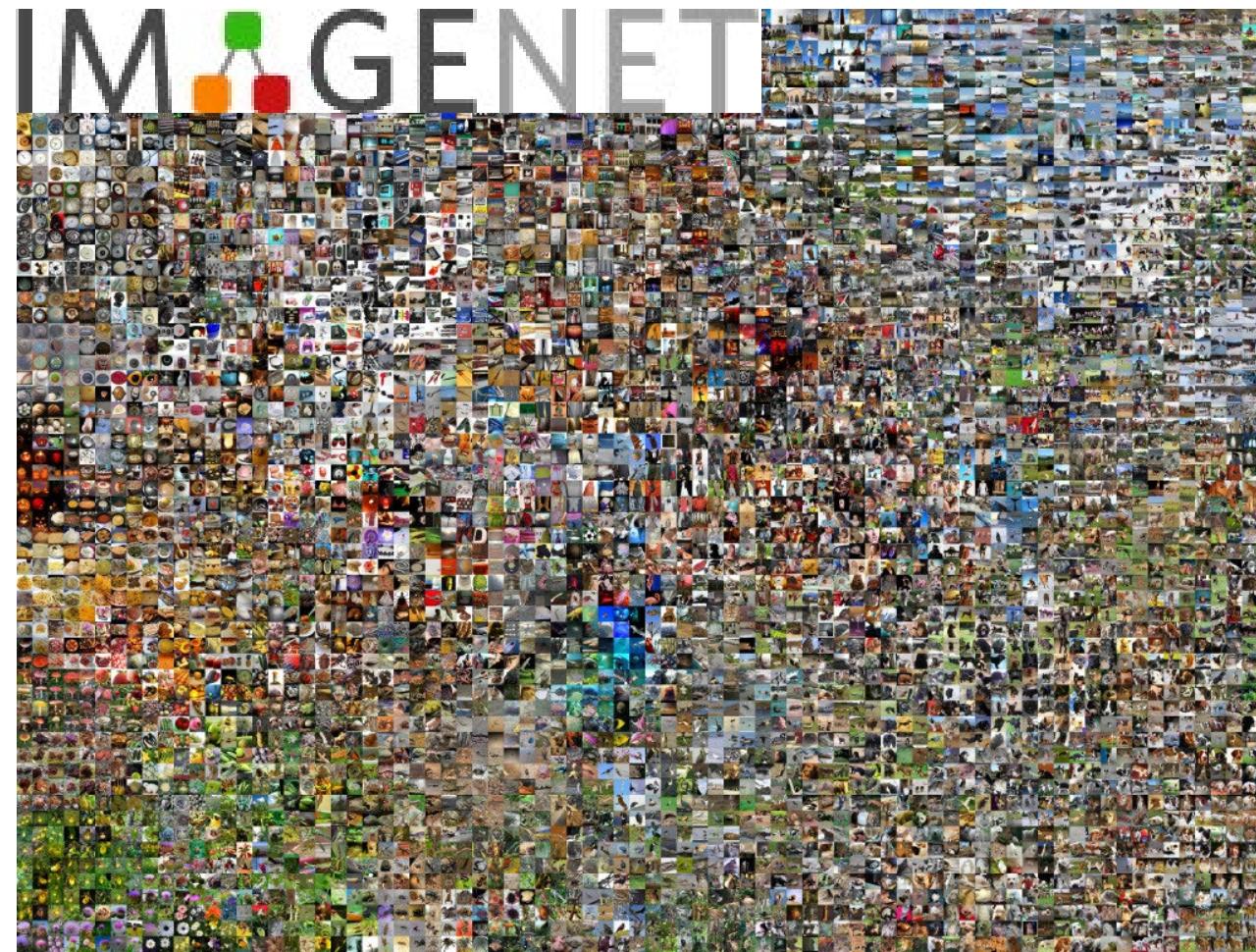
University of Toronto

hinton@cs.utoronto.ca



Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012.

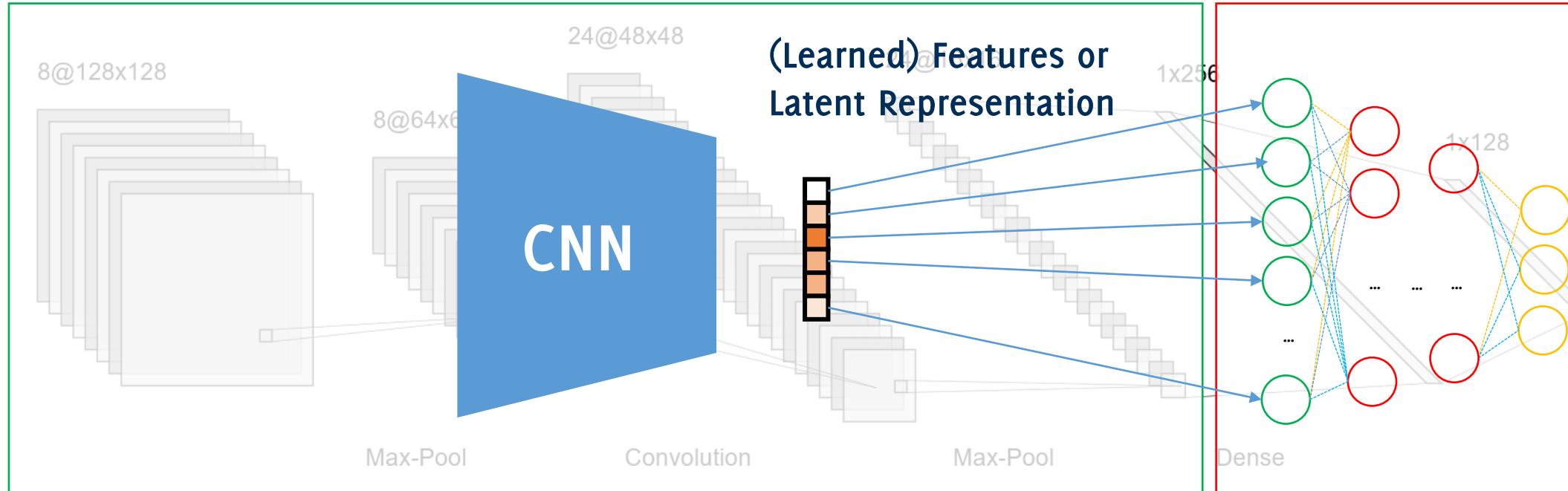
Key Enabling factors for CNNs: Large Collections of Annotated Data & Parallel Computing



<https://www.flickr.com/photos/nvidia/34686550412>

Data Driven Features

CNNs as powerful feature extractors

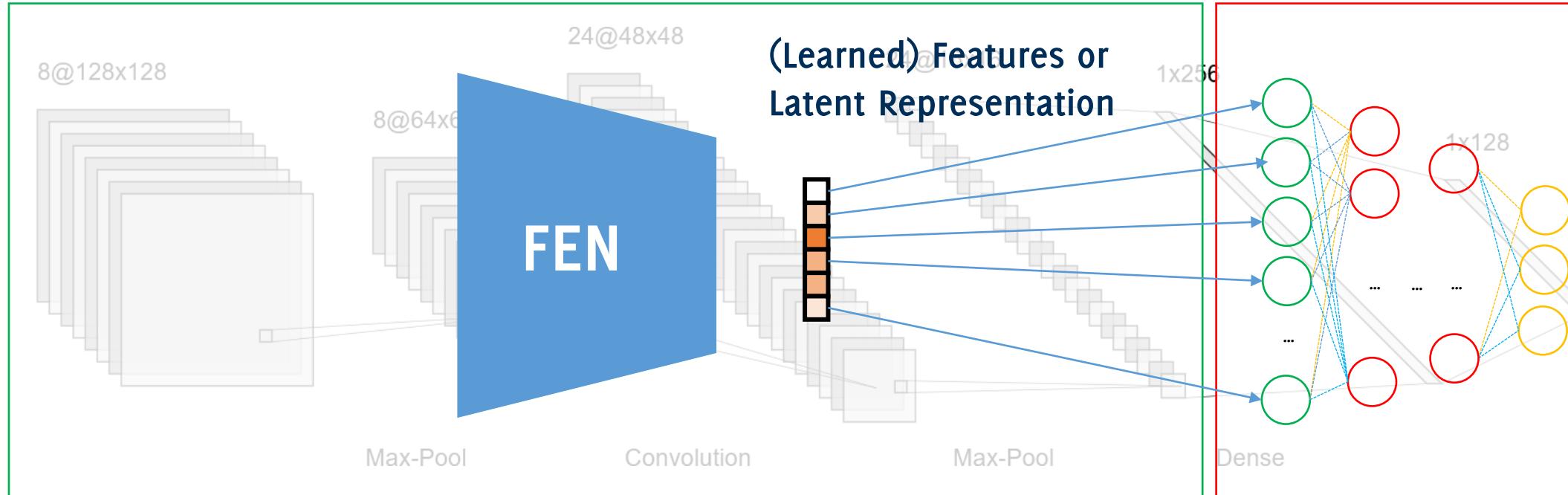


Data-Driven Feature
extraction

Feature
Classification

Typically, to learn meaningful representations, many layers are required
The network becomes deep

Feature Extraction Networks



Data-Driven Feature
extraction

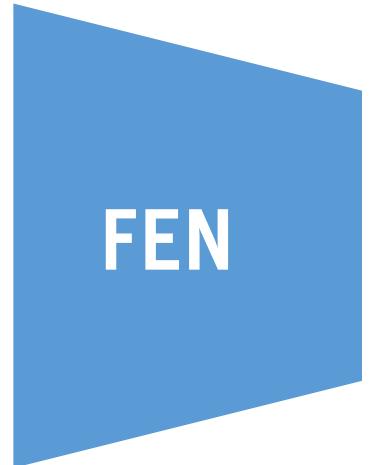
Feature
Classification

FEN: FEATURE EXTRACTION NETWORK, the convolutional block of CNN

Feature Extraction Networks

The new paradigm to solve visual recognition problems

- Instead of engineering features
- Train a CNN in an end-to-end manner and you get better features that optimized for solving the problem at-hand



Key advantages:

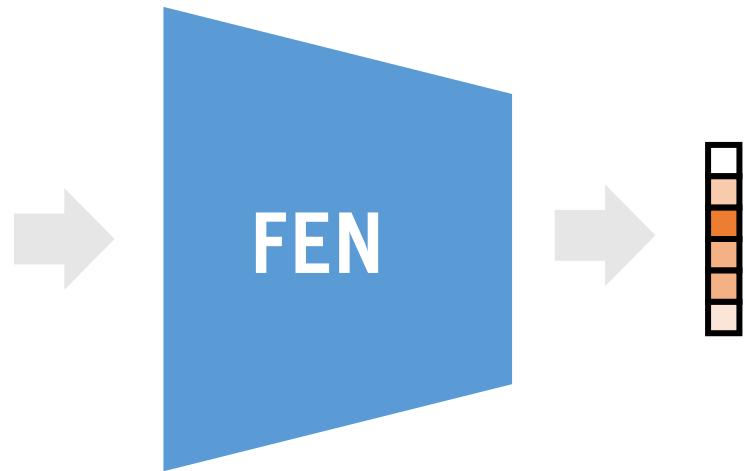
- **Everything** (feature extraction and classification) is optimized for improving the task at hand.
- **End-to-end trainable** solutions require no experts, just annotated data.
- Plenty of high-level frameworks (Keras, Tensorflow, PyTorch, TensorFlow Lite) that allows solving complex visual recognition by simply **programming black-boxes**.
- Democratisation of Computer Vision!
- Very effective...

Using Data-Driven Features

Features are Good For Image Retrieval

Feed a test image and compute its latent representation

Test image



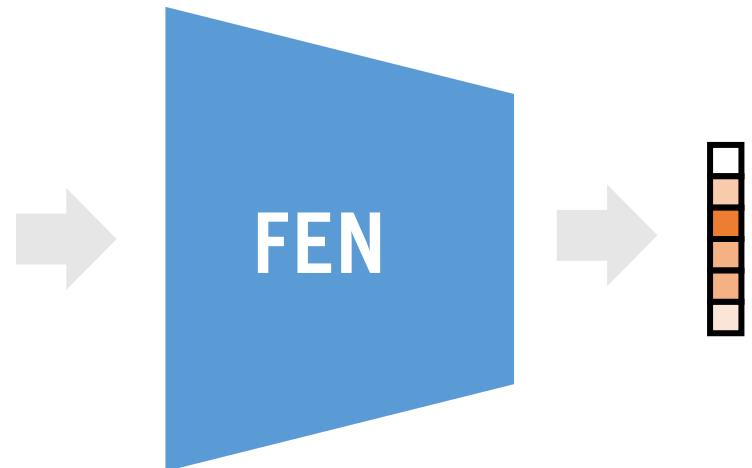
Latent Representation:
Data-Driven Feature Vector

Retrieve the training images having the closest latent representation

Features are Good For Image Retrieval

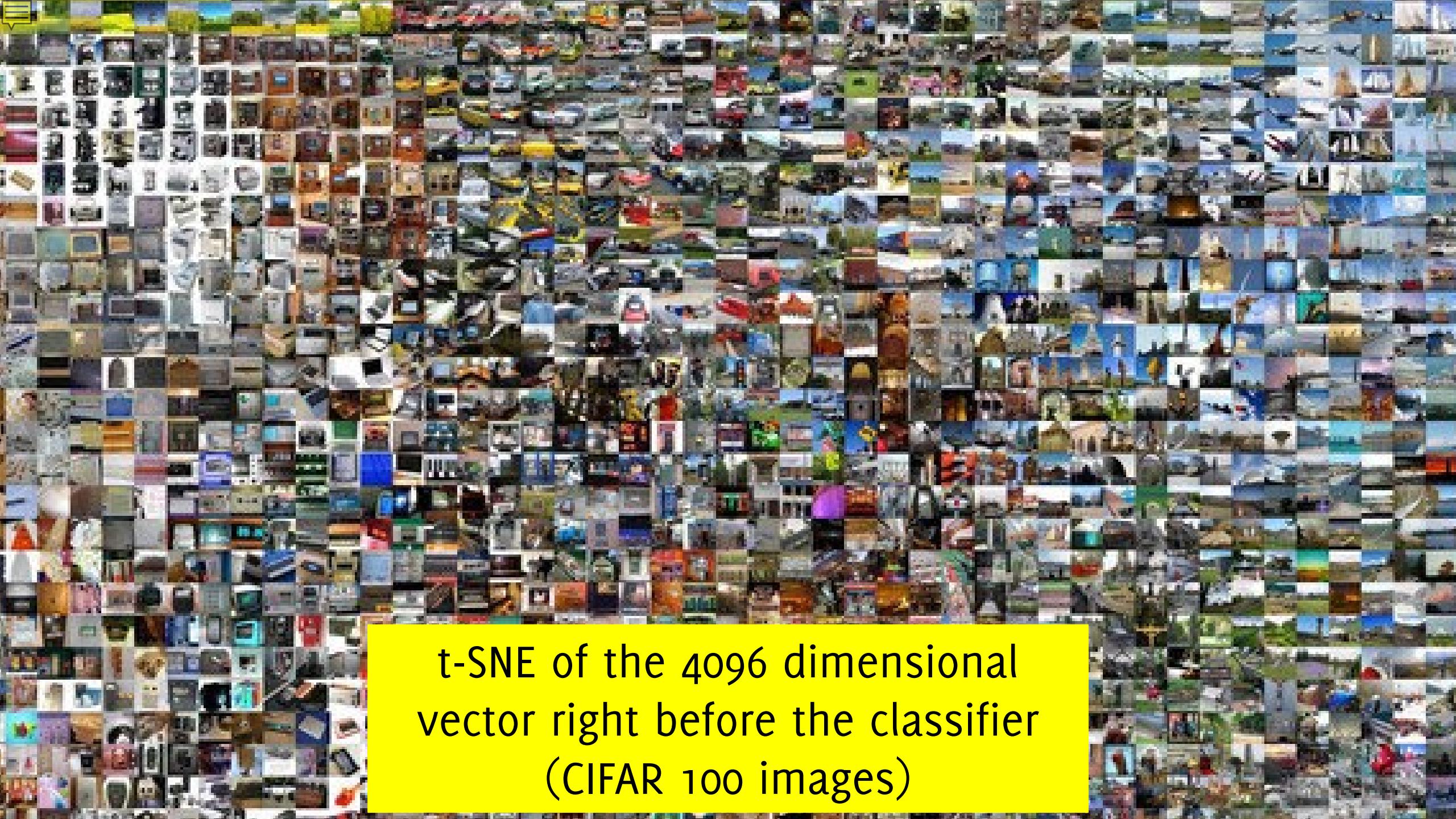
Feed a test image and compute its latent representation

Test image

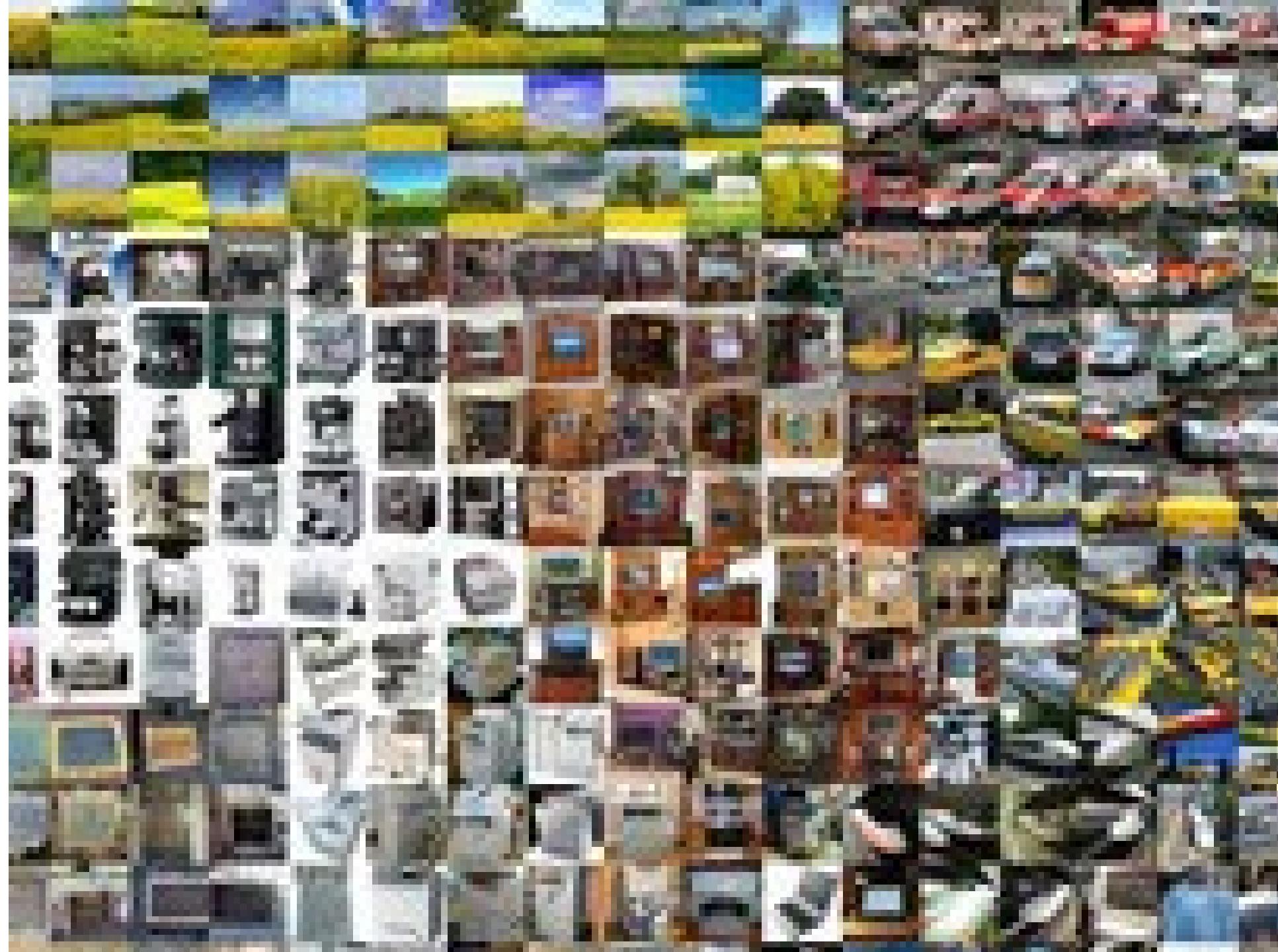


Training Images corresponding to the closest latent representations!

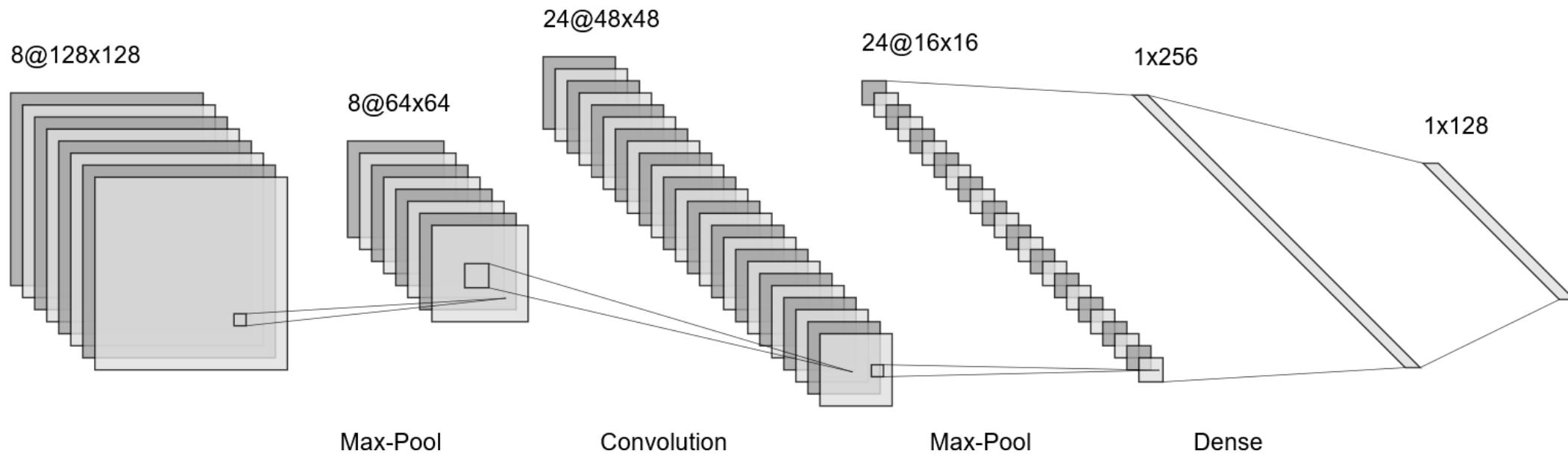




t-SNE of the 4096 dimensional
vector right before the classifier
(CIFAR 100 images)



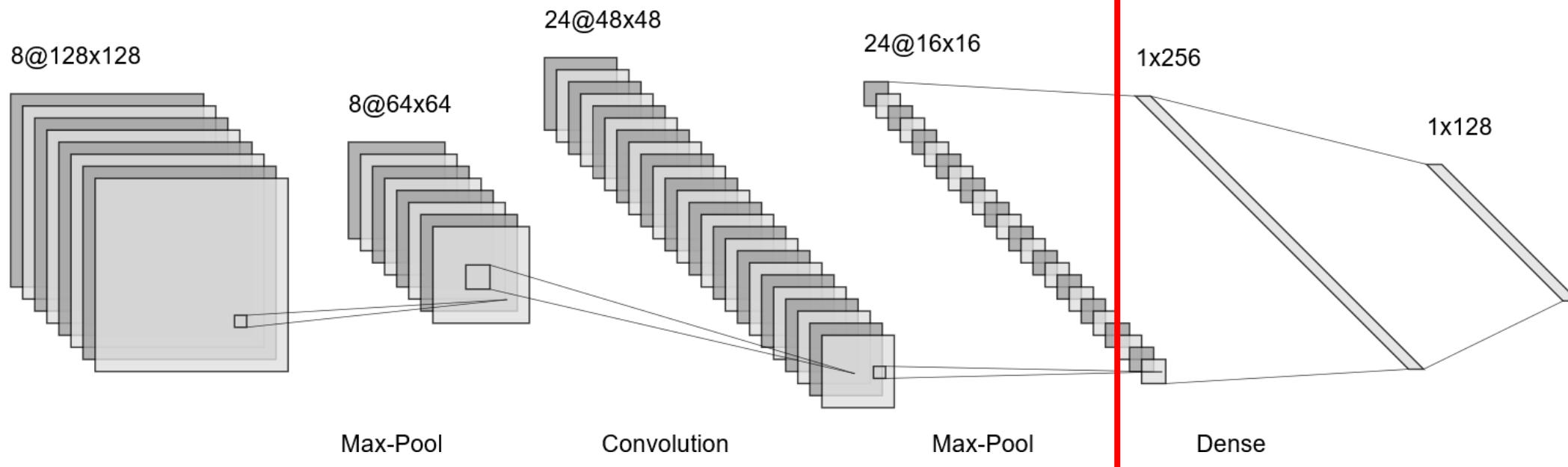
Features are good at solving other tasks



Data-Driven Feature extraction are «general purpose»

Feature Classification is
«task specific»

Features are good at solving other tasks

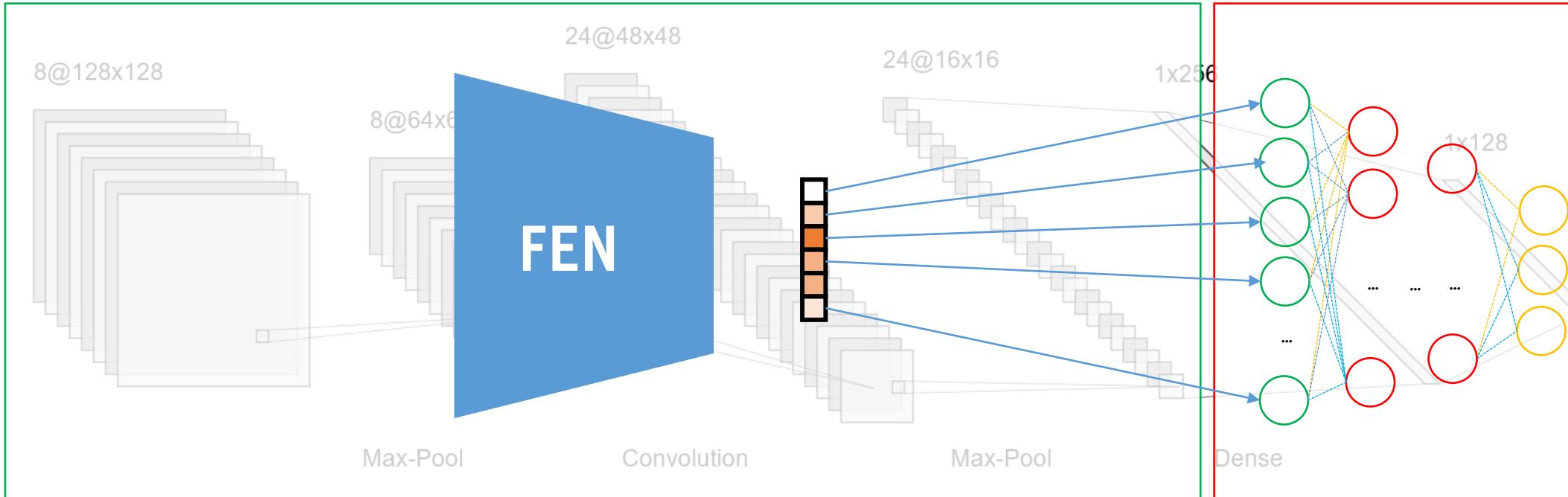


Data-Driven Feature extraction are «general purpose»

Feature Classification is
«task specific»

You might want to get rid
of this when solving a
different task

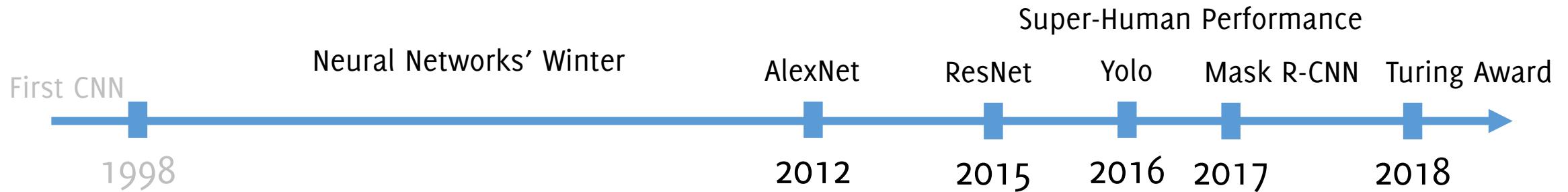
Transfer Learning



1. Freeze the FEN
2. Replace the Feature Classifier to solve the task at hand
3. Train / Fine tune the entire network (supervised) on the new task

**Feature
Classification**

A Steadily Increasing Power to Solve Visual Recognition Problems





This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the version available on IEEE Xplore.

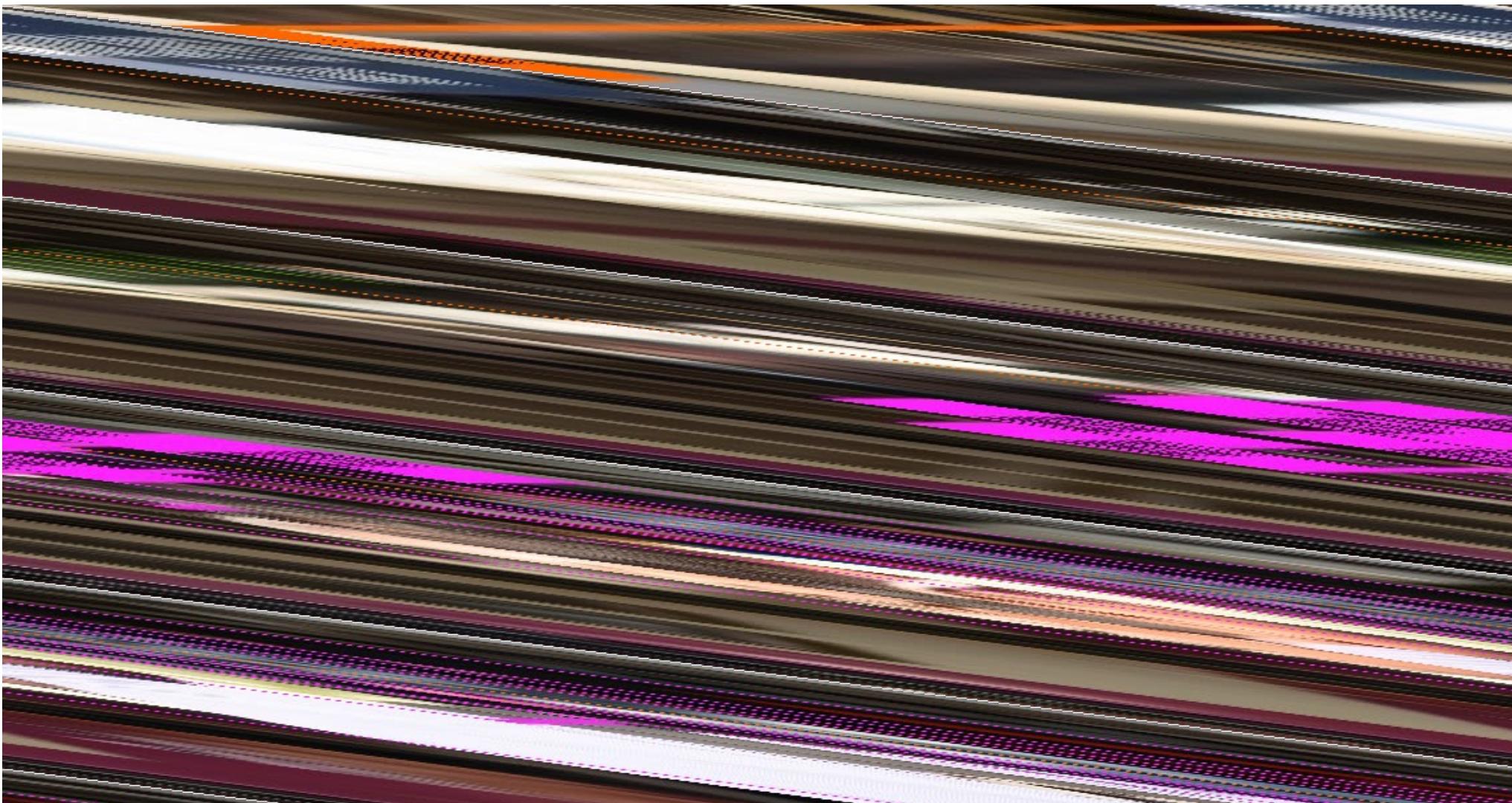
You Only Look Once: Unified, Real-Time Object Detection

Joseph Redmon*, Santosh Divvala*[†], Ross Girshick[¶], Ali Farhadi*[†]

University of Washington*, Allen Institute for AI[†], Facebook AI Research[¶]

<http://pjreddie.com/yolo/>

Object Detection (since 2015)





This ICCV paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the version available on IEEE Xplore.

Mask R-CNN

Kaiming He Georgia Gkioxari Piotr Dollár Ross Girshick
Facebook AI Research (FAIR)

Object Detection vs Instance Segmentation

**Object
Detection**



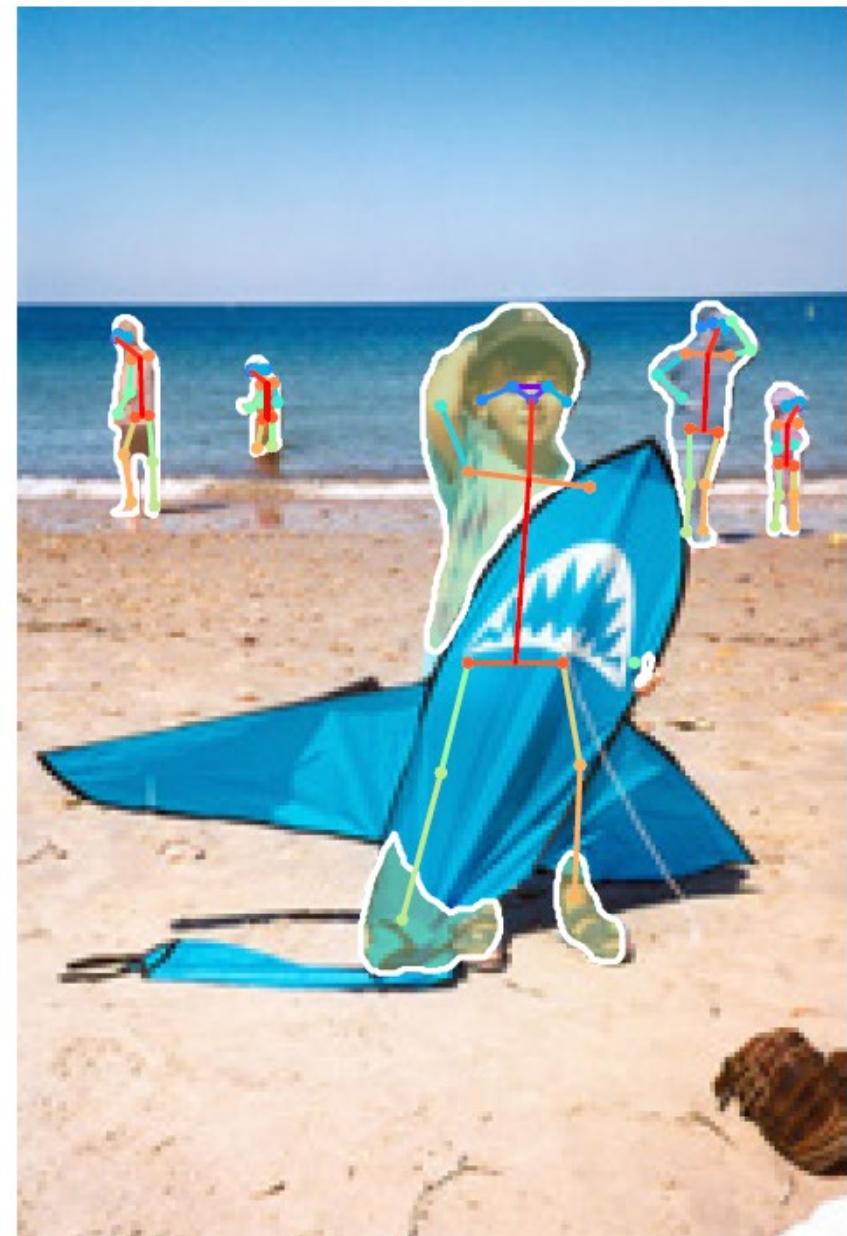
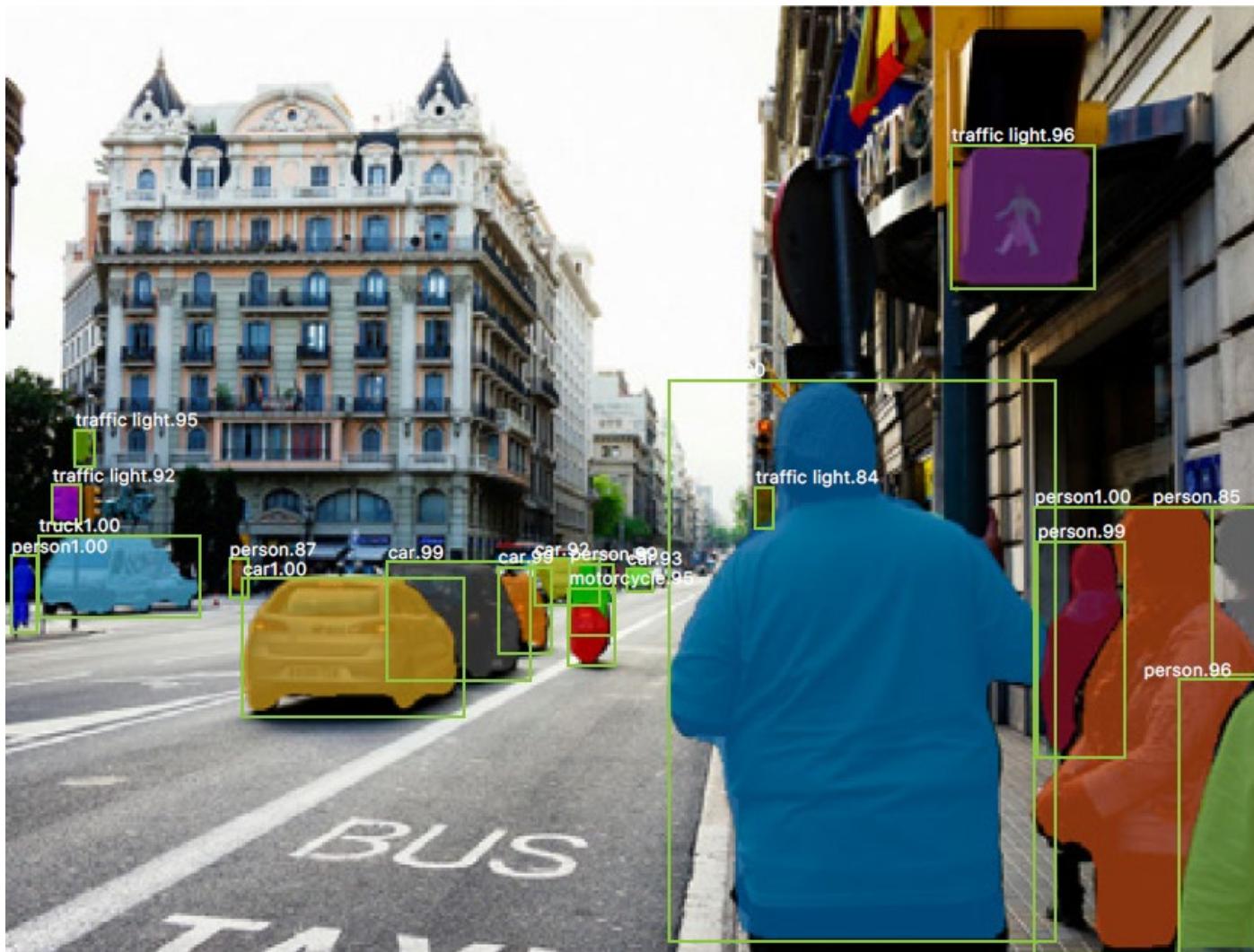
**Instance
Segmentation**



Fathers of the Deep Learning Revolution Receive ACM A.M. Turing Award

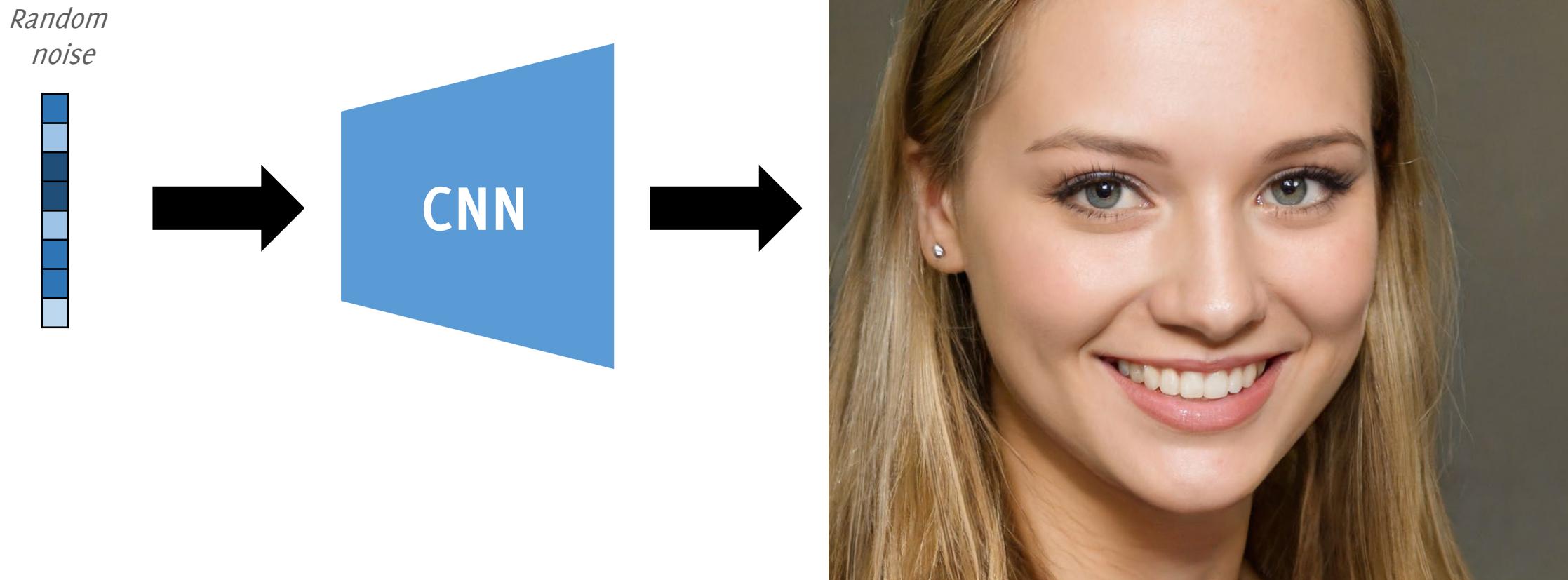
Bengio, Hinton and LeCun Ushered in Major Breakthroughs in Artificial Intelligence

Instance Segmentation / Human Pose Estimation



GAN: Generative Adversarial Networks

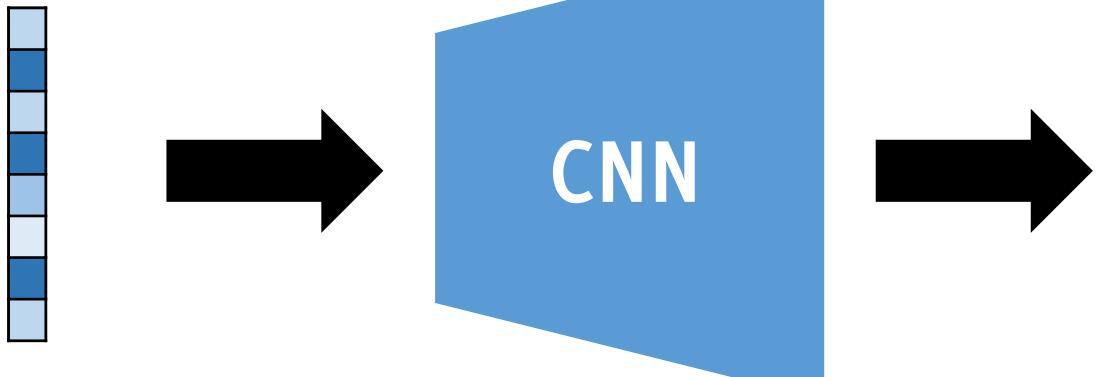
<https://thispersondoesnotexist.com>



GAN: Generative Adversarial Networks

<https://thispersondoesnotexist.com>

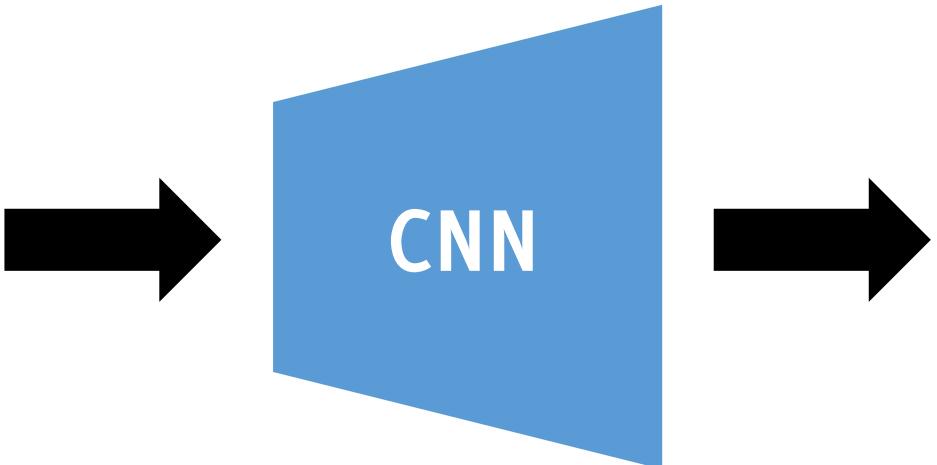
Random
noise



GAN: Generative Adversarial Networks

<https://thispersondoesnotexist.com>

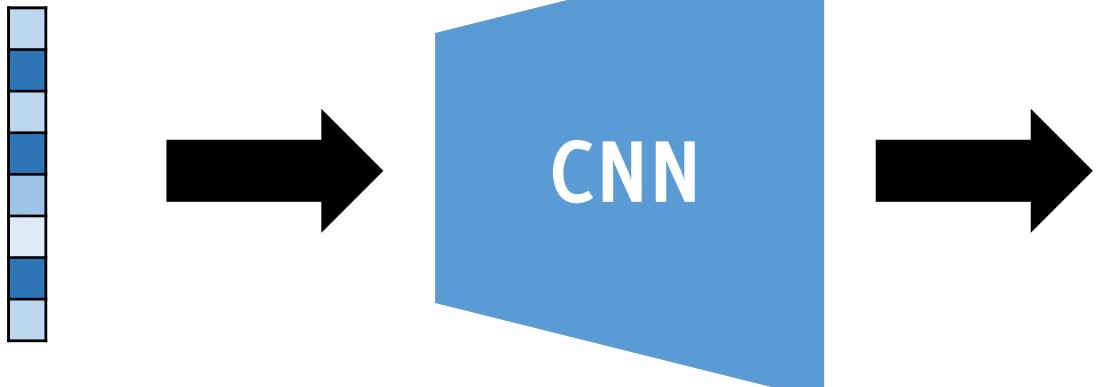
Random
noise



GAN: Generative Adversarial Networks

<https://thispersondoesnotexist.com>

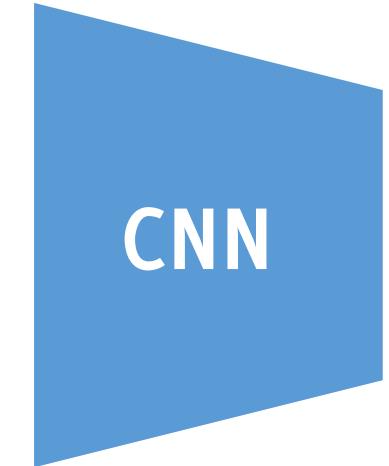
Random
noise



Why CNNs are so good?

Stacks of trained convolutional layers have become the dominant paradigm to solve visual recognition problems

- Neural Networks are very powerful function approximation
- We know how to train these on huge amount of data



CNNs have **built-in inductive biases** that are image-specific:

- **Spatial bias** (locality): convolutions operate on contiguous image regions, thus the resulting features.
- **Translation equivariance bias**: convolutions operate the same in all the image regions. Very important for handling high resolution images or solving object detection problems

On top of that, they enable **parallel and efficient processing**.

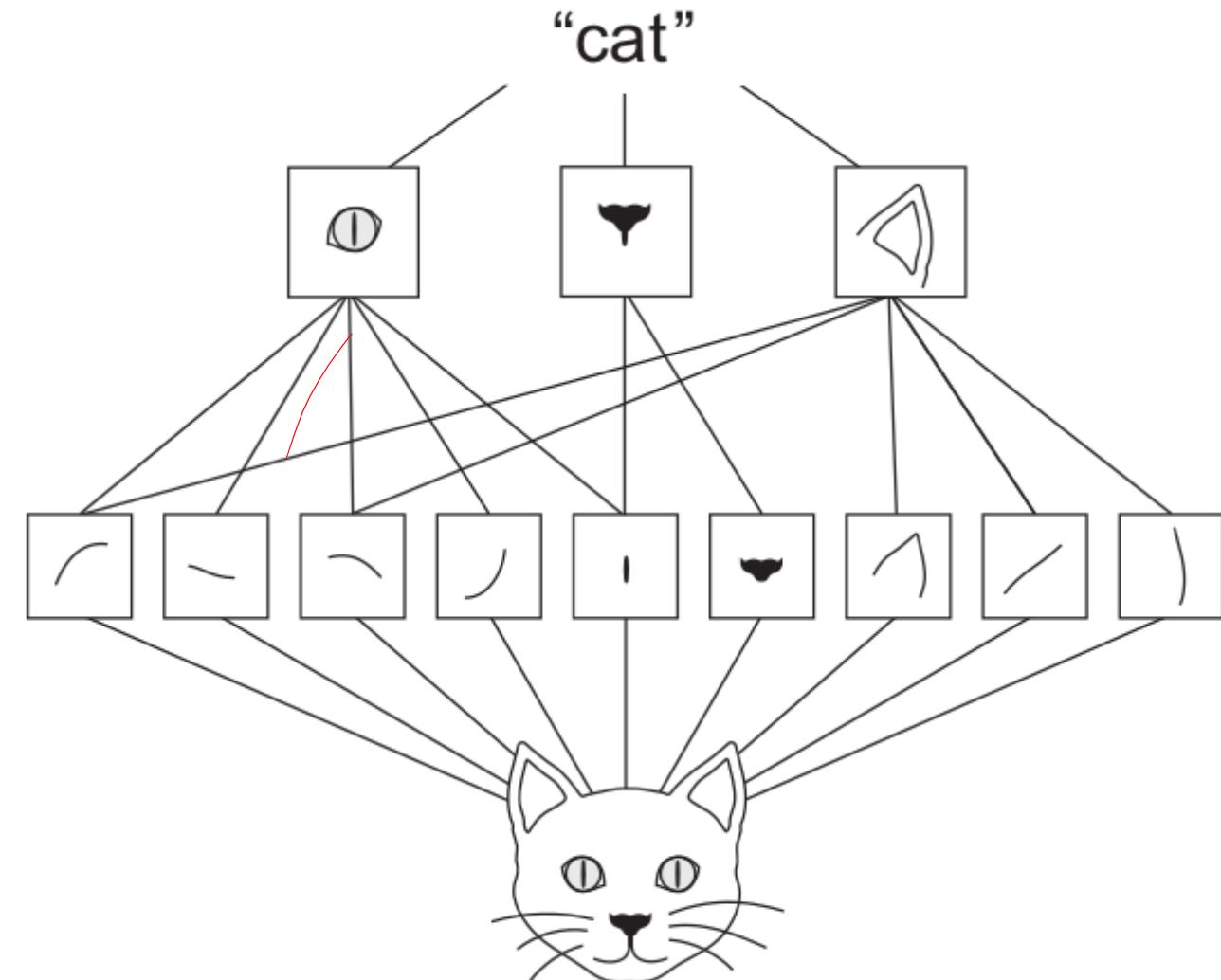
Inductive Biases in CNN

As we move to deeper layers:

- spatial resolution is reduced
- the number of maps increases

We search for **higher-level patterns**,
and **don't care too much about their
exact location**.

There are more high-level patterns
than low-level details!

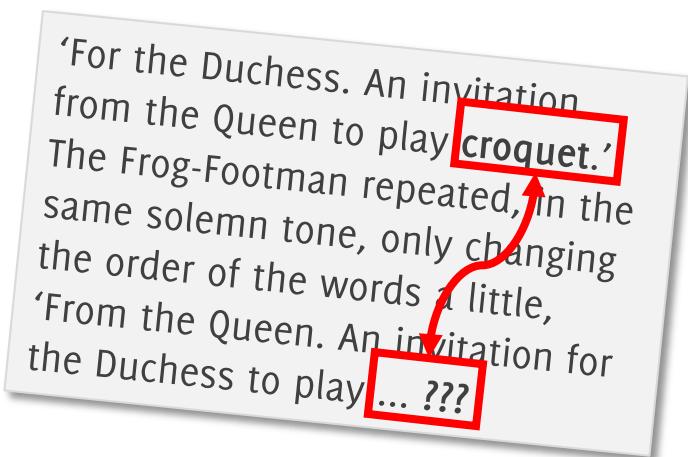


How can CNN exploit Long-Distance Dependencies?

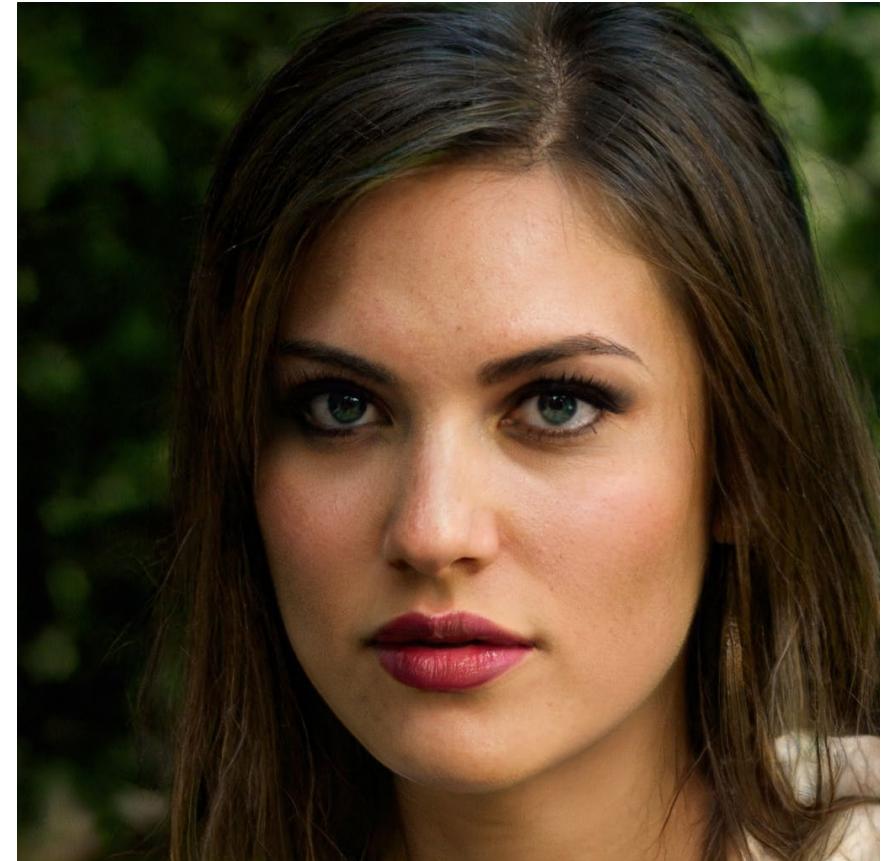
Long-Distance Dependencies

These are fundamental in NLP and for images as well

The relative position of distinctive patterns (eyes, lips) are useful for understanding image content



Img credits Mark Carman



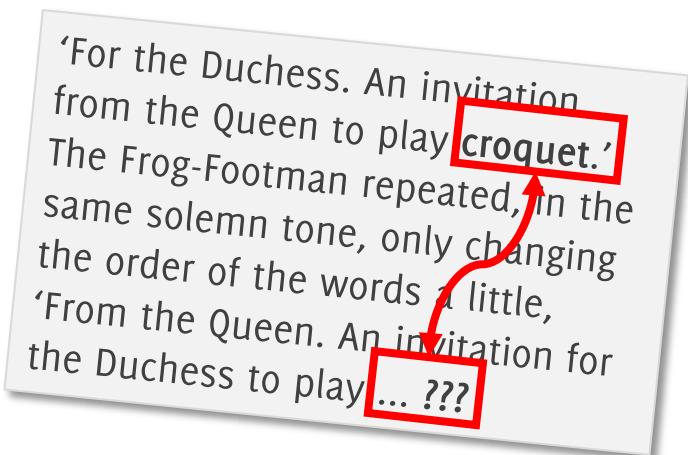
Imagined by a GAN ([generative adversarial network](#)) [StyleGAN2](#) (Dec 2019) - [Karras](#) et al. and Nvidia

Giacomo Boracchi

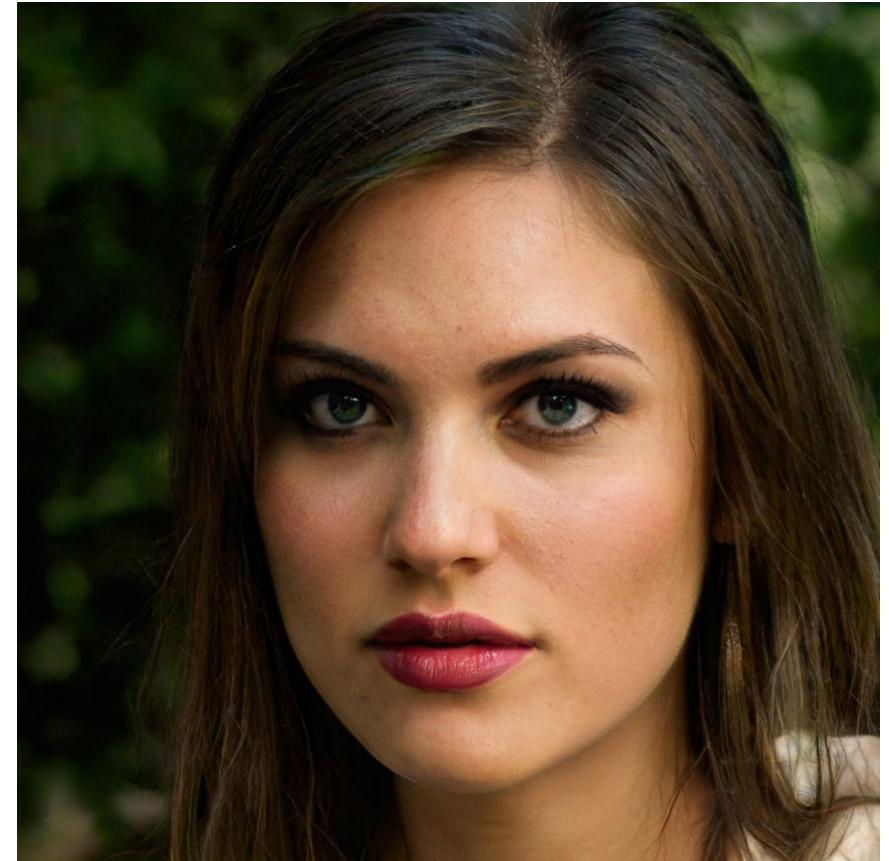
Long-Distance Dependencies

These are fundamental in NLP and for images as well

The latent representation gathers features extracted from portions of the image. These regions need to be sufficiently large to gather entire patterns (e.g. an eye).



Img credits Mark Carman



Imagined by a GAN ([generative adversarial network](#)) [StyleGAN2](#) (Dec 2019) - [Karras](#) et al. and Nvidia

The Receptive Field

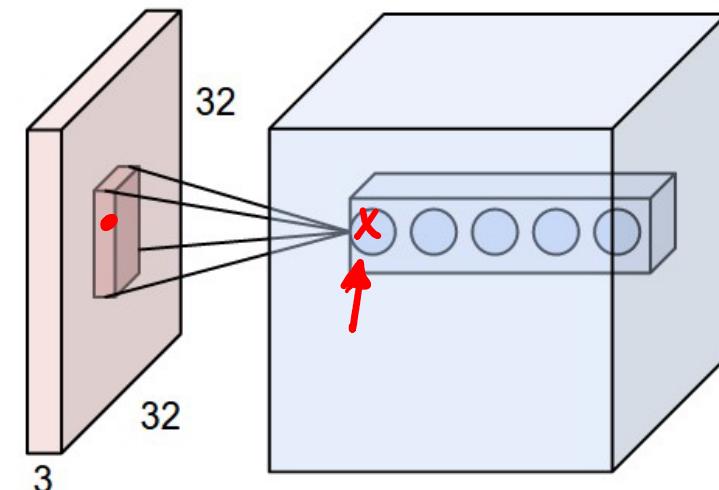
One of the basic concepts in deep CNNs.

Due to sparse connectivity, unlike in FC networks where the value of each output depends on the entire input, **in CNN each output only depends on a specific region in the input**.

This region in the input is the receptive field for that output

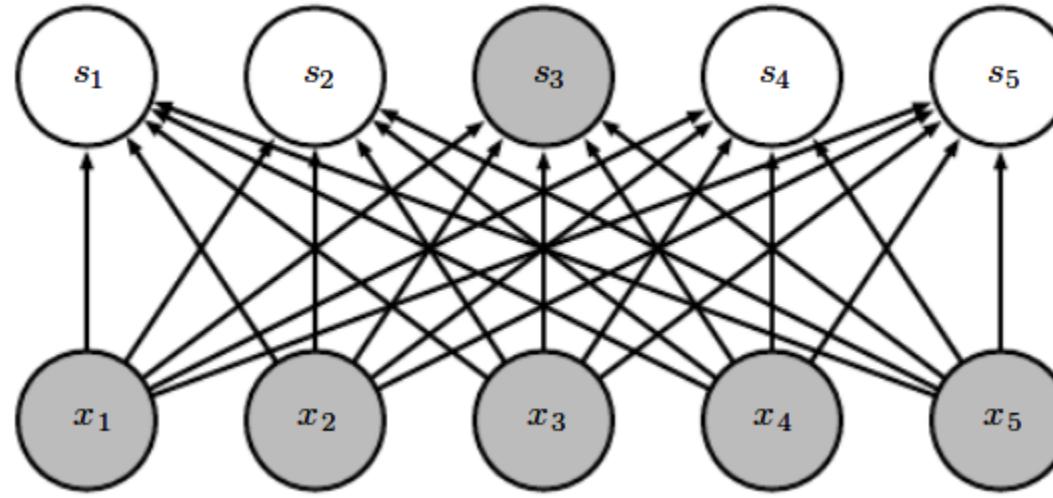
The deeper you go, the wider the receptive field is: maxpooling, convolutions and stride > 1 increase the receptive field

Usually, the receptive field refers to the final output unit of the network in relation to the network input, but the same definition holds for intermediate volumes

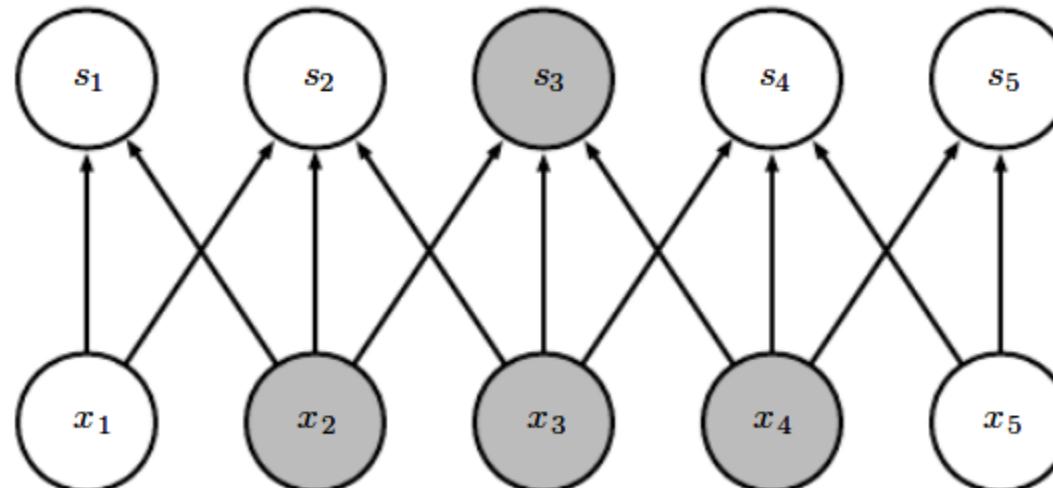


Receptive fields

MPL, Fully connected



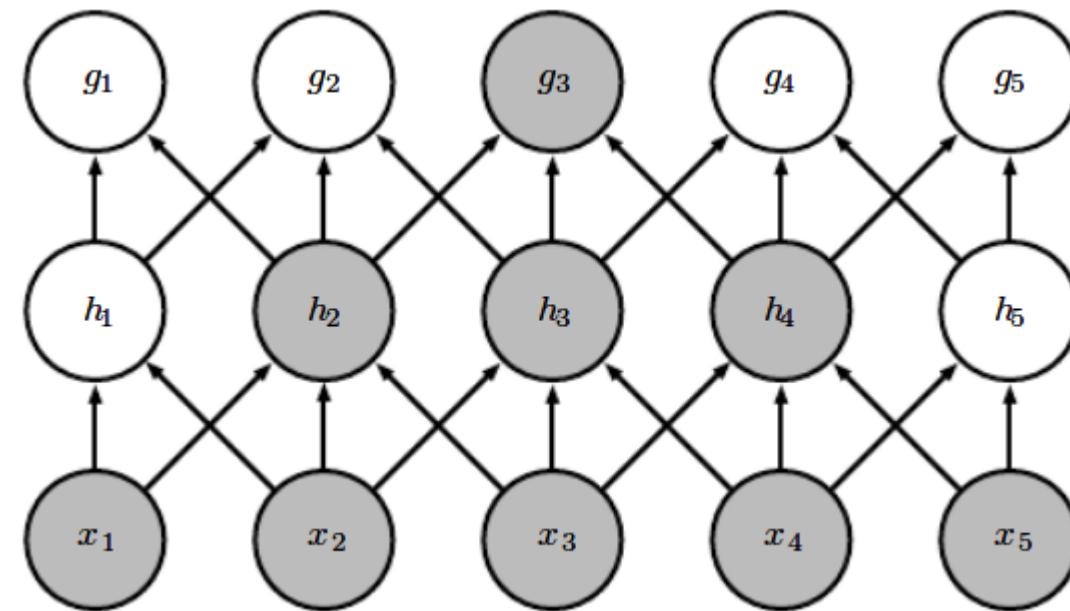
3×1 convolutional



Receptive fields

Deeper neurons depend on wider patches of the input (convolution is enough to increase receptive field, no need of maxpooling)

3x1 convolutional
3x1 convolutional



Exercise

Input:



Conv 3x3

Conv 3x3

Conv 3x3

Conv 3x3

Conv 3x3

Conv 3x3

map



Receptive fields

Input



Conv 3x3

Conv 3x3

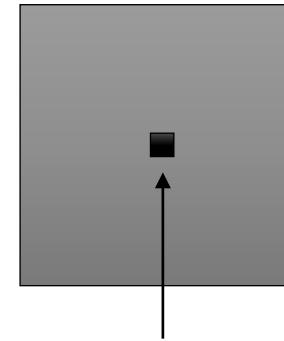
Conv 3x3

Conv 3x3

Conv 3x3

Conv 3x3

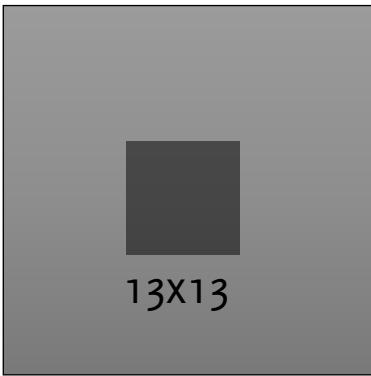
map



How large is the receptive field of the black neuron?

Receptive fields

Input



Conv 3x3

Conv 3x3

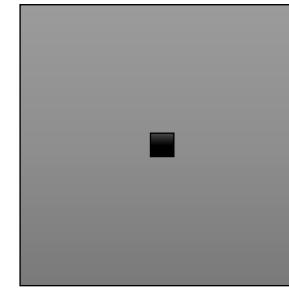
Conv 3x3

Conv 3x3

Conv 3x3

Conv 3x3

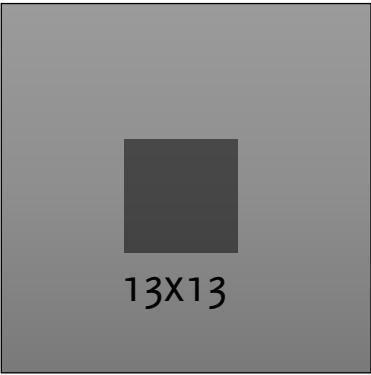
map



How large is the receptive field of the black neuron?

Receptive fields

Input



Conv 3x3

Conv 3x3

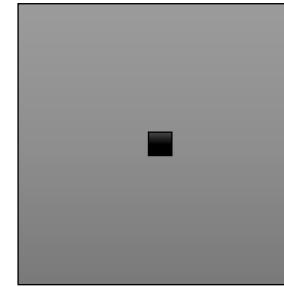
Conv 3x3

Conv 3x3

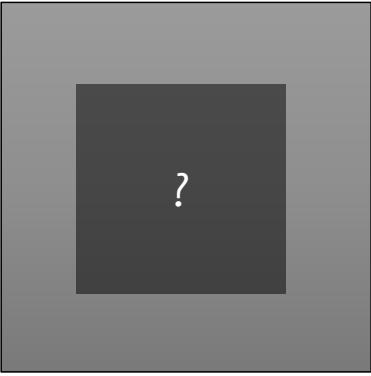
Conv 3x3

Conv 3x3

map



How large is the receptive field of the black neuron?



Conv 3x3

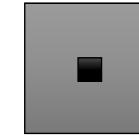
MP 2x2

Conv 3x3

MP 2x2

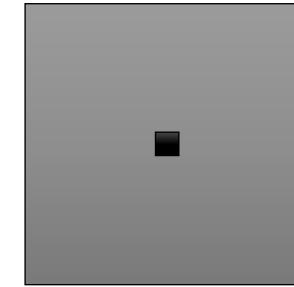
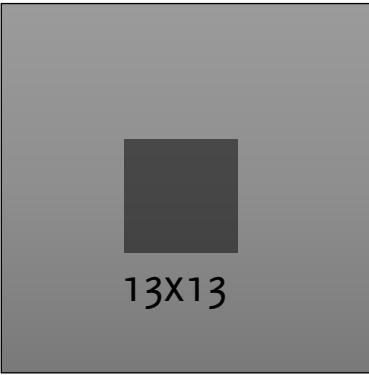
Conv 3x3

MP 2x2



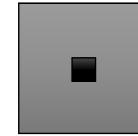
Receptive fields

Input



map

How large is the receptive field of the black neuron?



More About Receptive Fields

Understanding the Effective Receptive Field in Deep Convolutional Neural Networks

Wenjie Luo*

Yujia Li*

Raquel Urtasun

Richard Zemel

Department of Computer Science
University of Toronto

{wenjie, yujiali, urtasun, zemel}@cs.toronto.edu

The Effective Receptive Field

Not all pixels in a receptive field contribute equally to an output unit's response. Intuitively it is easy to see that pixels at the center of a receptive field have a much larger impact on an output.

In the forward pass, central pixels can propagate information to the output through many different paths, while the pixels in the outer area of the receptive field have very few paths to propagate its impact.

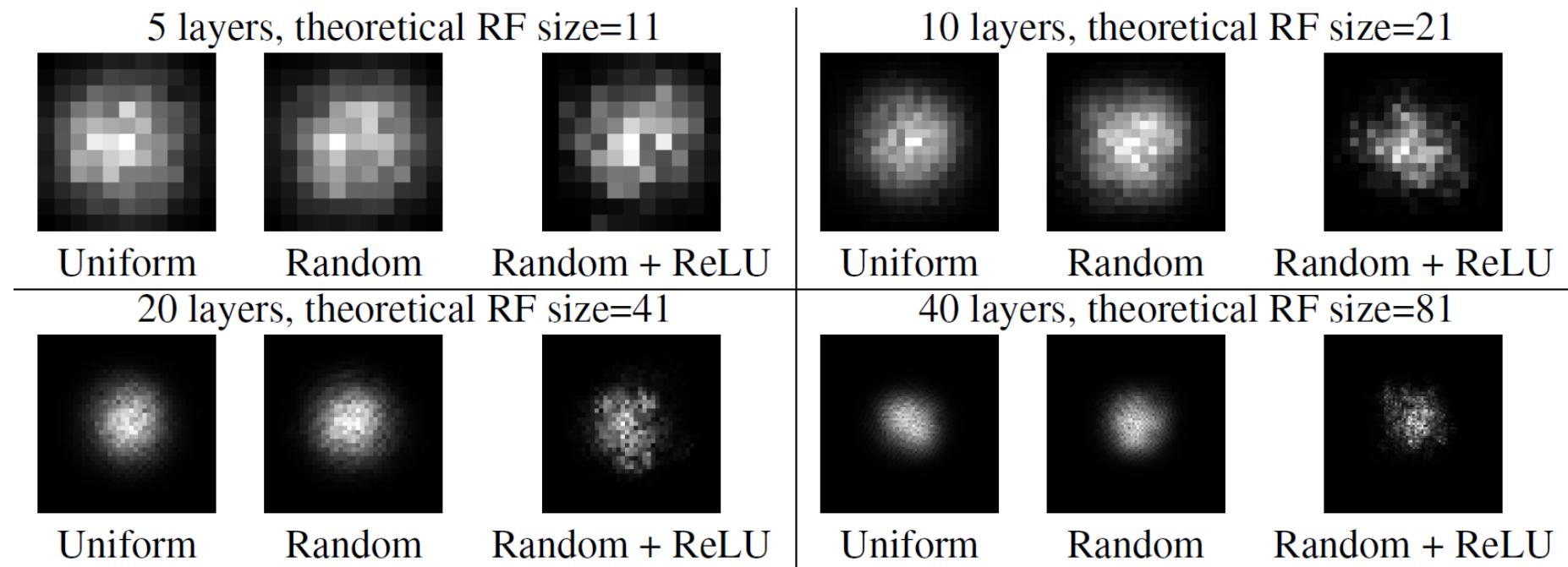
In the backward pass, gradients from an output unit are propagated across all the paths, and therefore the central pixels have a much larger magnitude for the gradient from that output.

In many cases, the impact in a receptive field distributes as a Gaussian: the effective receptive field, only occupies a fraction of the theoretical receptive field.

The Effective Receptive Field

Place a gradient signal of 1 at the center of the output plane and 0 everywhere else, and then **back-propagate this gradient** through the network to get input gradients.

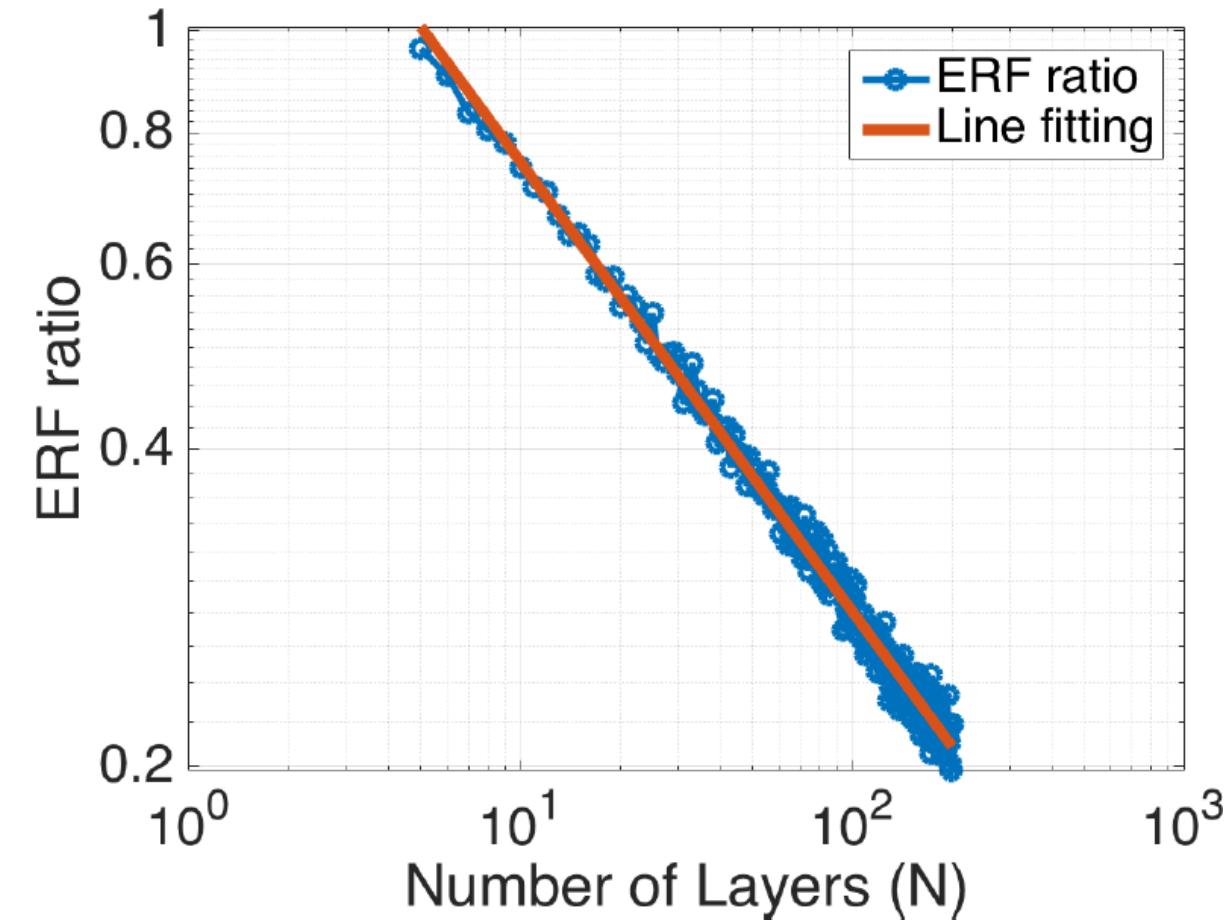
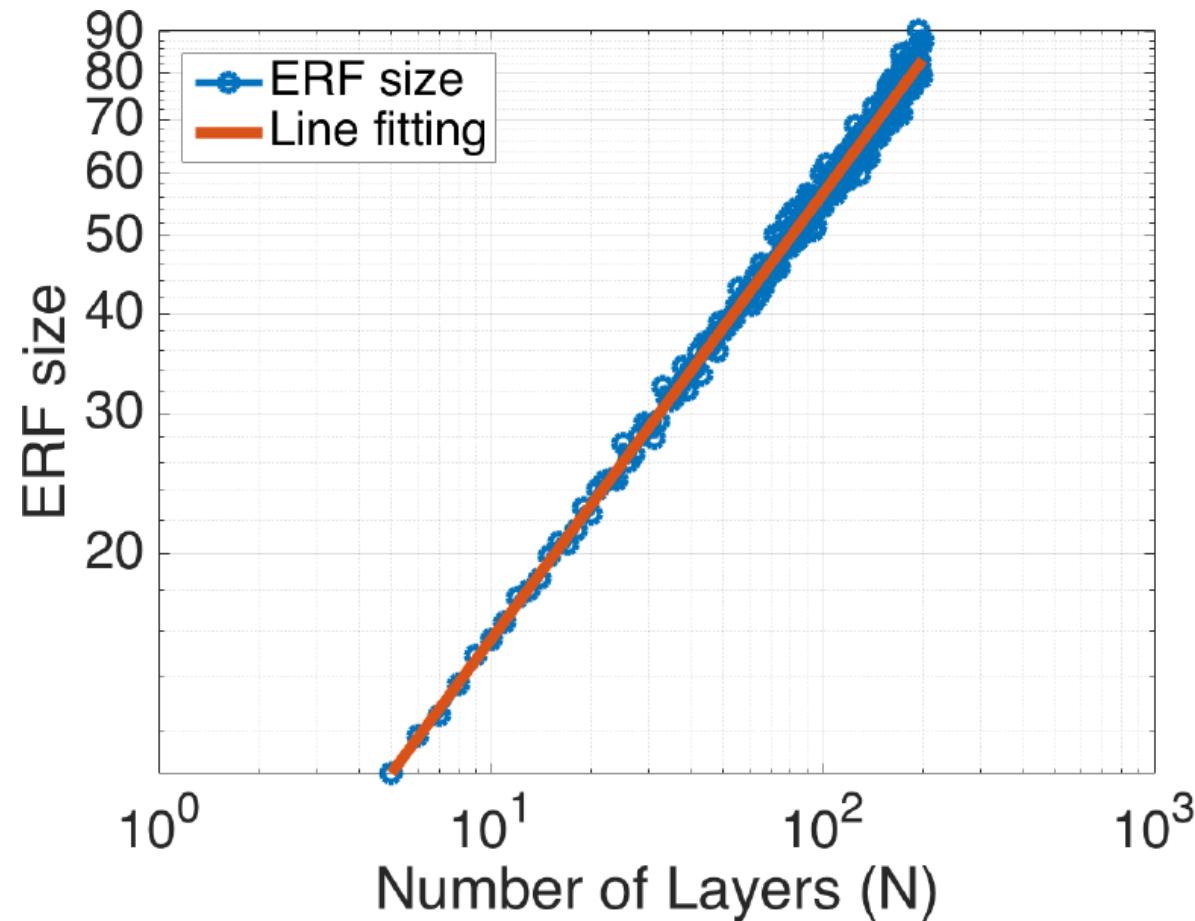
Average the resulting gradients at the input over **multiple initialization** of the network



The deeper the network, the smaller the ERF w.r.t. the theoretical one!

The Effective Receptive Field

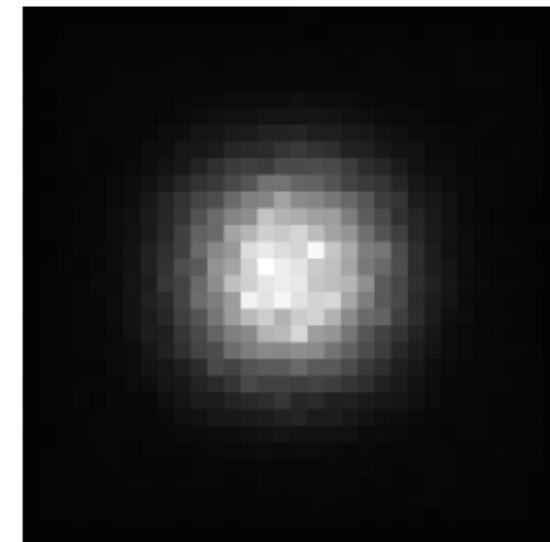
How the network depth influences the ERF and the theoretical one



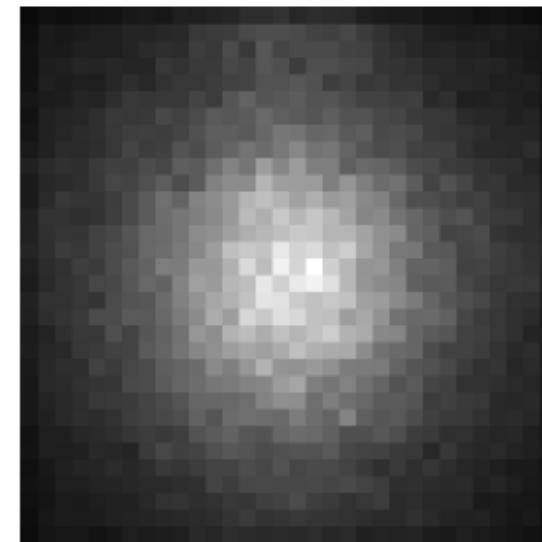
The Effective Receptive Field

Network training increases the ERF. Is random initialization bad?

CIFAR 10

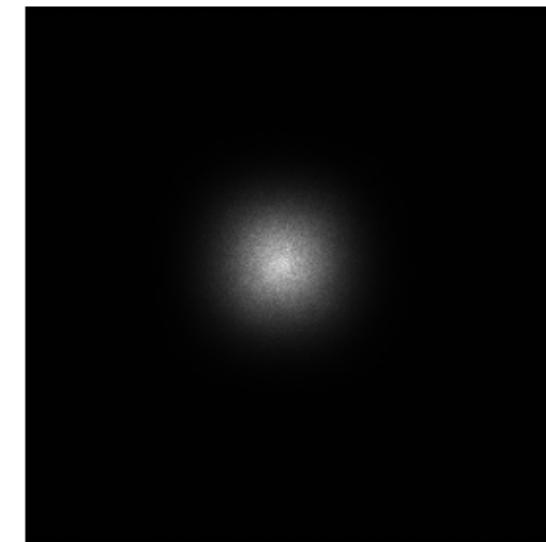


Before Training

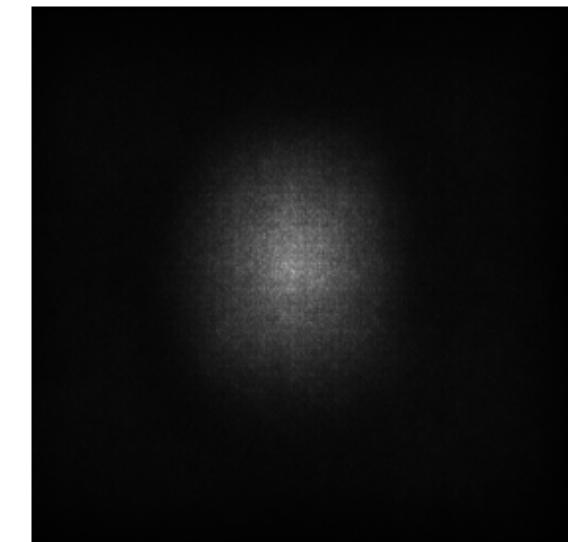


After Training

CamVid



Before Training



After Training

Part2: Attention in Images and Vision Transformers

Nonlocality and Self-similarity in Images

Let's go back to hand-crafted algorithms... there is still much to learn!

Self-Similarity in Natural Images

Patches in natural images are self-similar.

For each reference patch (in red) it is possible to find many other patches exhibiting a similar content.

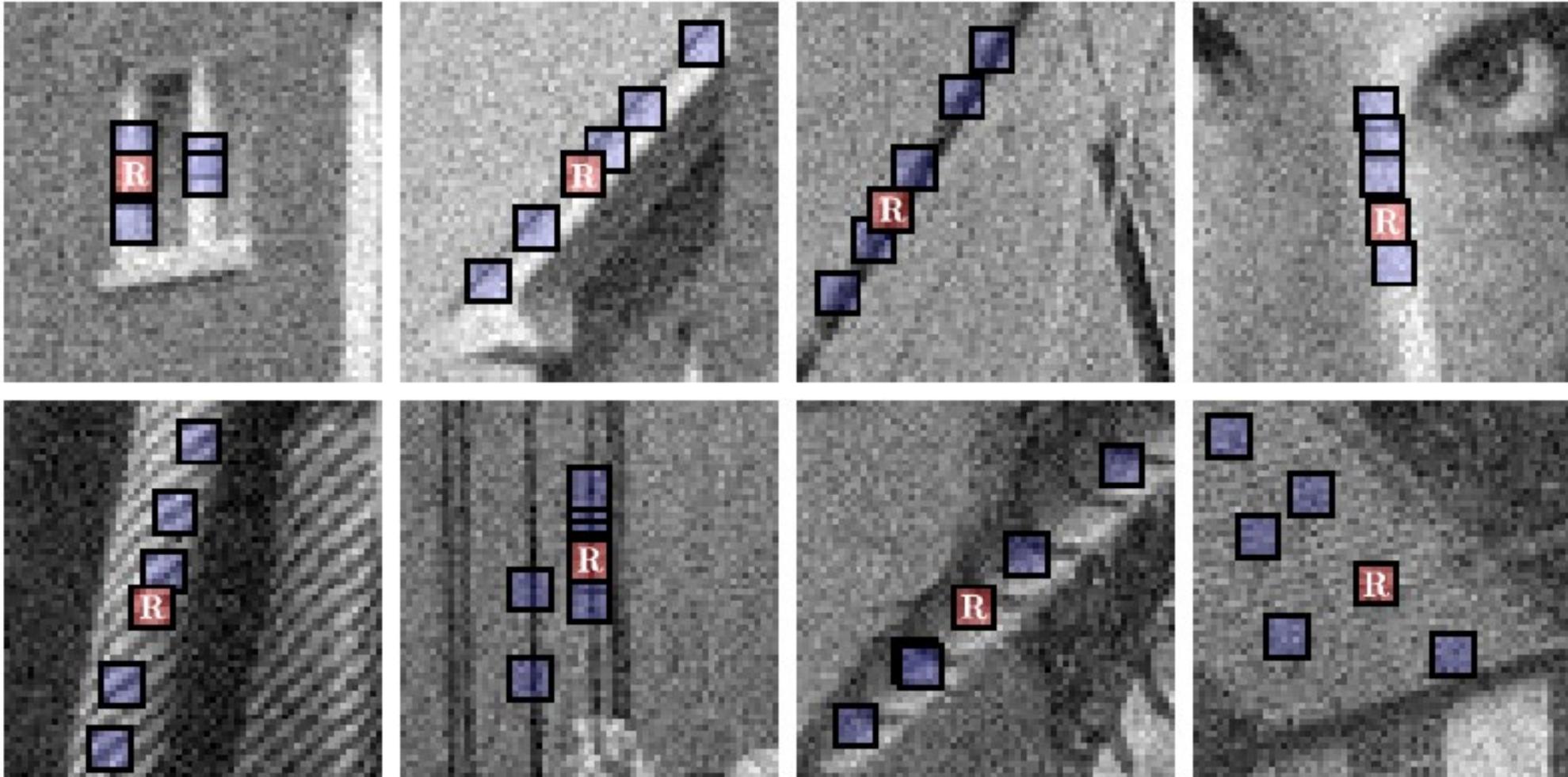
This has been used as a powerful prior to design **image restoration algorithms over the past few years**.

We will see a **key algorithm** to exploit long-distance dependencies in images



Image Credit: <https://webpages.tuni.fi/foi/GCF-BM3D/>

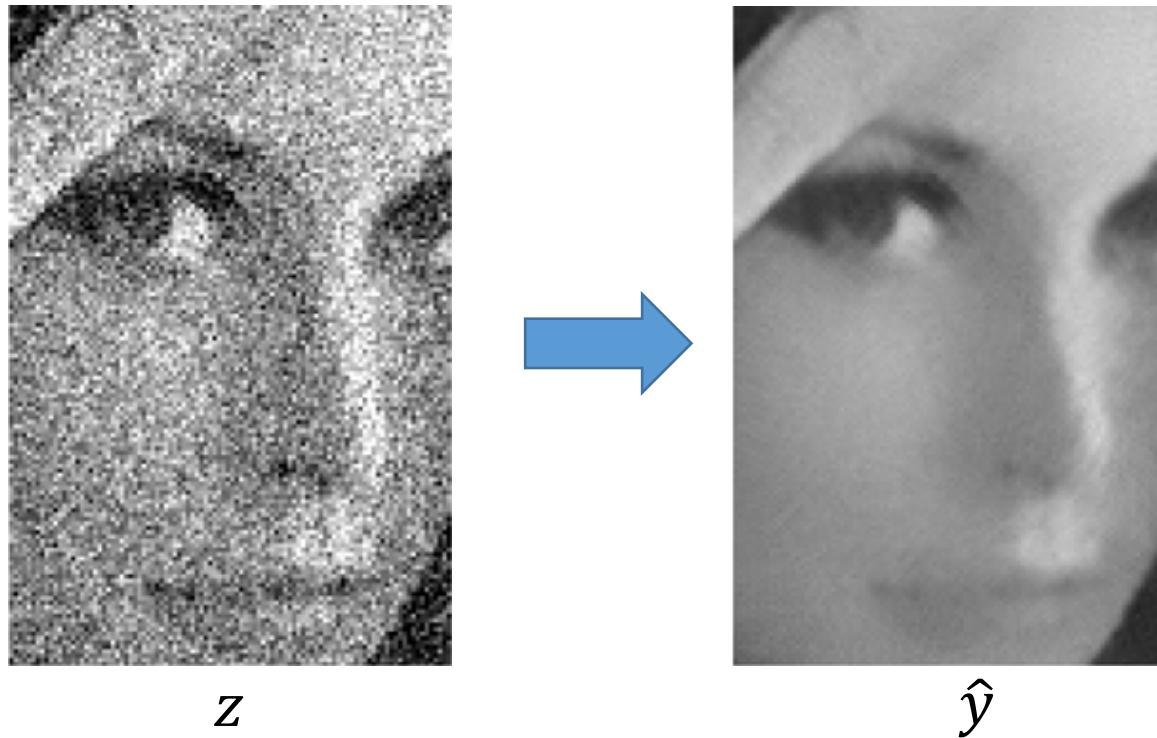
Self-Similarity in Natural Images



Self-similarity is a Powerful Denoising Prior

The goal of **image denoising** is to compute \hat{y} *realistic* estimate of the original image y , given the noisy observation z

Denoising is an **ill posed problem** and requires some form of **regularization** to promote outputs that are close to natural images.



Denoising: Image Formation Model

Observation model is

$$z(x) = y(x) + \eta(x), \quad x \in \mathcal{X}$$



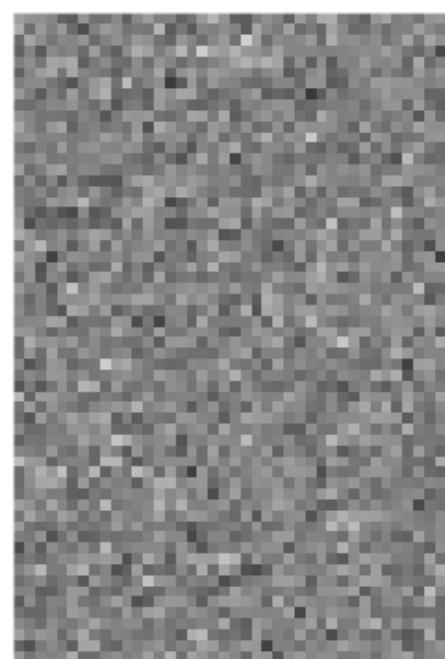
z

=



y

+



η

Denoising: Image Formation Model

Observation model is

$$z(x) = y(x) + \eta(x), \quad x \in \mathcal{X}$$

where:

- x denotes the pixel coordinates in the domain $\mathcal{X} \subset \mathbb{Z}^2$
- y is the original (noise-free and unknown) image, $y \in [0,1]$
- z is the noisy observation, $z \in [0,1]$ (clipping)
- η is the noise realization

For the sake of simplicity, we assume Additive White Gaussian Noise (AWGN):

$\eta \sim N(0, \sigma^2)$ and $\eta(x)$ are all independent realizations.

The noise standard deviation σ is also assumed as known.

Image Patches

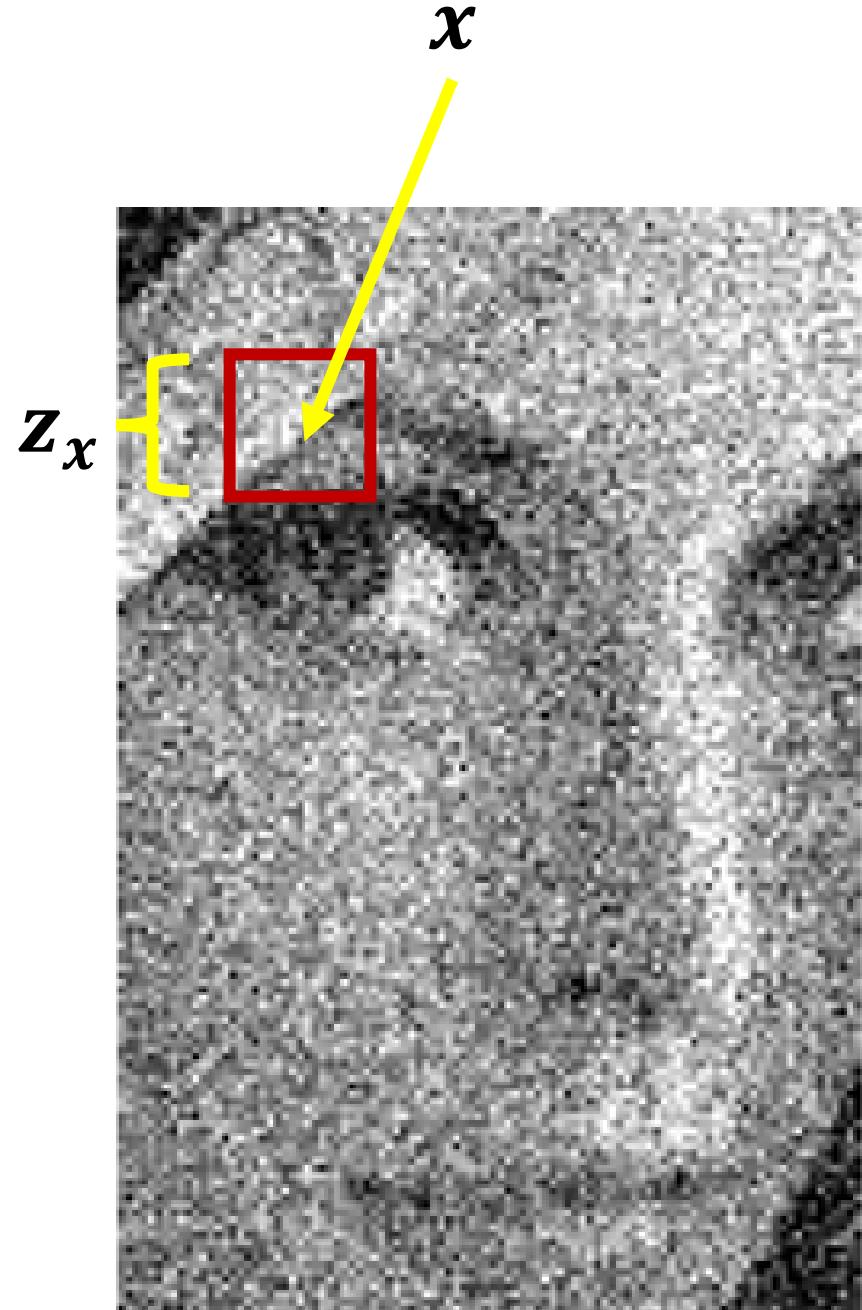
Let $U \subset \mathbb{Z}^2$ be a spatial neighborhood centered at the origin $(0,0) \in \mathbb{Z}^2$,

- we define a patch centered at a pixel $x \in X$ in the observation z

$$\mathbf{z}_x = \{z(x + u), \quad u \in U\}$$

- a patch centered at a pixel $x \in X$ in the original image y

$$\mathbf{y}_x = \{y(x + u), \quad u \in U\}$$



Self-similarity for Denoising

Prior: similar patches will correspond to similar central values.

Namely, the patch-wise distance **correlates well with the distance of the two central pixels** in the noisy-free patches

The idea is to assess the similarity between pixels $y(x_1)$ and $y(x_2)$ (not available), via the similarity of the corresponding noisy patches \mathbf{z}_{x_1} and \mathbf{z}_{x_2} .

Thus, weights based on photometric differences should be computed from $\|\mathbf{z}_{x_1} - \mathbf{z}_{x_2}\|_2$ rather than $z(x_1) - z(x_2)$.



Windowed Patch Distance in NL-means

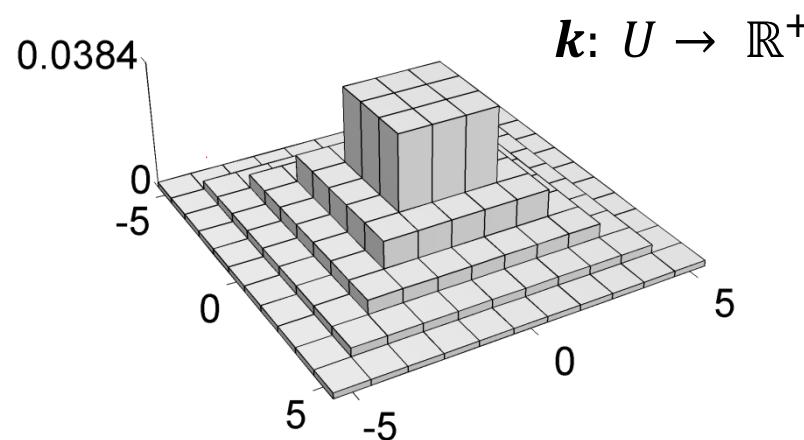
The distance operator is defined as a function $f(z_{x_1}, z_{x_2})$ of the *windowed quadratic distance* between patches

$$f(z_{x_1}, z_{x_2}) = \sum_{u \in U} (z(x_1 + u) - z(x_2 + u))^2 k(u)$$

Where $k: U \rightarrow \mathbb{R}^+$ is a windowing kernel (we use k since w is being used for the aggregation weights)

Windowed Patch Distance in NL-means

The windowing kernel $\mathbf{k}: U \rightarrow \mathbb{R}^+$ adjusts the contribution of each difference term depending on the position of u with respect to the patch center.



$$\|z_{x_1} - z_{x_2}\|_2^2$$
$$\|(z_{x_1} - z_{x_2})\sqrt{\mathbf{k}}\|_2^2$$

d performs a **pixel-wise** comparison of the patches

the decay of \mathbf{k} reflects how much similarity between $y(x_1)$ and $y(x_2)$ may be implied from the similarity between $y(x_1 + u)$ and $y(x_2 + u)$ when $u \neq 0$.



Non Local Means Filter (NL-means)

The denoised image \hat{y} is a weighted average of all image pixels

$$\hat{y}(x_1) = \sum_{x_2 \in X} w(\mathbf{z}_{x_1}, \mathbf{z}_{x_2}) z(x_2), \quad \forall x_1 \in X$$

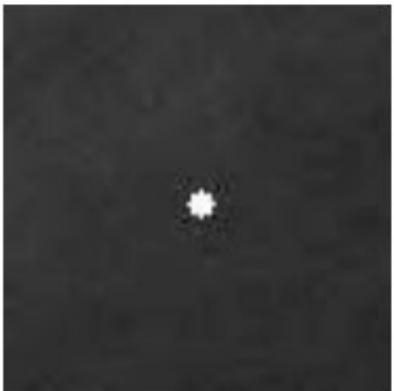
where weights $\{w(x_1, x_2)\}$ are adaptively defined depending on the similarity between two noisy patches \mathbf{z}_{x_1} and \mathbf{z}_{x_2}

$$w(\mathbf{z}_{x_1}, \mathbf{z}_{x_2}) = \frac{e^{\left(-\frac{f(\mathbf{z}_{x_1}, \mathbf{z}_{x_2})}{h^2}\right)}}{\sum_X e^{\left(-\frac{f(\mathbf{z}_{x_1}, \mathbf{z}_{x_2})}{h^2}\right)}}$$

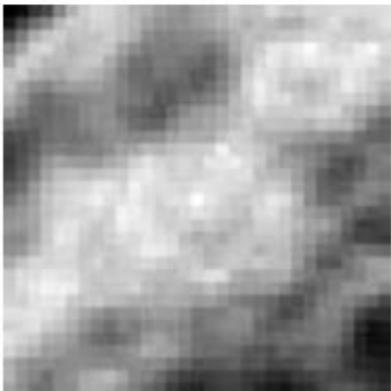
- $f(\mathbf{z}_{x_1}, \mathbf{z}_{x_2})$: distance measure between patches in x_1 and x_2 ,
- $h > 0$ is a smoothing parameter ($h = \sigma$).
- \mathbf{z}_{x_1} similar to $\mathbf{z}_{x_2} \Rightarrow f(\mathbf{z}_{x_1}, \mathbf{z}_{x_2})$ is small $\Rightarrow w(x_1, x_2)$ large
- NL-means operates pixel-wise

Weights in Nonlocal Means

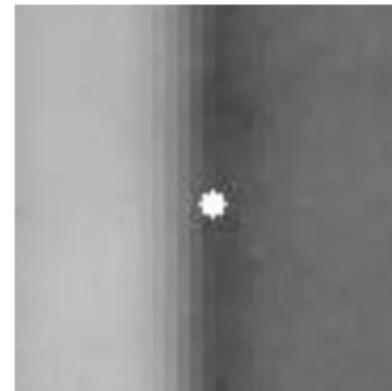
Selected pixel



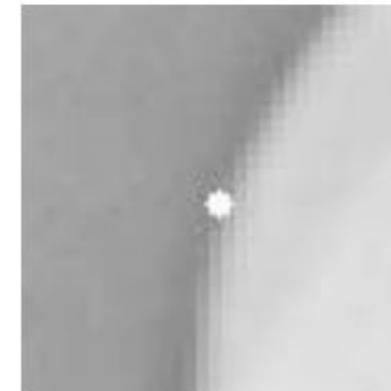
Weights from neigh.



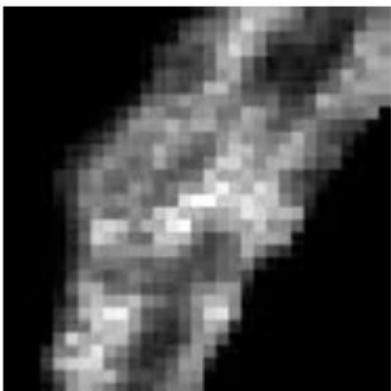
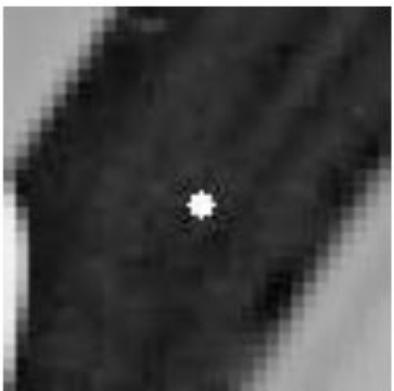
(a)



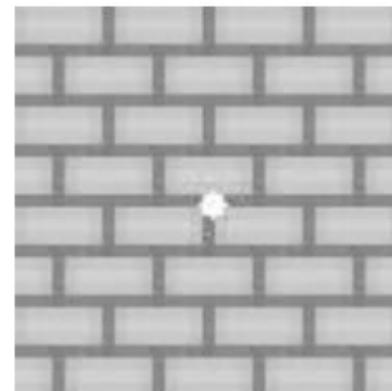
(b)



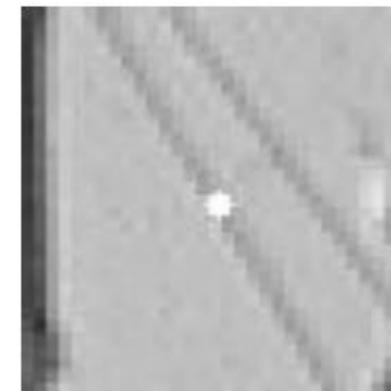
(c)



(d)



(e)



(f)

Weights from Patch-distances



Attention Mechanisms and Nonlocal Blocks

Non-local Neural Networks



This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the version available on IEEE Xplore.

Non-local Neural Networks

Xiaolong Wang^{1,2*} Ross Girshick² Abhinav Gupta¹ Kaiming He²

¹Carnegie Mellon University

²Facebook AI Research

Non-local Neural Networks

In images, long-distance dependences are learned by stacking many convolutional layers.

Increasing the receptive field is not simple:

- Computationally inefficient
- Networks becomes difficult to train

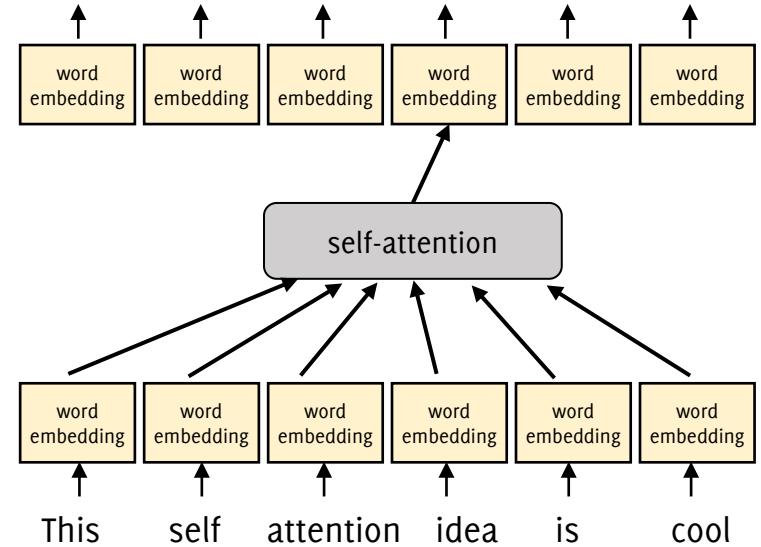
New non-local operation in CNN that is inspired to NonLocal Means Denoising.

Compute a weighted sum of all the features in the input activation map

Self Attention

In NLP, self attention is a mechanism for:

- Combining all the input word embeddings to produce new word embedding
- Each high-level embedding is **weighted average** of word embeddings below it



Non-local Blocks bring this mechanism to neural networks meant for images and videos.

Rather than word embedding, self-attention will involve features of the same activation map

Non local operator

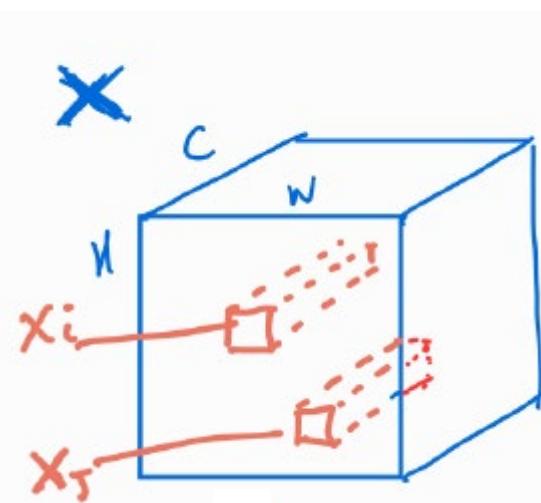
Generic non-local operator

$$y_i = \frac{1}{\mathcal{C}(x_i)} \sum_j f(x_i, x_j) g(x_j)$$

where (notation is NOT consistent with NLMs)

- x_i and x_j are feature vectors at different spatial location of an activation map.
 x_i is the reference, x_j spans the entire activation
- $f(x_i, x_j)$ is a scalar distance between the two activations
- $g(x_j)$ is an embedding of the feature map x_j
- $\mathcal{C}(x_i)$ is a normalization factor

$$\mathcal{C}(x_i) = \sum_j f(x_i, x_j)$$



The Embeddings $\frac{1}{C(x_i)} \sum_j f(x_i, x_j) g(x_j)$

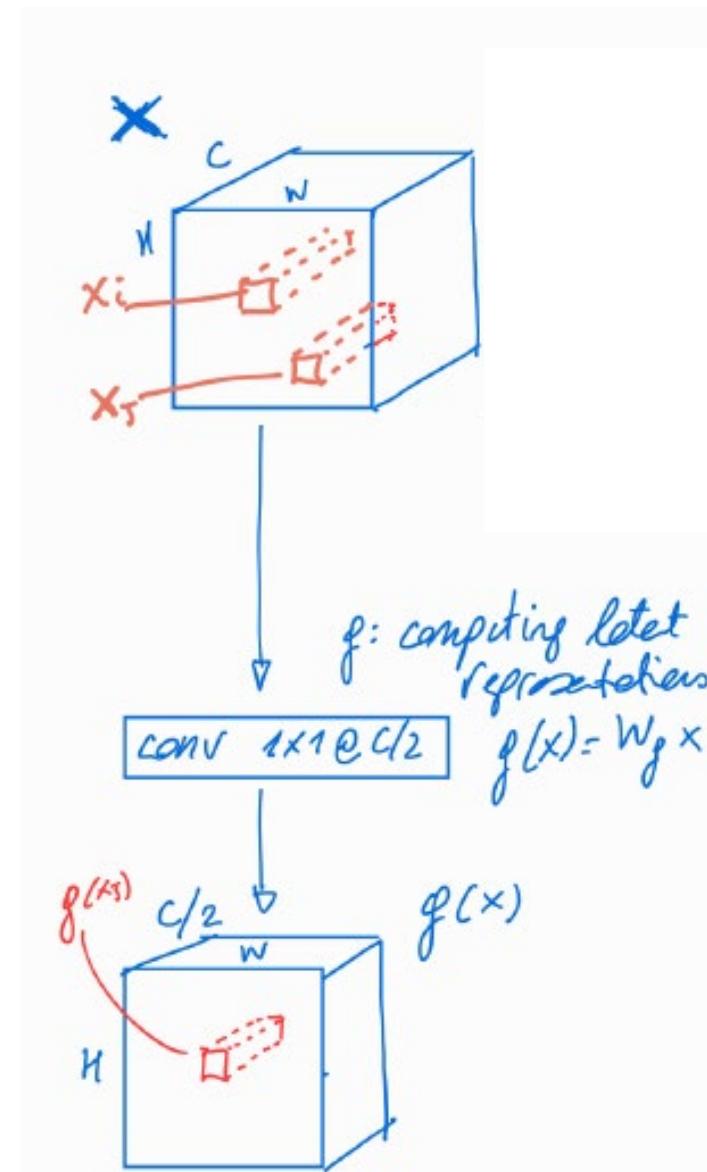
All the embeddings are typically computed by linear transformations

$$\theta(x_i) = W_\theta x_i$$

$$\phi(x_j) = W_\phi x_j$$

$$g(x_j) = W_g x_j$$

For the sake of efficiency, these are 1×1 conv layers with $C/2$ filters, to halve the number of channels ($W_\theta \in \mathbb{R}^{C/2 \times C}$)



We adopt different indexes i, j depending on the role played in the block, but they both span the entire activation x

The Embeddings $\frac{1}{C(x_i)} \sum_j f(x_i, x_j) g(x_j)$

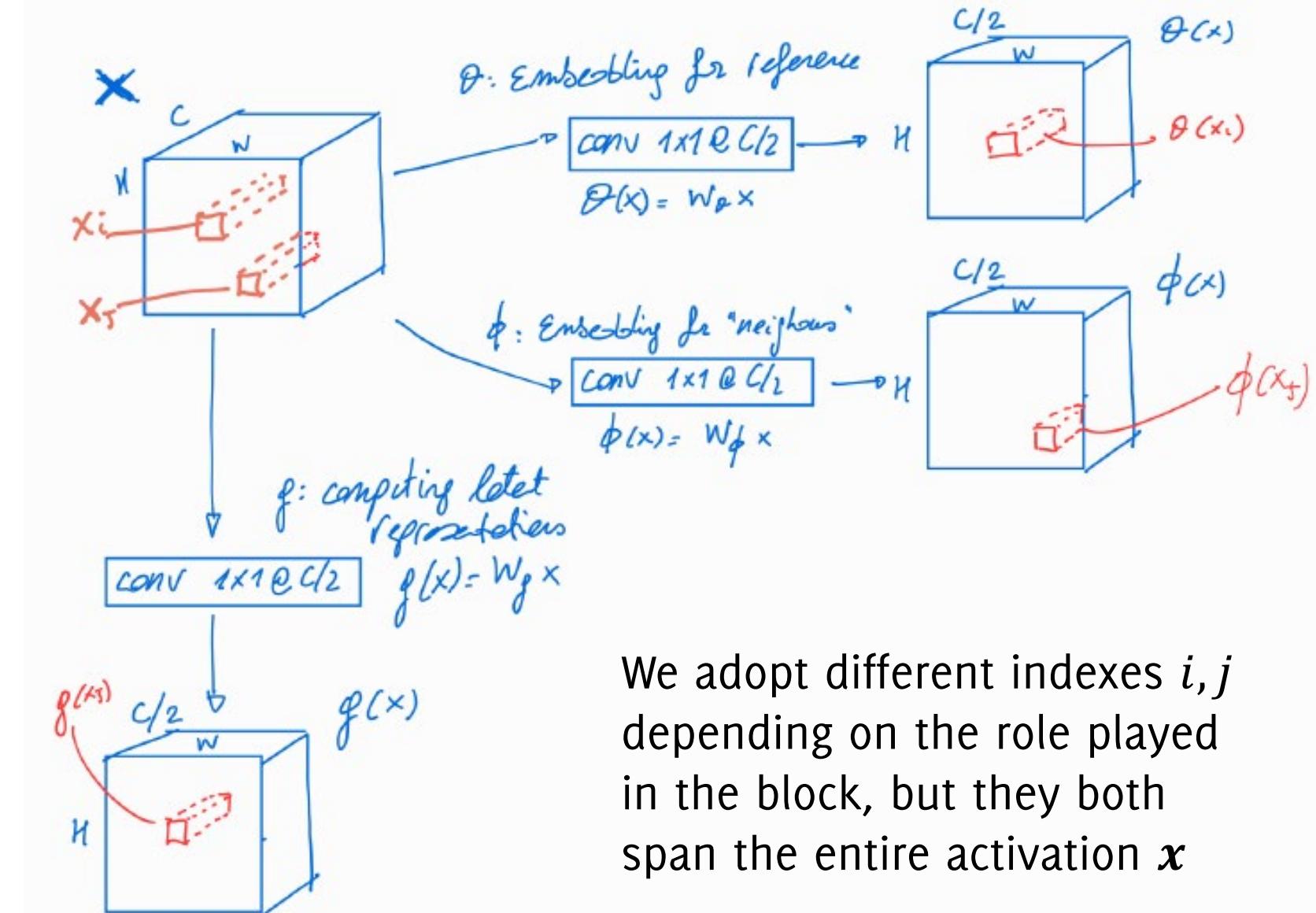
All the embeddings are typically computed by linear transformations

$$\theta(x_i) = W_\theta x_i$$

$$\phi(x_j) = W_\phi x_j$$

$$g(x_j) = W_g x_j$$

For the sake of efficiency, these are 1×1 conv layers with $C/2$ filters, to halve the number of channels ($W_\theta \in \mathbb{R}^{C/2 \times C}$)



We adopt different indexes i, j depending on the role played in the block, but they both span the entire activation x

The distance $\frac{1}{\mathcal{C}(\mathbf{x}_i)} \sum_j f(\mathbf{x}_i, \mathbf{x}_j) g(\mathbf{x}_j)$

$$\mathbf{y}_i = \frac{1}{\mathcal{C}(\mathbf{x}_i)} \sum_j f(\mathbf{x}_i, \mathbf{x}_j) g(\mathbf{x}_j)$$

This has to be a scalar function, here are a few examples

Gaussian:

$$f(\mathbf{x}_i, \mathbf{x}_j) = e^{\mathbf{x}_i^T \mathbf{x}_j}$$

Embedded Gaussian:

$$f(\mathbf{x}_i, \mathbf{x}_j) = e^{\mathbf{x}_i^T W_\theta^T W_\phi \mathbf{x}_j}$$

Dot Product:

$$f(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$$

Concatenation:

$$f(\mathbf{x}_i, \mathbf{x}_j) = \text{ReLU}(\mathbf{w}_f^T \cdot [\mathbf{x}_i W_\theta, W_\phi \mathbf{x}_j])$$

The distance $\frac{1}{\mathcal{C}(\mathbf{x}_i)} \sum_j f(\mathbf{x}_i, \mathbf{x}_j) g(\mathbf{x}_j)$

$$\mathbf{y}_i = \frac{1}{\mathcal{C}(\mathbf{x}_i)} \sum_j f(\mathbf{x}_i, \mathbf{x}_j) g(\mathbf{x}_j)$$

This has to be a scalar function, here are a few examples

Gaussian:

$$f(\mathbf{x}_i, \mathbf{x}_j) = e^{\mathbf{x}_i^T \mathbf{x}_j}$$

Embedded Gaussian:

$$f(\mathbf{x}_i, \mathbf{x}_j) = e^{\mathbf{x}_i^T W_\theta^T W_\phi \mathbf{x}_j}$$

Dot Product

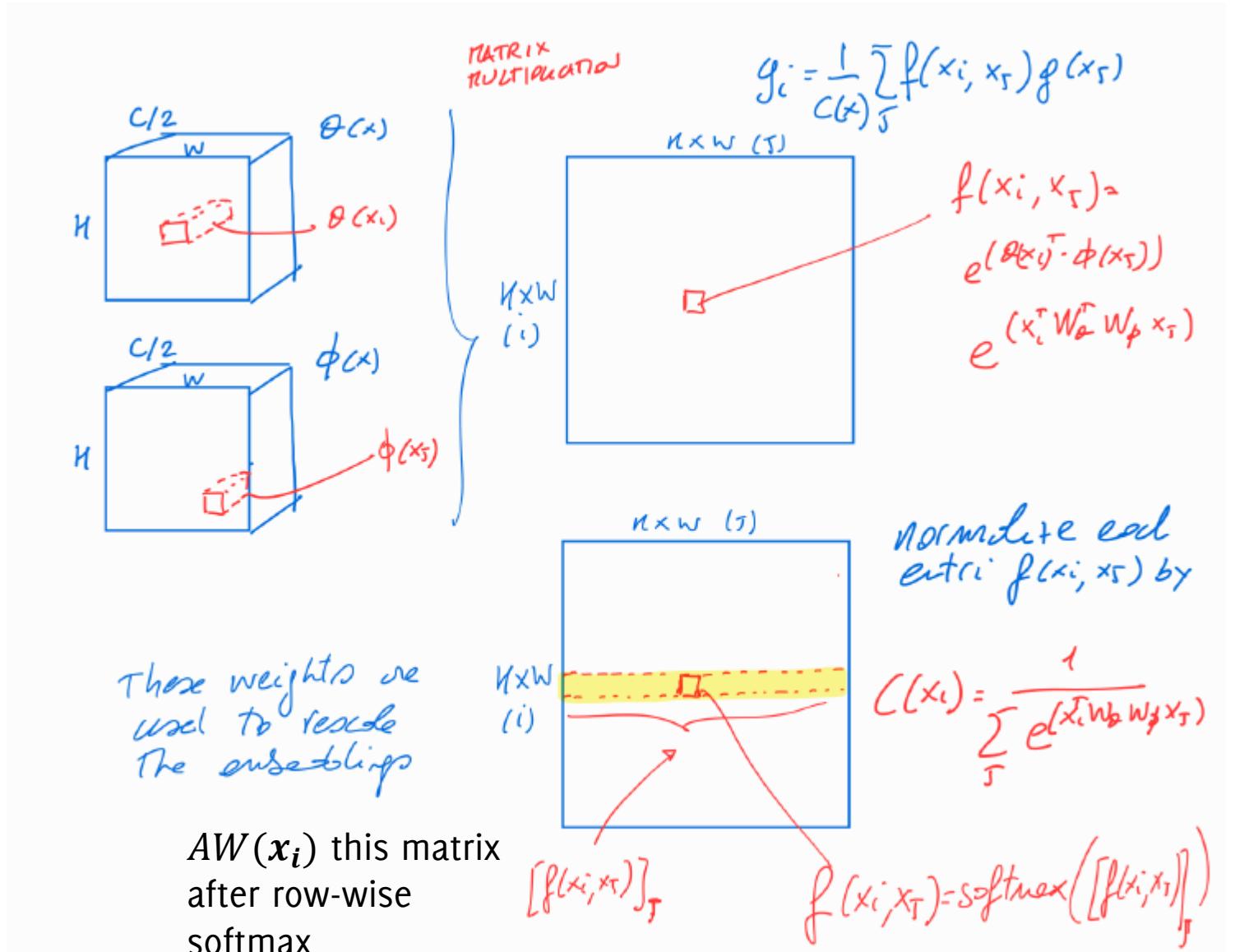
$$f(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$$

Concatenation

$$f(\mathbf{x}_i, \mathbf{x}_j) = \text{ReLU}(\mathbf{w}_f^T \cdot [\mathbf{x}_i W_\theta, W_\phi \mathbf{x}_j])$$

The attention weights: values of $f(x_i, x_j)$

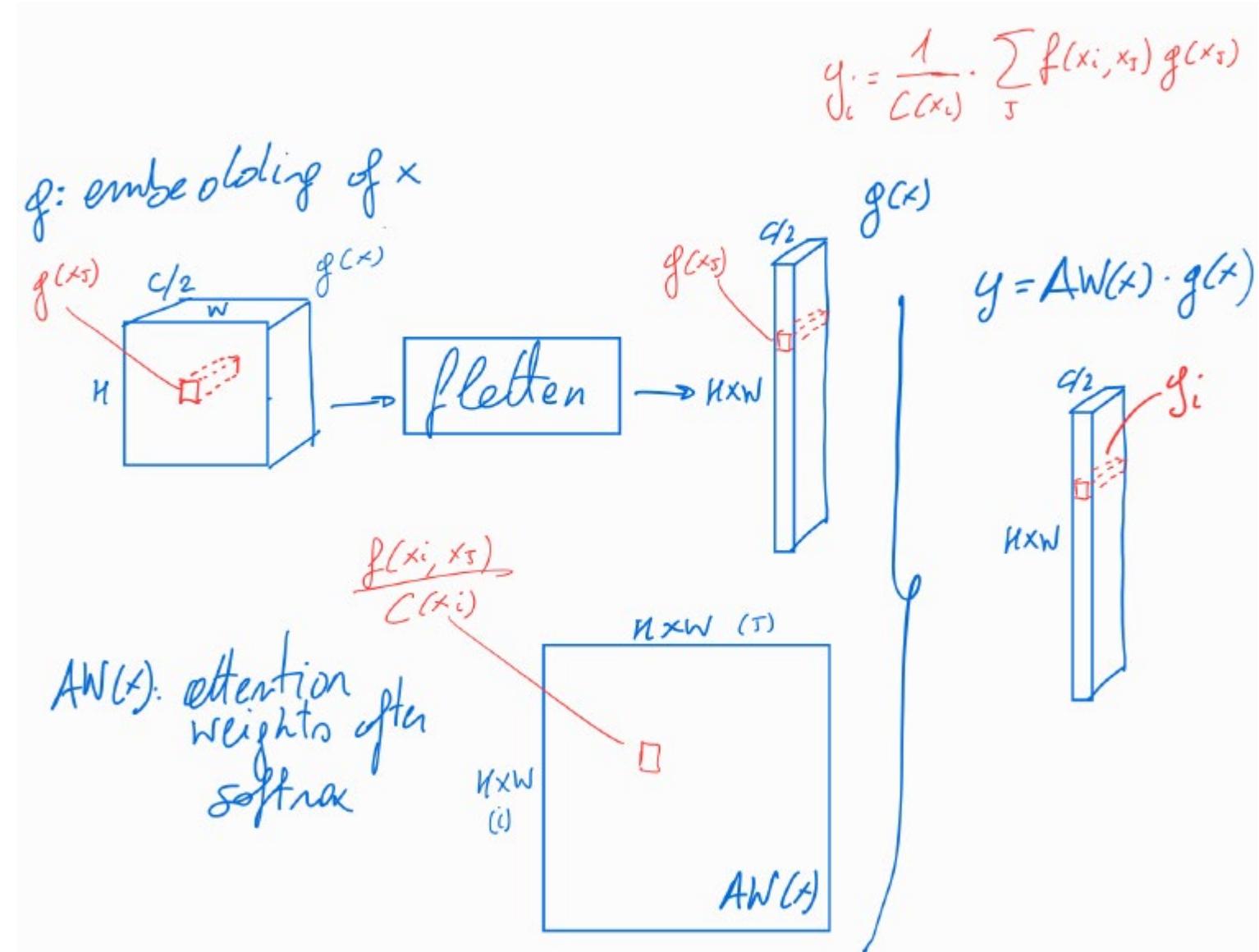
In case of embedded Gaussians, the weights given by $f(x_i, x_j)$ are computed by matrix multiplication and normalized by a softmax (over the rows of the weight matrix)



The self-attention on the activations

The self-attention is obtained by multiplying (tensor multiplication) the embedding $g(x_i)$ against the obtained weights.

By doing so we compute the output y_i in the entire activation map



The Non-local Block

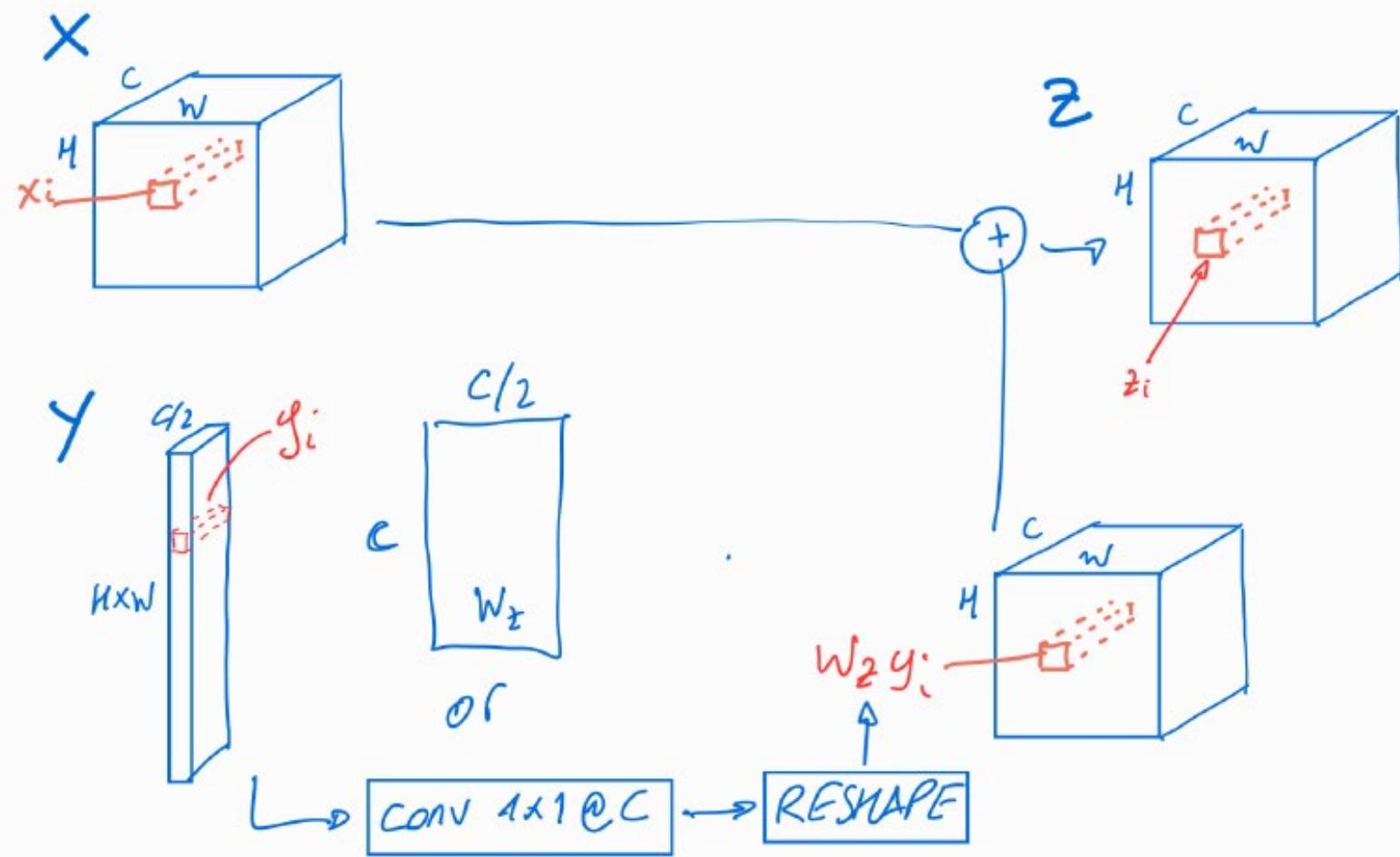
The output of the non-local block is not y itself, but this is used with skip connections

$$z_i = W_z y_i + x_i$$

W_z is also a convolutional layer with 1×1 convolutions.

This makes simpler to introduce the block in a pretrained model
(initializing W_z to zeros)

$$z_i = W_z y_i + x_i \quad (\text{RESIDUAL LEARNING})$$



Efficiency notes

The number of channels C in the embedding is halved to reduce computations.

Similarly, it is possible to perform spatial pooling on $\phi(x_j)$ and $g(x_j)$, to reduce the terms in the sum

$$\mathbf{y}_i = \frac{1}{\mathcal{C}(\mathbf{x}_i)} \sum_j f(\mathbf{x}_i, \mathbf{x}_j) g(\mathbf{x}_j)$$

This give rise to rectangular weight matrix $AW(\mathbf{x})$, but \mathbf{y}_i is still defined over the entire activation block

Video Classification Models

The Non-local block is seamlessly inserted in 3D CNN to perform video classification (as illustrated here)

«Nevertheless, our method surpasses all the existing RGB or RGB + flow based methods by a good margin. *Without using optical flow and without any bells and whistles*, our method is on par with the heavily engineered results of the 2017 competition winner»

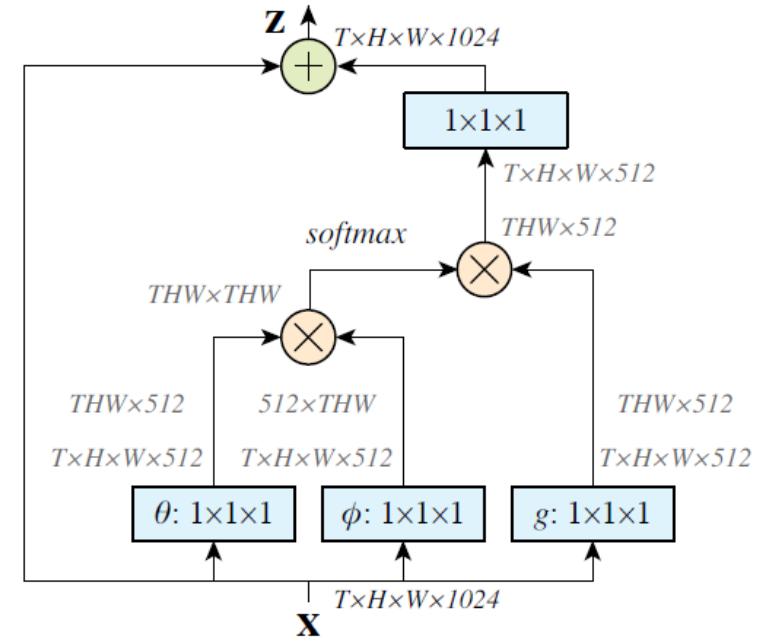


Figure 2. A spacetime **non-local block**. The feature maps are shown as the shape of their tensors, e.g., $T \times H \times W \times 1024$ for 1024 channels (proper reshaping is performed when noted). “ \otimes ” denotes matrix multiplication, and “ \oplus ” denotes element-wise sum. The softmax operation is performed on each row. The blue boxes denote $1 \times 1 \times 1$ convolutions. Here we show the embedded Gaussian version, with a bottleneck of 512 channels. The vanilla Gaussian version can be done by removing θ and ϕ , and the dot-product version can be done by replacing softmax with scaling by $1/N$.

Vision Transformers

Vision Transformers (ViT)

Published as a conference paper at ICLR 2021

AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

Alexey Dosovitskiy^{*,†}, Lucas Beyer^{*}, Alexander Kolesnikov^{*}, Dirk Weissenborn^{*},
Xiaohua Zhai^{*}, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer,
Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby^{*,†}

^{*}equal technical contribution, [†]equal advising

Google Research, Brain Team

{adosovitskiy, neilhoulsby}@google.com

Background

Self-attention-based architectures, in particular **Transformers**, have become the **model of choice in NLP**.

Dominant approach:

1. train huge models on a **large corpus**,
2. **fine tune** on a specific task

Transformers achieve SoA

In **Visual Recognition**, **CNNs remain dominant** with SoA achieved by **ResNet-like architecture**

Vision Transformer (ViT)

- The first *pure transformer* architecture applied on images without relying on convolutional blocks
- Non-overlapping patches are processed as tokens in text.
- The entire image is fed to a Transformer as a sequence of patch embeddings.

Vision Transformers (proudly) **lack some of the inductive biases (priors) inherent to CNNs**, such as translation equivariance and locality!

Trained on “mid-size” dataset (e.g. Alexnet 1K), does not pair ResNet performance.

Vision Transformer (ViT) and Large Scale Training

When trained or pre-trained on very large datasets (ImageNet 21K, JFT-300M)

- Outperformed CNN-based architectures.
- Large scale training enable learning 2D patterns of images even without inductive biases, namely a model that does not promote such a 2D structure.

The same paradigm as for text:

- training on a huge dataset,
- fine tuning for a specific task with fewer data

Large Datasets

“Mid-size” datasets:

- Imagenet 1k (1K classes, 1M images). The one used for “alexnet”

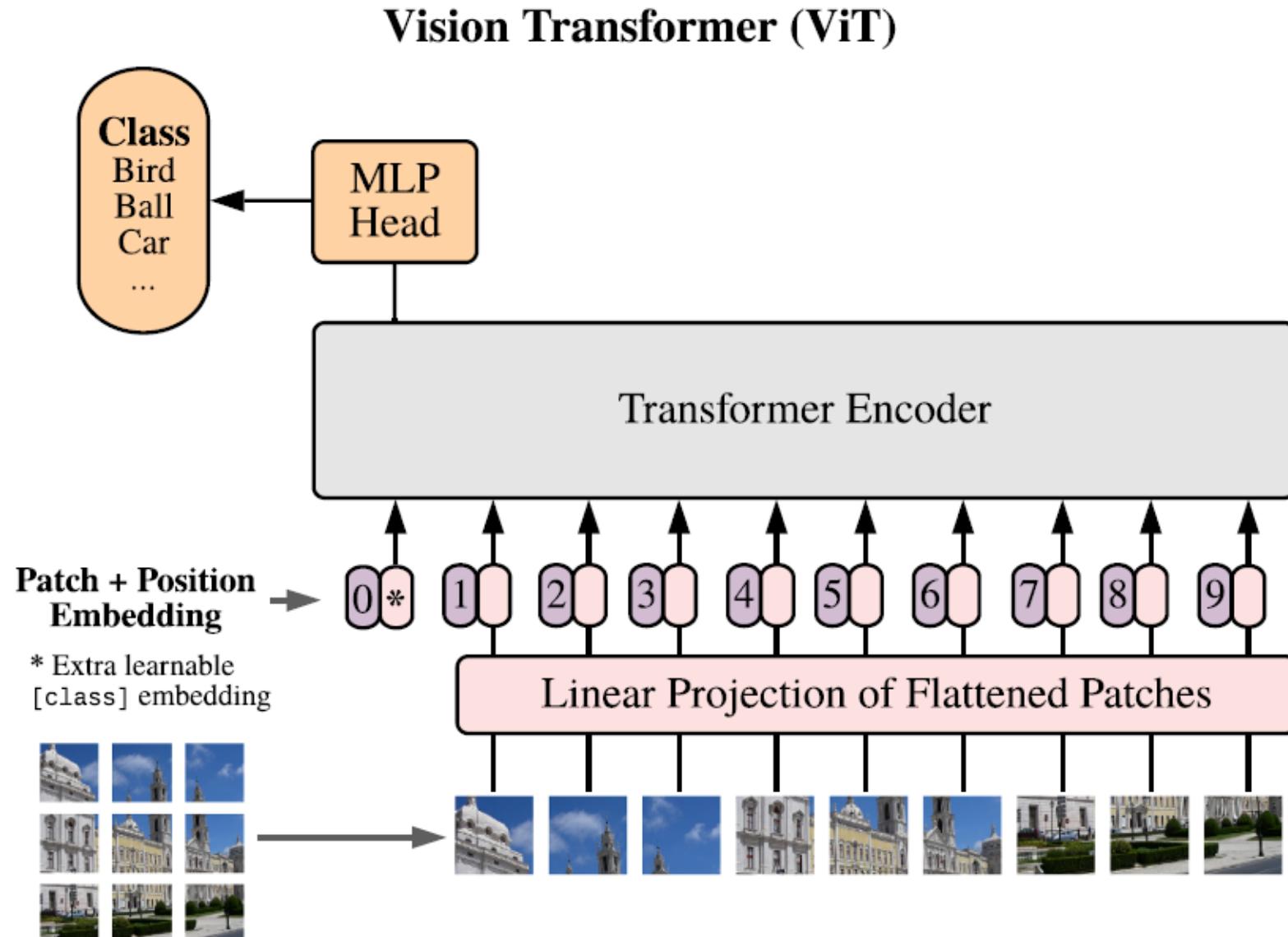
“Large” Datasets:

- Imagenet 21K (21K classes, 14M images a superset of Imagenet 1K)
- JFT-300M (18K classes, 303M images) is an **internal Google dataset** used for training image classification models. Over one billion labels for the 300M images (a single image can have multiple labels) out of which approximately 375M are selected via an algorithm.
- Typically images here are larger (384x384?)

Watch out: vectors are considered as rows in
this paper

... and notation is not consistent at all with the
rest of the slides

Model Overview



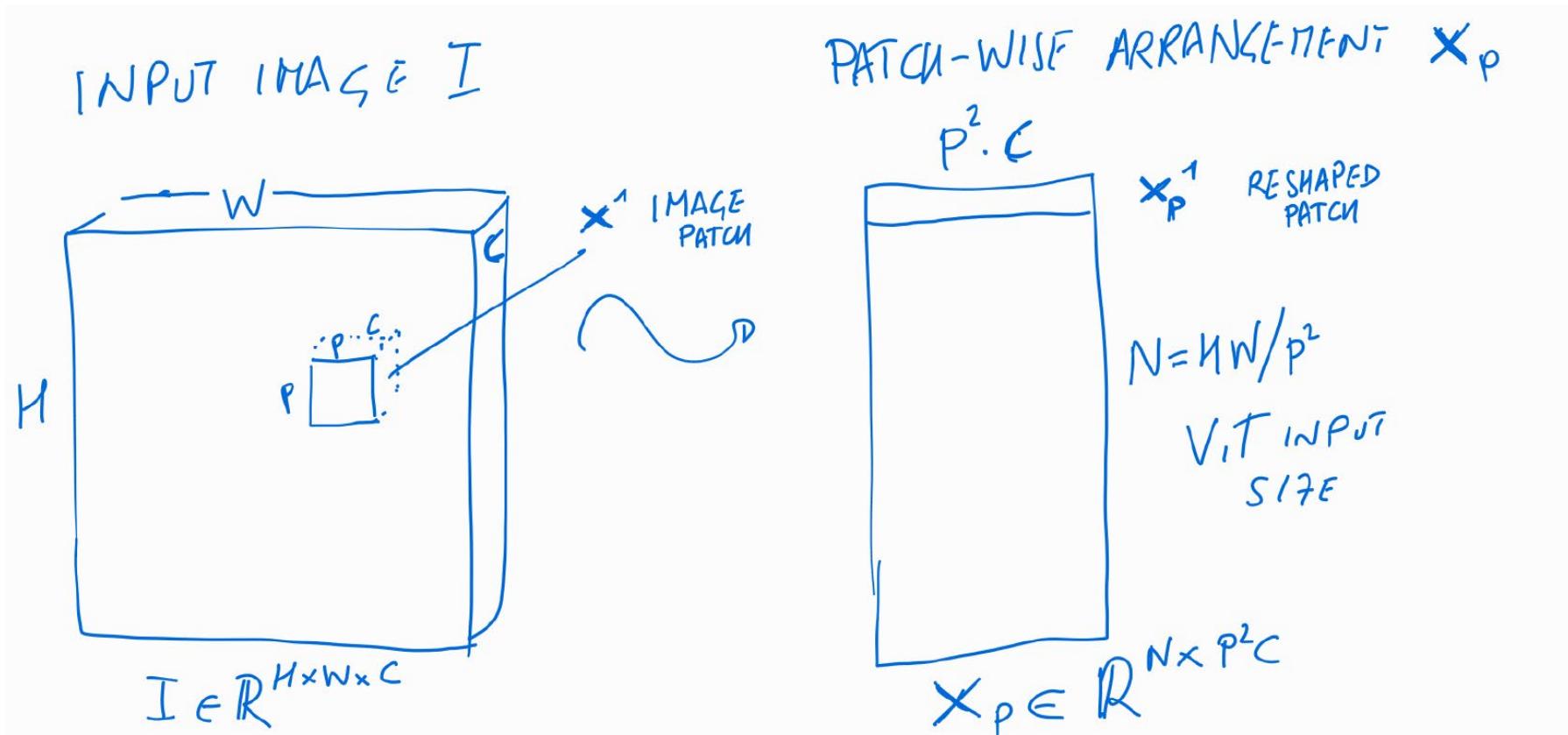
Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). *An image is worth 16x16 words: Transformers for image recognition at scale*. ICLR 2021.

Patch-wise image arrangement

Images $I \in \mathbb{R}^{H \times W \times C}$ are divided in non-overlapping patches

$$x^i \in \mathbb{R}^{P \times P \times C}$$

Each patch is arranged in a row vector of a matrix $x_p \in \mathbb{R}^{N \times P^2 C}$,
being $N = HW/P^2$ the number of extracted patches



Patches are typically
 16×16 , images are
 $\approx 256 \times 256$
hence the title
*An image is worth
 16×16 words..*

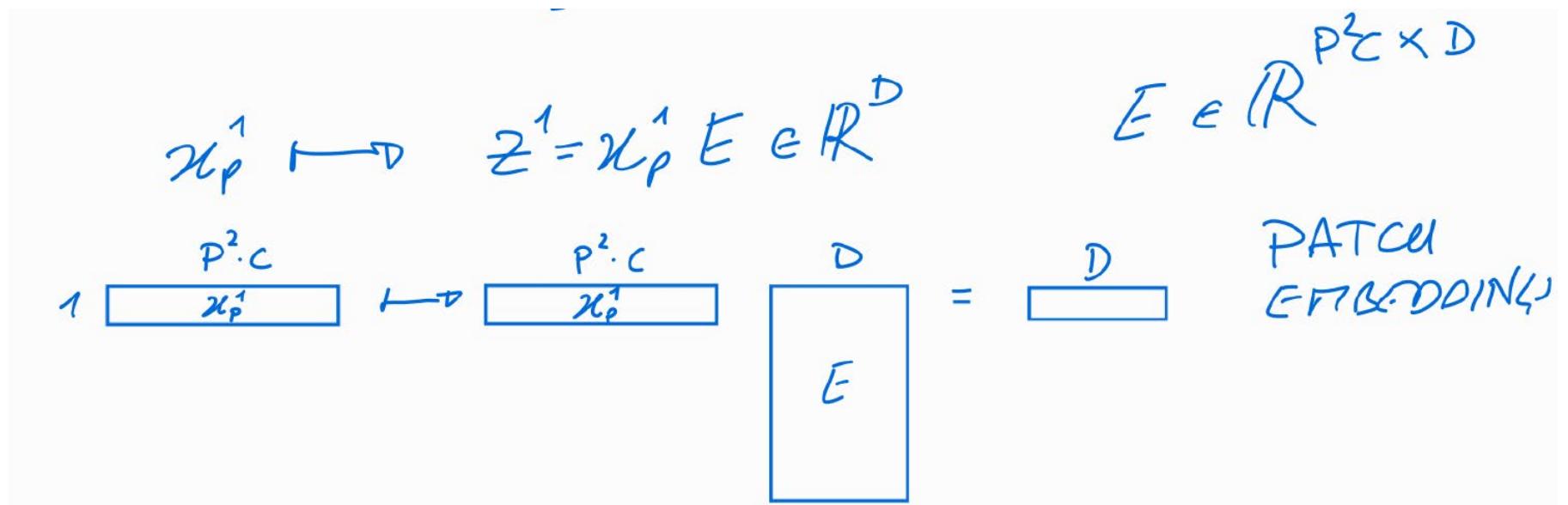
Patch Embedding

The i -th patch is embedded in a D -dimensional vector by a learnable linear mapping (remember these are row vectors)

$$x_p^i \rightarrow Ex_p$$

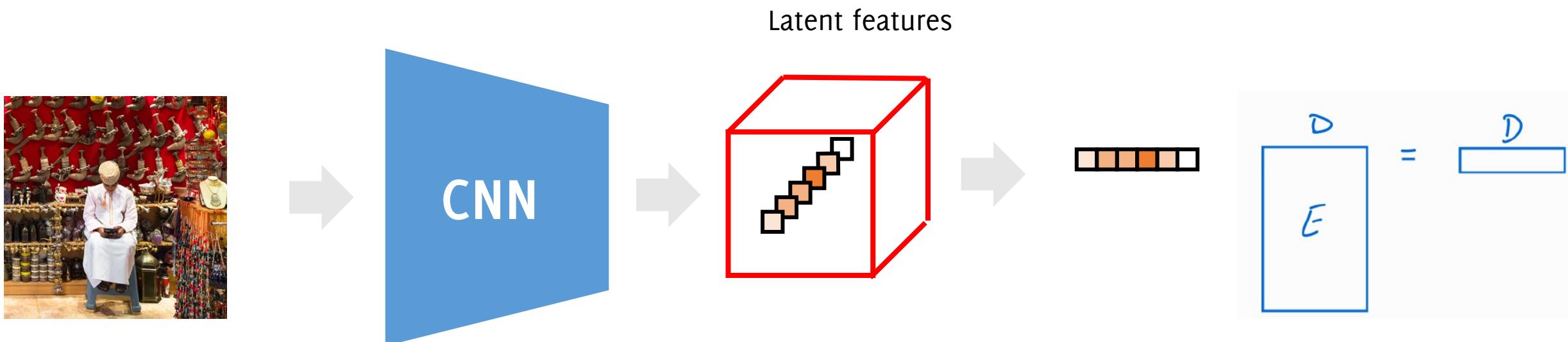
where

$$E \in \mathbb{R}^{3P^2 \times D}$$



Alternative to Linear Embedding

It is also possible to perform the linear embedding on the latent representation of a CNN backbone.



Embedding the entire image

The image is embedded into a matrix

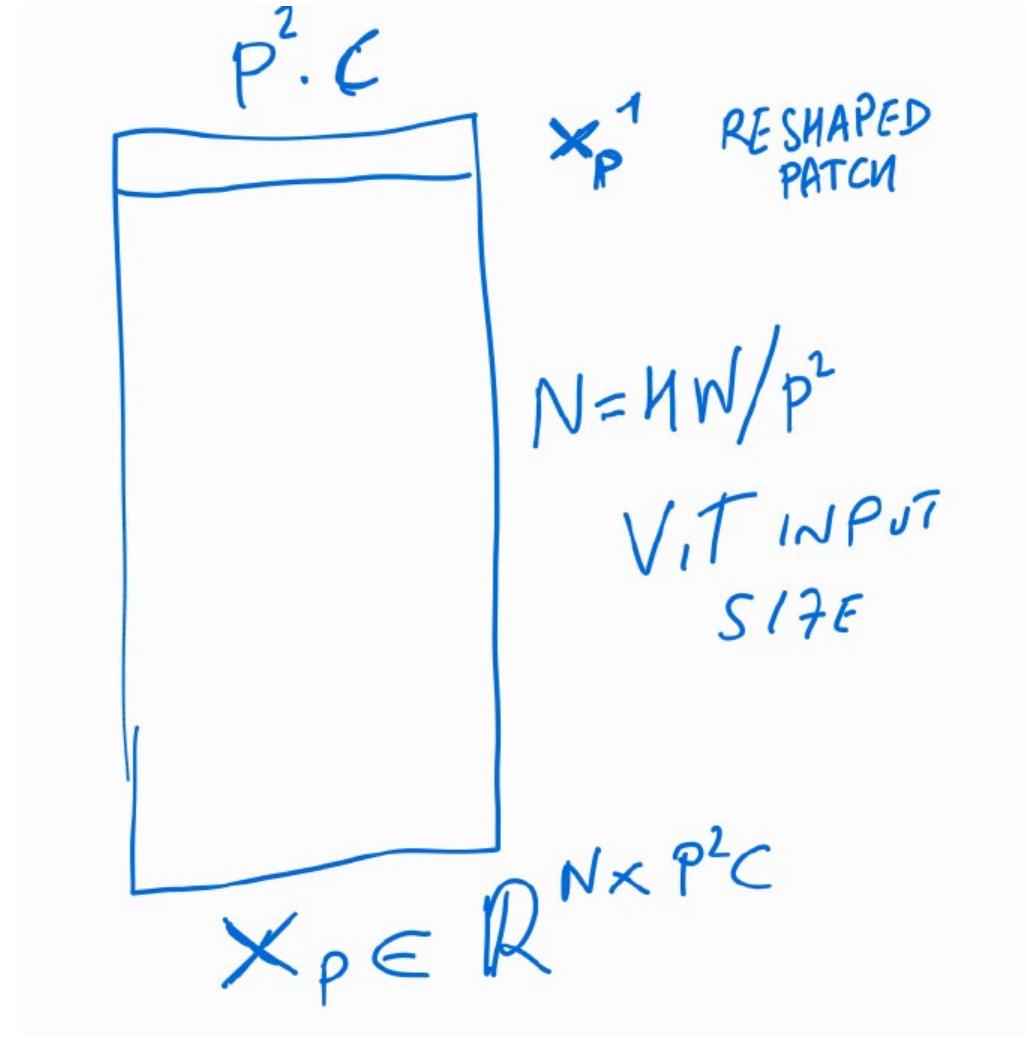
$$x_p \in \mathbb{R}^{N \times P^2 C}$$

where $N = HW/P^2$ the number of patches.

N is equivalent to the **size of a sentence** fed to the transformer and can in principle vary.

N is set to a **maximum value** and in case of short sentences this is padded with zero.

The coefficients of the transformers are not influenced by the input lenght N



Positional Encoding

The positional encoding is *added* to their respective embedded patches

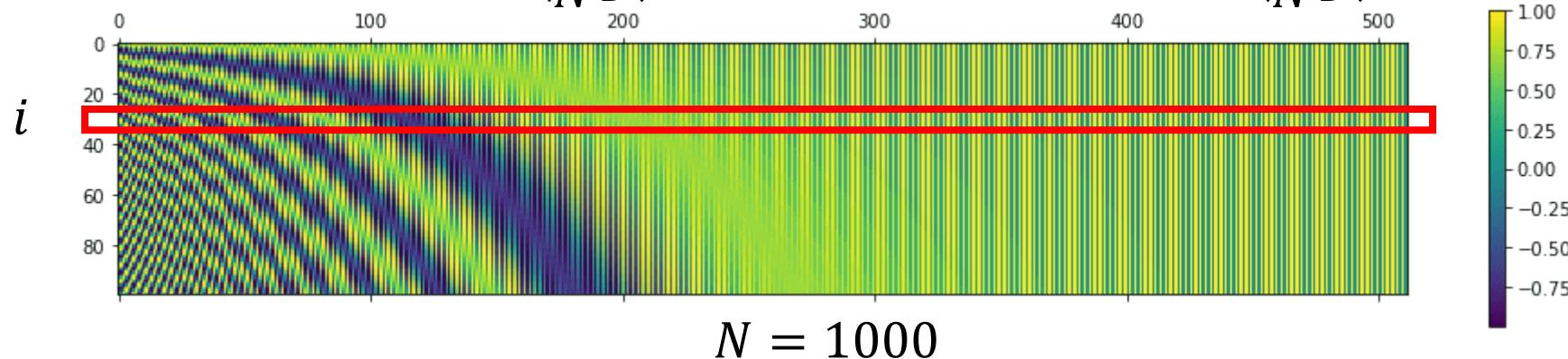
$$I \xrightarrow{\quad} z_0 = \begin{bmatrix} x_{\text{CLASS}}^D \\ z_p^1 \\ \vdots \\ z_p^{N+1} \end{bmatrix} + \begin{bmatrix} E_{\text{pos}}^D \\ \vdots \end{bmatrix}$$

Positional Encoding

Since each patch in an image simultaneously flows through the Transformer, the model itself doesn't have any sense of position/order for the patch in the image.

We can enumerate the position of patches (1-d ordering seems ok) or we can define a mapping from the i -th patch to a D -dimensional vector

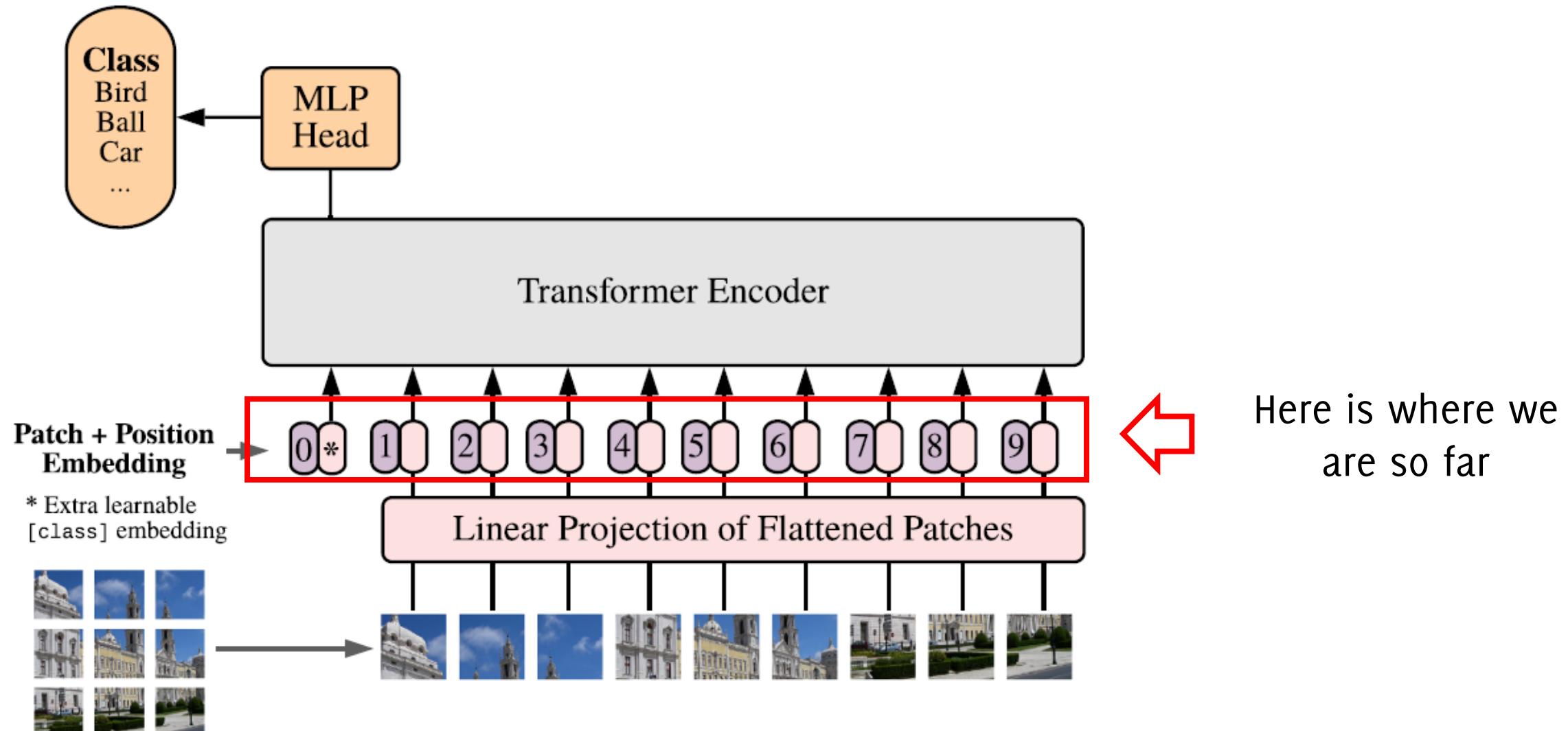
$$PE(i, 2j) = \sin\left(\frac{i}{N^{\frac{D}{2}}}\right), \quad PE(i, 2j + 1) = \cos\left(\frac{i}{N^{\frac{D}{2}}}\right)$$



Or more in general the positional embedding can be learned as a **matrix product** (after such preliminary encoding in a D -dimensional vector)

Model Overview

Vision Transformer (ViT)



Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). *An image is worth 16x16 words: Transformers for image recognition at scale*. ICLR 2021.

Model Output

This is added as an additional learnable embedding in position 0

At the end of the Transformer's stack, this vector is fed to a MLP (possibly with an hidden layer) to predict image class

$$z_0 = \begin{bmatrix} x_{\text{CLASS}} \\ z_p^1 \\ \vdots \\ z_p^N \end{bmatrix} + E_{\text{pos}}$$

Model Output

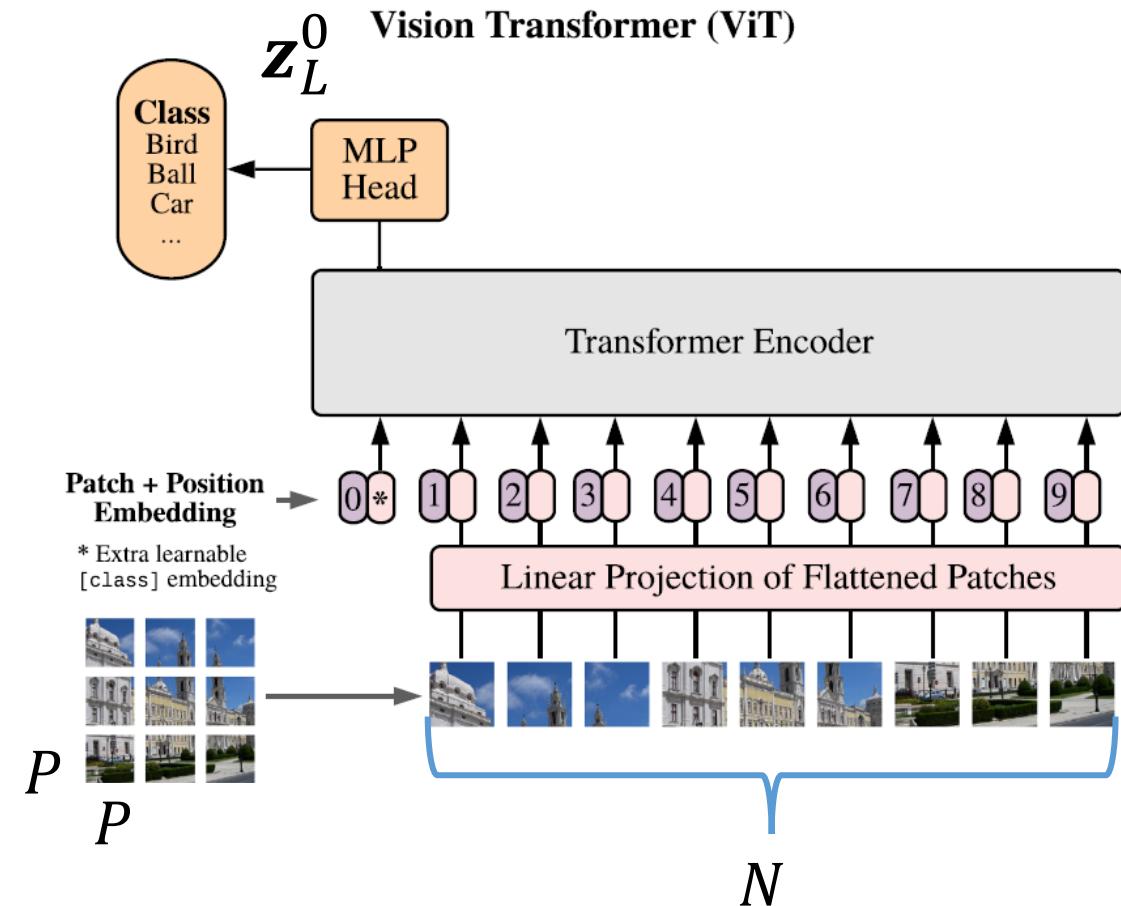
The entire image is embedded into

$$z_0 = [label; \dots; \dots] + E_{pos}$$

Where *label* is the label to be predicted.

Since ViT will be trained like BERT, it will
infer the missing word, corresponding to the
label.

A classification head is attached to z_L^0



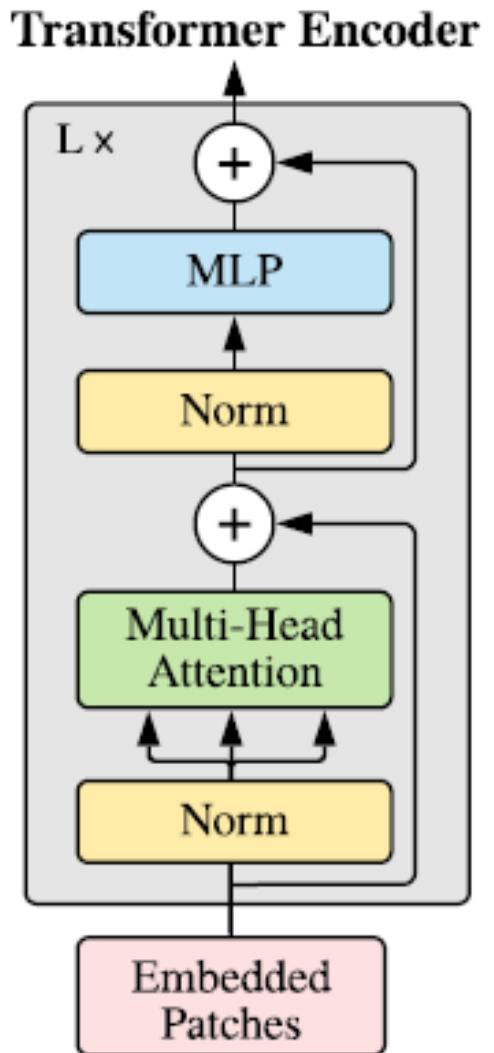
Inside the Transformer Block

It alternates between

- A Multiheaded Self Attention (MSA) layer
- A Multi-layer Perceptrons (MLP) layer
- Including a LayerNorm in betwevn

This a **plain vanilla transformer** as for text

(only the input changes in ViT)



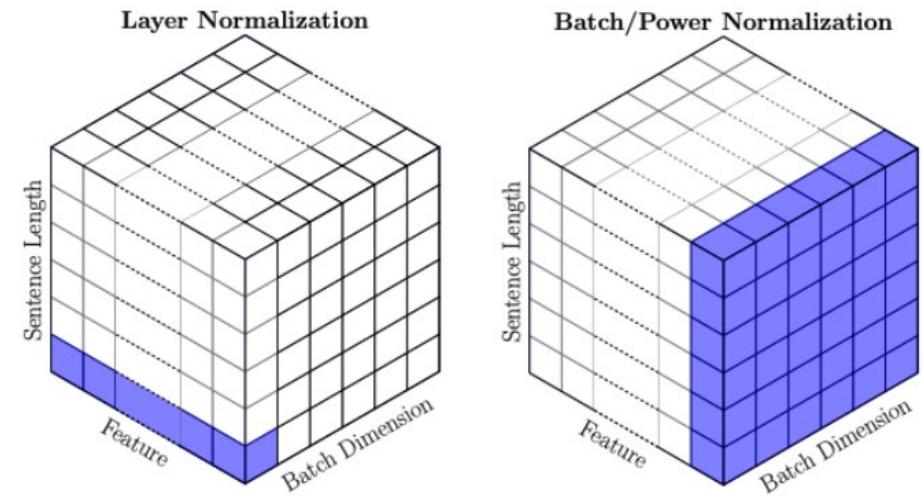
LayerNorm

Standardize (like z-score subtracting the mean and dividing by the variance) all the entries of each representation (ignoring the batch-dimension).

Involves all the activations in a layer for a single training sample.

$$\mu^l = \frac{1}{H} \sum_{i=1}^H a_i^l \quad \sigma^l = \sqrt{\frac{1}{H} \sum_{i=1}^H (a_i^l - \mu^l)^2}$$

Like batch normalization, it is possible to learn bias and offset after that.

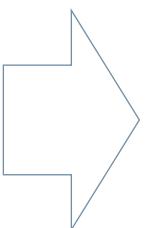


This is meant to **speed up** training R-NN and Transformer architecture

Self-Attention

$$\begin{matrix} x_1 \\ x_2 \end{matrix} \begin{bmatrix} \text{light green} & \text{light green} & \text{light green} \\ \text{light green} & \text{light green} & \text{light green} \\ \text{light green} & \text{light green} & \text{light green} \end{bmatrix} \times \begin{bmatrix} \text{white} & \text{white} & \text{white} \\ \text{white} & \text{white} & \text{white} \end{bmatrix} = \begin{matrix} q_1 \\ q_2 \end{matrix} \begin{bmatrix} \text{blue} & \text{blue} & \text{blue} \\ \text{blue} & \text{blue} & \text{blue} \end{bmatrix}$$

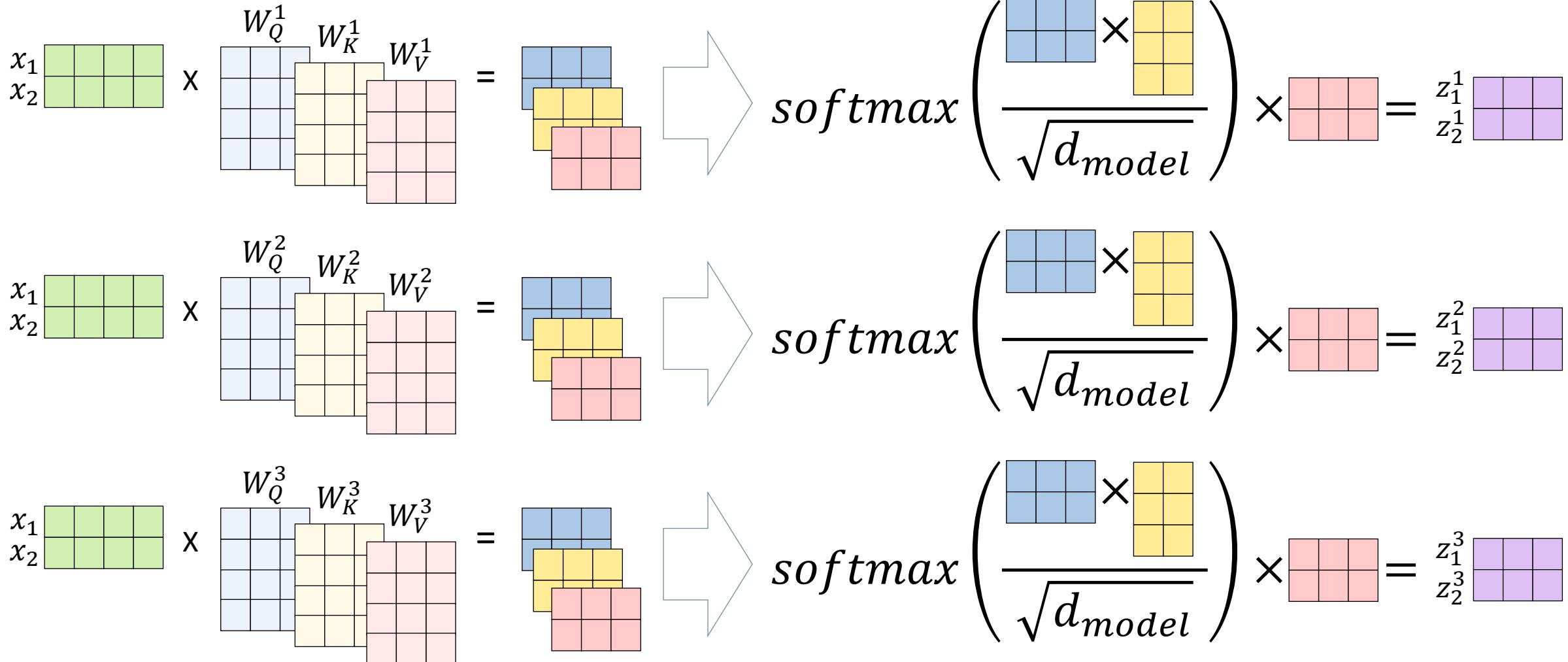
$$\begin{matrix} x_1 \\ x_2 \end{matrix} \begin{bmatrix} \text{light green} & \text{light green} & \text{light green} \\ \text{light green} & \text{light green} & \text{light green} \\ \text{light green} & \text{light green} & \text{light green} \end{bmatrix} \times \begin{bmatrix} \text{yellow} & \text{yellow} & \text{yellow} \\ \text{yellow} & \text{yellow} & \text{yellow} \\ \text{yellow} & \text{yellow} & \text{yellow} \\ \text{yellow} & \text{yellow} & \text{yellow} \end{bmatrix} = \begin{matrix} k_1 \\ k_2 \end{matrix} \begin{bmatrix} \text{yellow} & \text{yellow} & \text{yellow} \\ \text{yellow} & \text{yellow} & \text{yellow} \end{bmatrix}$$



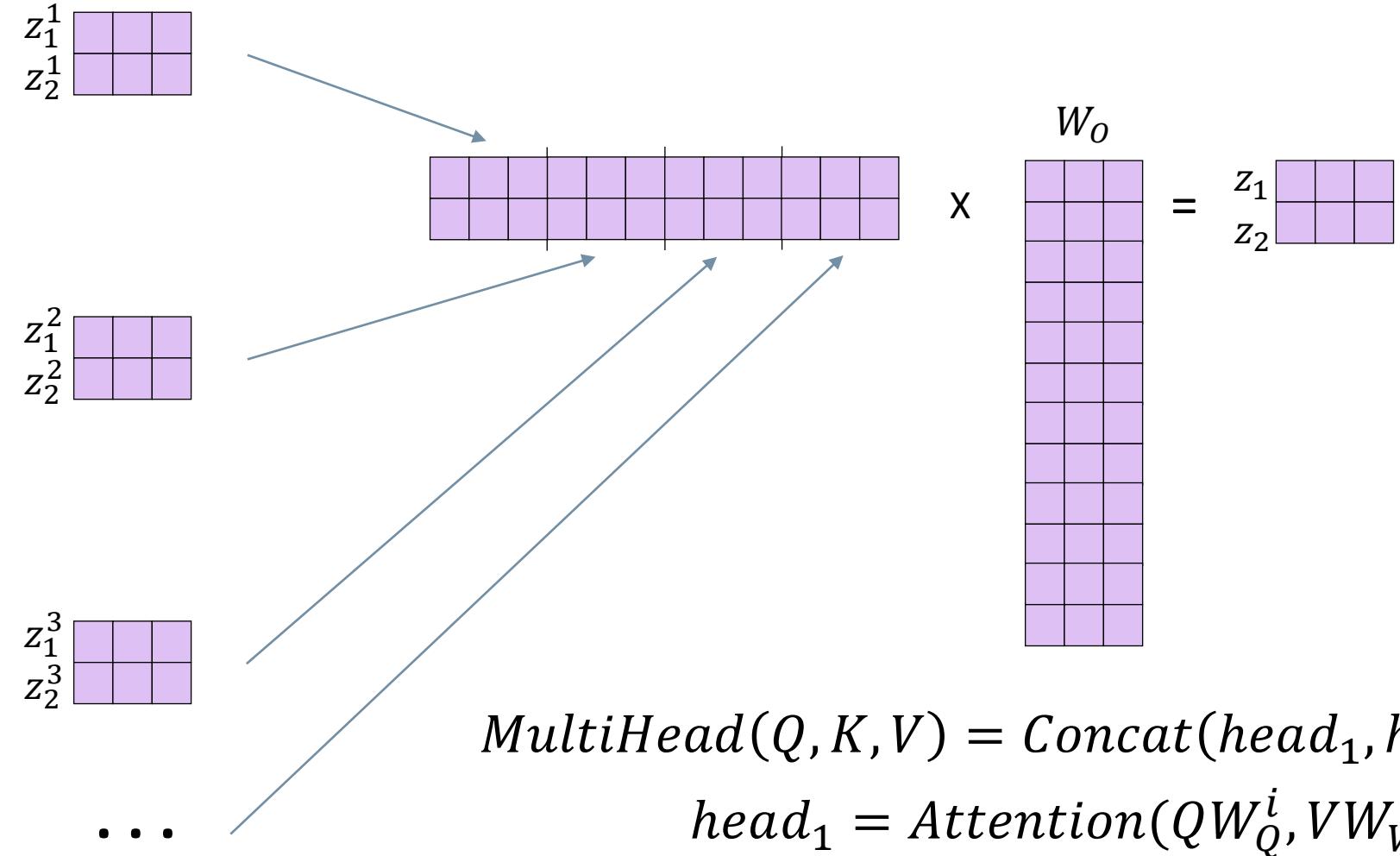
$$softmax\left(\frac{\begin{bmatrix} \text{blue} & \text{blue} & \text{blue} \\ \text{blue} & \text{blue} & \text{blue} \end{bmatrix} \times \begin{bmatrix} \text{yellow} & \text{yellow} & \text{yellow} \\ \text{yellow} & \text{yellow} & \text{yellow} \end{bmatrix}}{\sqrt{d_{model}}}\right) \times \begin{bmatrix} \text{pink} & \text{pink} & \text{pink} \\ \text{pink} & \text{pink} & \text{pink} \end{bmatrix} = \begin{bmatrix} \text{purple} & \text{purple} & \text{purple} \\ \text{purple} & \text{purple} & \text{purple} \end{bmatrix}$$

$$\begin{matrix} x_1 \\ x_2 \end{matrix} \begin{bmatrix} \text{light green} & \text{light green} & \text{light green} \\ \text{light green} & \text{light green} & \text{light green} \\ \text{light green} & \text{light green} & \text{light green} \end{bmatrix} \times \begin{bmatrix} \text{pink} & \text{pink} & \text{pink} \\ \text{pink} & \text{pink} & \text{pink} \\ \text{pink} & \text{pink} & \text{pink} \\ \text{pink} & \text{pink} & \text{pink} \end{bmatrix} = \begin{matrix} v_1 \\ v_2 \end{matrix} \begin{bmatrix} \text{pink} & \text{pink} & \text{pink} \\ \text{pink} & \text{pink} & \text{pink} \end{bmatrix}$$

Multiheaded Self-Attention (MSA)



Multiheaded Self-Attention (MSA)



Summarizing

These operations are executed in a sequence of L blocks.

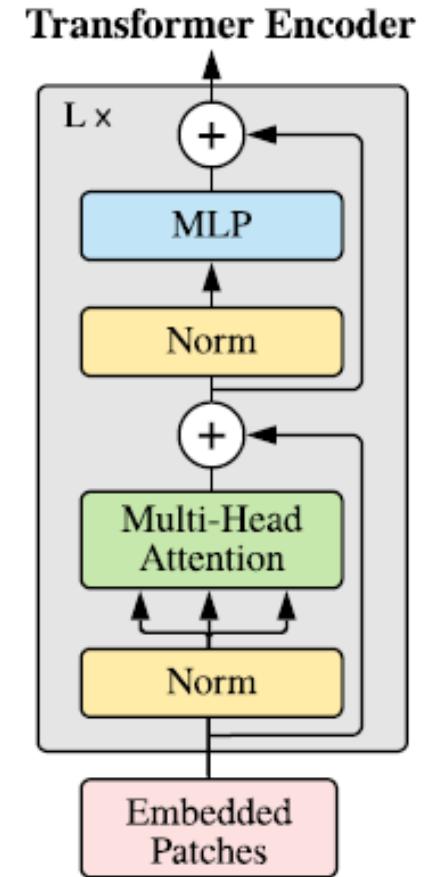
Each block contains residual connections

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \mathbf{x}_p^2 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{\text{pos}}, \quad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1) \times D}$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \quad \ell = 1 \dots L$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \quad \ell = 1 \dots L$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0)$$



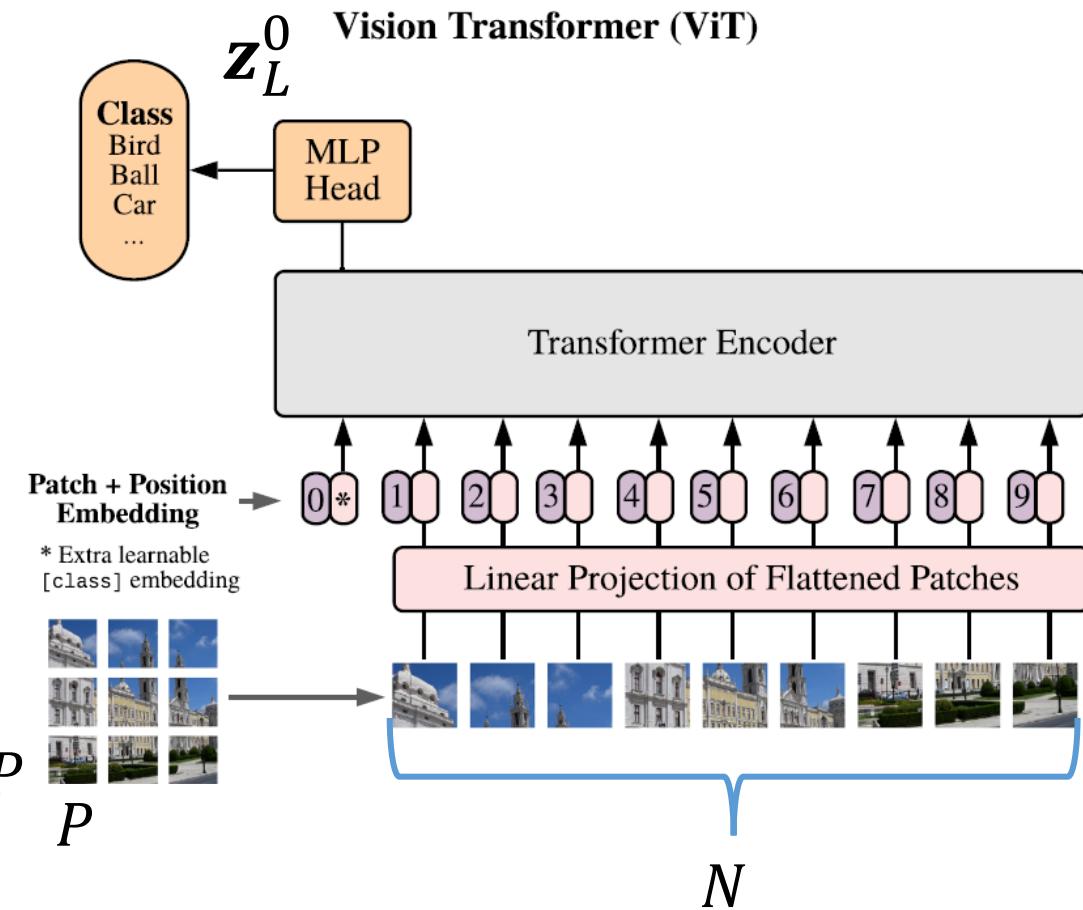
ViT Training

Pre-training: train ViT as a classifier on the huge dataset with the MLP on the top made of a single hidden layer

Fine-Tuning: the MLP is discarded and replaced by a linear layer mapping to the K output classes of the specific classification problem

When operating on higher resolution images, the patch-size P cannot be changed.

The same model is used on a longer sequence P but the positional embeddings are linearly interpolated



Inductive bias in ViT

ViT include much **fewer image-specific inductive bias than a CNN** (locality, translation invariance... these properties are granted by design to the CNN)

Here the only locality clues are:

- locality, image is always used batch-wise (patchify layer)
- Interpolation of 2D positional embeddings (to adjust location in images)...

However, there are no substantial differences between 1D or 2D positional embeddings...

Trained Models

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Table 1: Details of Vision Transformer model variants.

Classification Performance

ViT-[MODEL_TYPE] / [PATCH_SIZE]

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	88.55 ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReAL	90.72 ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	99.50 ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	—
CIFAR-100	94.55 ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	—
Oxford-IIIT Pets	97.56 ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	—
Oxford Flowers-102	99.68 ± 0.02	99.74 ± 0.00	99.61 ± 0.02	99.63 ± 0.03	—
VTAB (19 tasks)	77.63 ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	—
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k



This ICCV paper is the Open Access version, provided by the Computer Vision Foundation.

Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

Swin Transformer: Hierarchical Vision Transformer using Shifted Windows

Ze Liu^{1,2†*} Yutong Lin^{1,3†*} Yue Cao^{1*} Han Hu^{1*‡} Yixuan Wei^{1,4†}
Zheng Zhang¹ Stephen Lin¹ Baining Guo¹

¹Microsoft Research Asia ²University of Science and Technology of China
³Xian Jiaotong University ⁴Tsinghua University

{v-zeliul,v-yutlin,yuecao,hanhu,v-yixwe,zhez,stevelin,bainguo}@microsoft.com

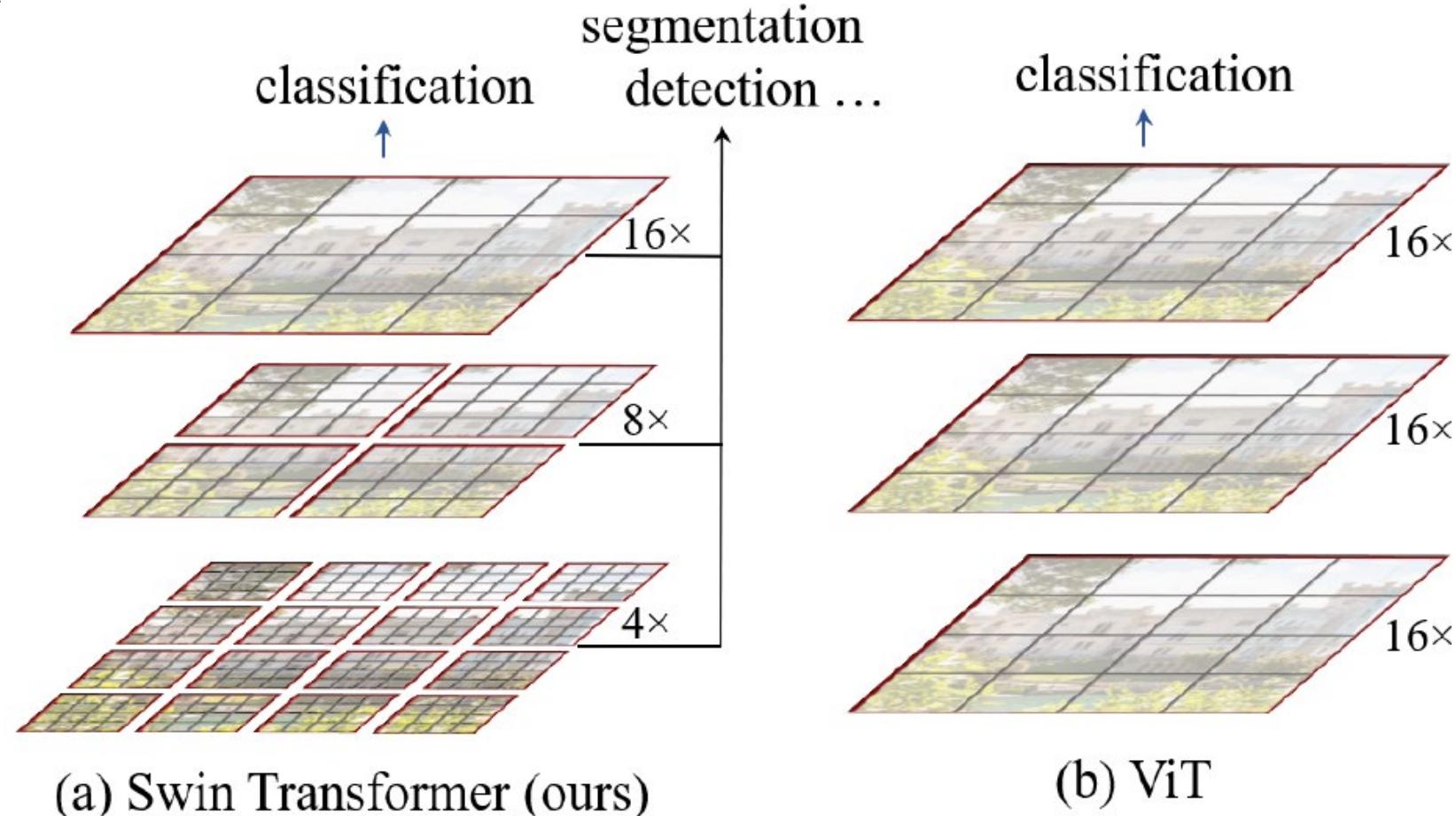
Vanilla ViT models

- achieve state-of-the-art performance in Image Classification when trained on very large datasets
- The only **image-specific inductive bias** consists in the «patchify» layer, which crops and embed image patches.
- As a matter of fact, **ViT backbones do not excell** when solving sophisticated vision tasks (e.g. object detection).
- Intractable for high-resolution images, due to the quadratic complexity of global attention mechanism

Next Step: expand the Transformers to serve as a general-purpose backbone for computer vision, as it does for NLP and as CNNs do in vision.

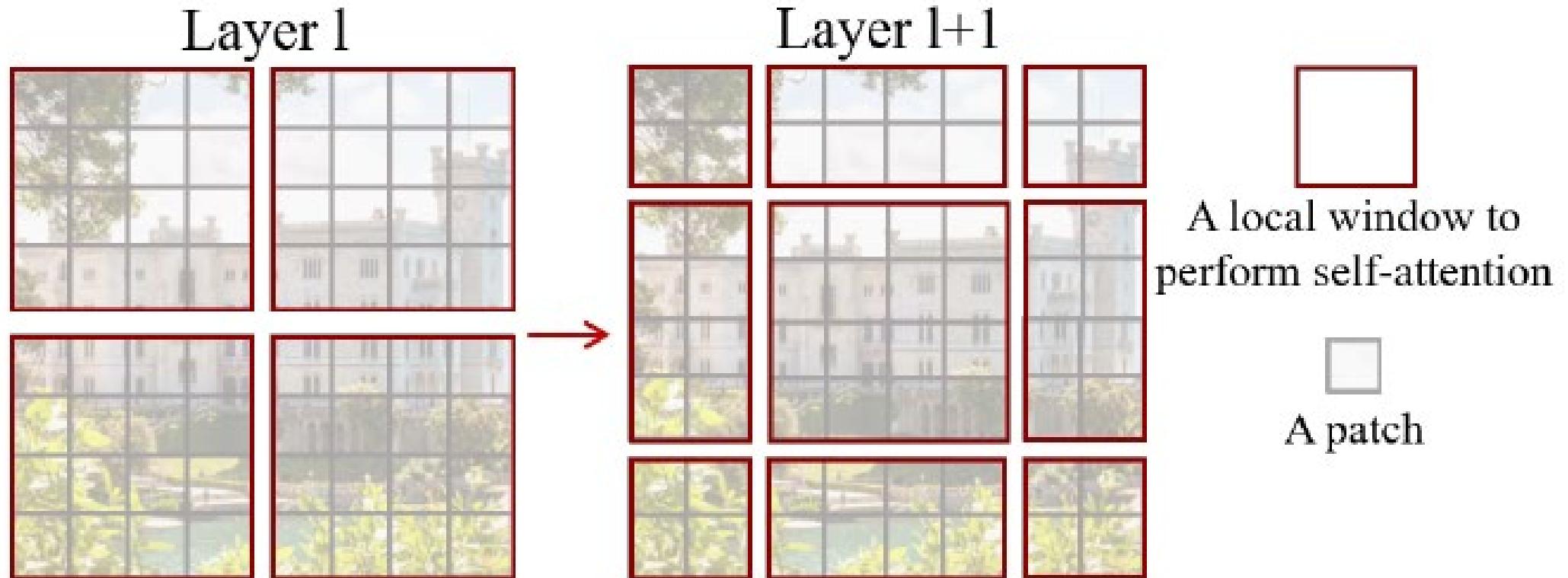
Hierarchical (Swin) Transformers

- Constructs a hierarchical feature map to address scale variations in images
- Provides accurate dense predictions also on high-resolution images
- achieves strong performance on image classification, object detection and semantic segmentation



Hierarchical (Swin) Transformers

Introduces **attention within local windows**: performs sliding window / local processing like CNNs. This result in connections among different attention regions



Combining Text and Images

Advanced Classification Settings

Closed Set Classification Settings (the standard)

You are given a training set

$$TR = \{(\mathbf{x}(t), y(t)), \mathbf{x} \in \mathbb{R}^d, y \in \Lambda\}$$

You train a classifier

$$\mathcal{K}: \mathbb{R}^d \rightarrow \Lambda$$

Which has to classify instances having label Λ .

Open Set Recognition Settings

The classifier \mathcal{K} at test time need to recognize possible input samples that do not belong to any class in Λ

$$\mathcal{K}: \mathbb{R}^d \rightarrow \{\Lambda, "unknown"\}$$

These are more general settings and require changing \mathcal{K} after training or adopting specific training procedures

Advanced Classification Settings

Open set recognition requires specific model or training procedures

- Even though one of the simplest solution seems to be monitoring the maximum of logit!

Still, this does not provide any hint more than “unknown” for images that do not belong to Λ .

... a much more appealing alternative would be to train a computer vision model having “natural language” as a target.

Advanced Classification Settings

Zero Shot Learning at test time, the classifier observes samples from classes do not belong to Λ , and **needs to predict the class that they belong to!**

This looks impossible... indeed Zero-shot learning leverage additional information to exploit regularities in the data

- e.g. classes are accompanied by pre-defined structured description (e.g. “red head” description to unseen bird classes)
- Free textual description (“zebra is a striped horse”)
- Class similarity in the embedding

Few Shot Learning: a very small amount of annotated training samples (e.g. a dozen) is provided for fine tuning the model



Learning Transferable Visual Models From Natural Language Supervision

Alec Radford^{*1} Jong Wook Kim^{*1} Chris Hallacy¹ Aditya Ramesh¹ Gabriel Goh¹ Sandhini Agarwal¹
Girish Sastry¹ Amanda Askell¹ Pamela Mishkin¹ Jack Clark¹ Gretchen Krueger¹ Ilya Sutskever¹

CLIP: Contrastive Language Image Pre-Training

Idea: *Learning perception from the supervision contained in natural language paired with images.*

What kind of «natural language paired with images»?

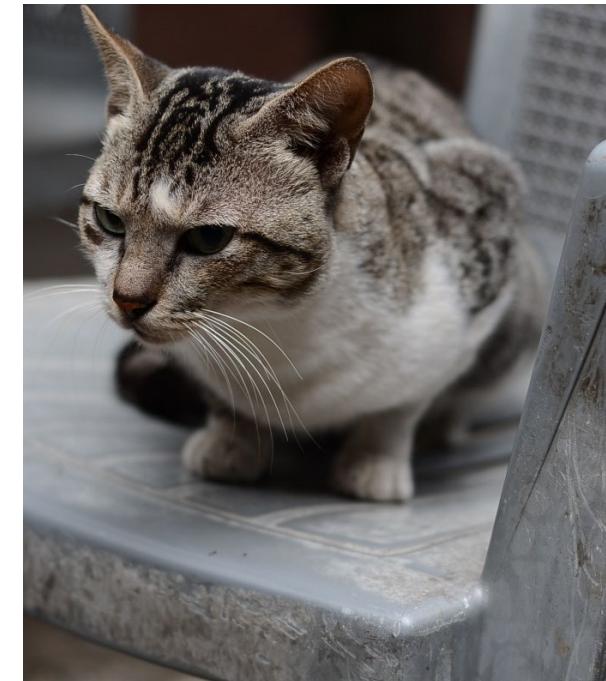
CLIP: Contrastive Language Image Pre-Training

Idea: *Learning perception from the supervision contained in natural language paired with images.*

What kind of «natural language paired with images»?

Image Captions, of course!

WIT: WebImageText, a new dataset of 400 million (image, text) pairs collected from publicly available sources on the Internet, having the text answering among 500K queries.



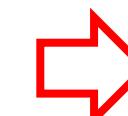
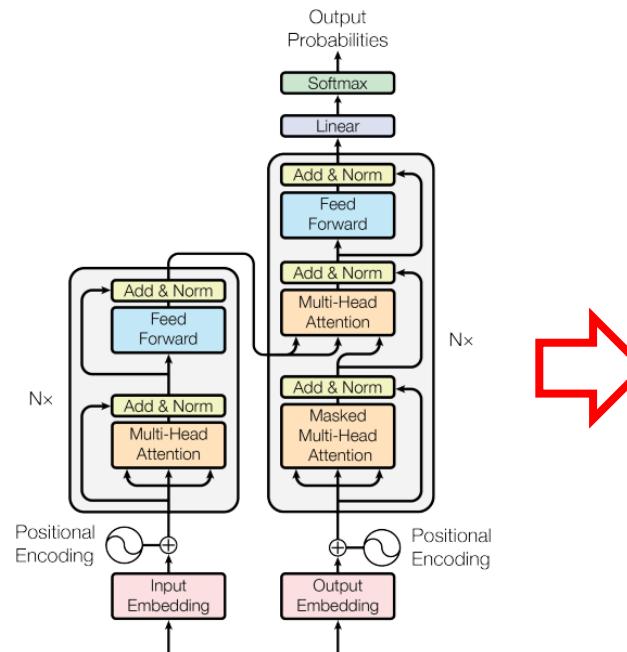
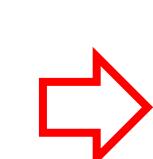
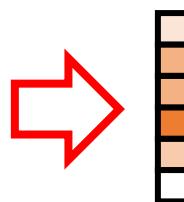
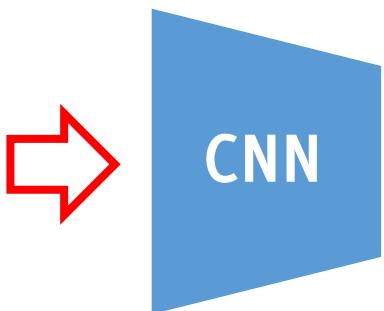
A cat sitting on a chair

Captioning

Jointly train two networks:

- a transformer for text
- a CNN for images / a ViT

predict the exact words of each caption given the input image



*A cat sitting
on a chair*

CLIP: Contrastive Language Image Pre-Training

In order to learn visual representation by text supervision there are better training options.

Different formulation of the problem: train models to predict **which text “as a whole” is paired with which image** (given a batch).

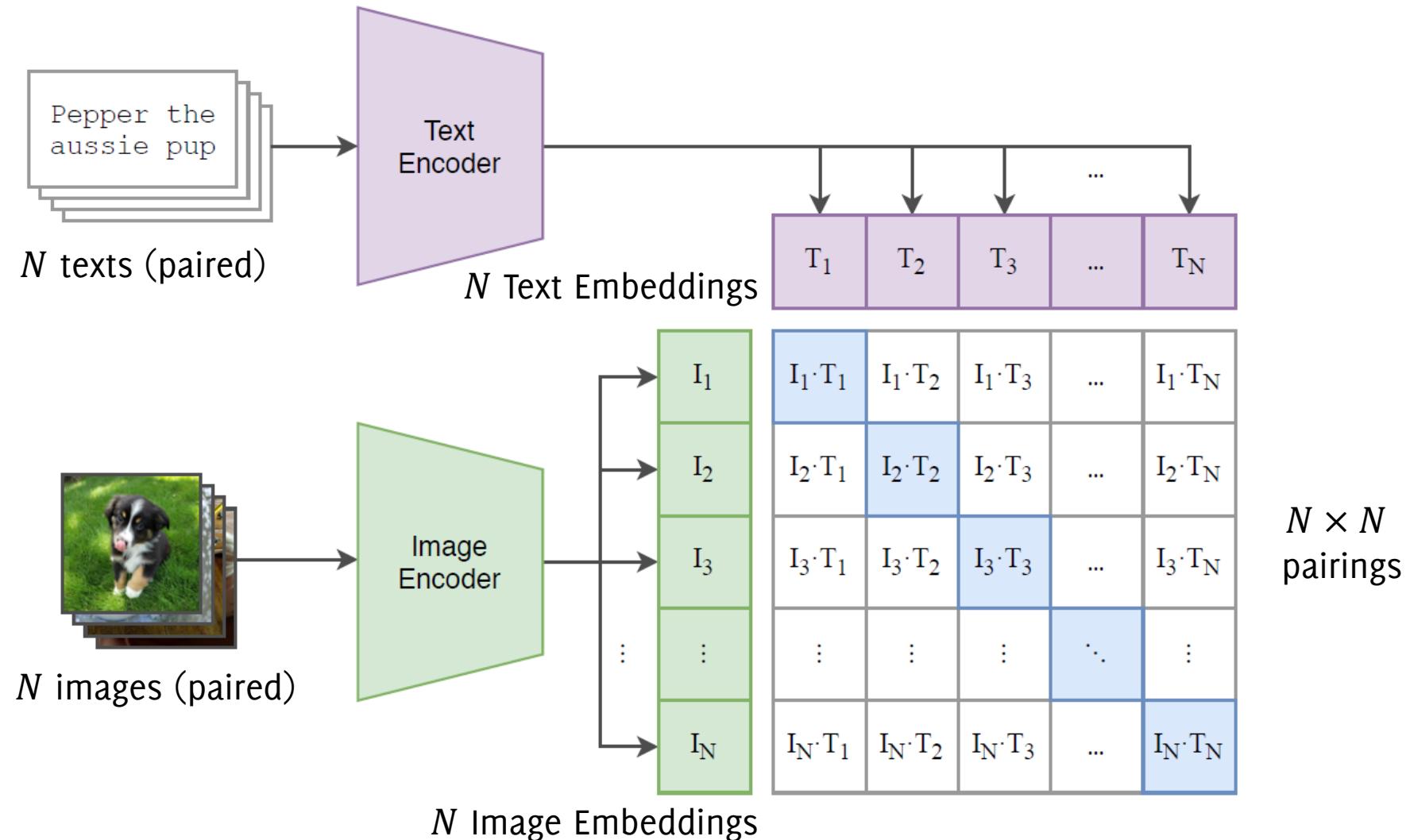
Models trained by minimizing **the contrastive loss** were shown to **outperform models trained to predict the equivalent objective**.

CLIP: Contrastive Language Image Pre-Training

CLIP jointly trains:

- an image encoder
- a text encoder

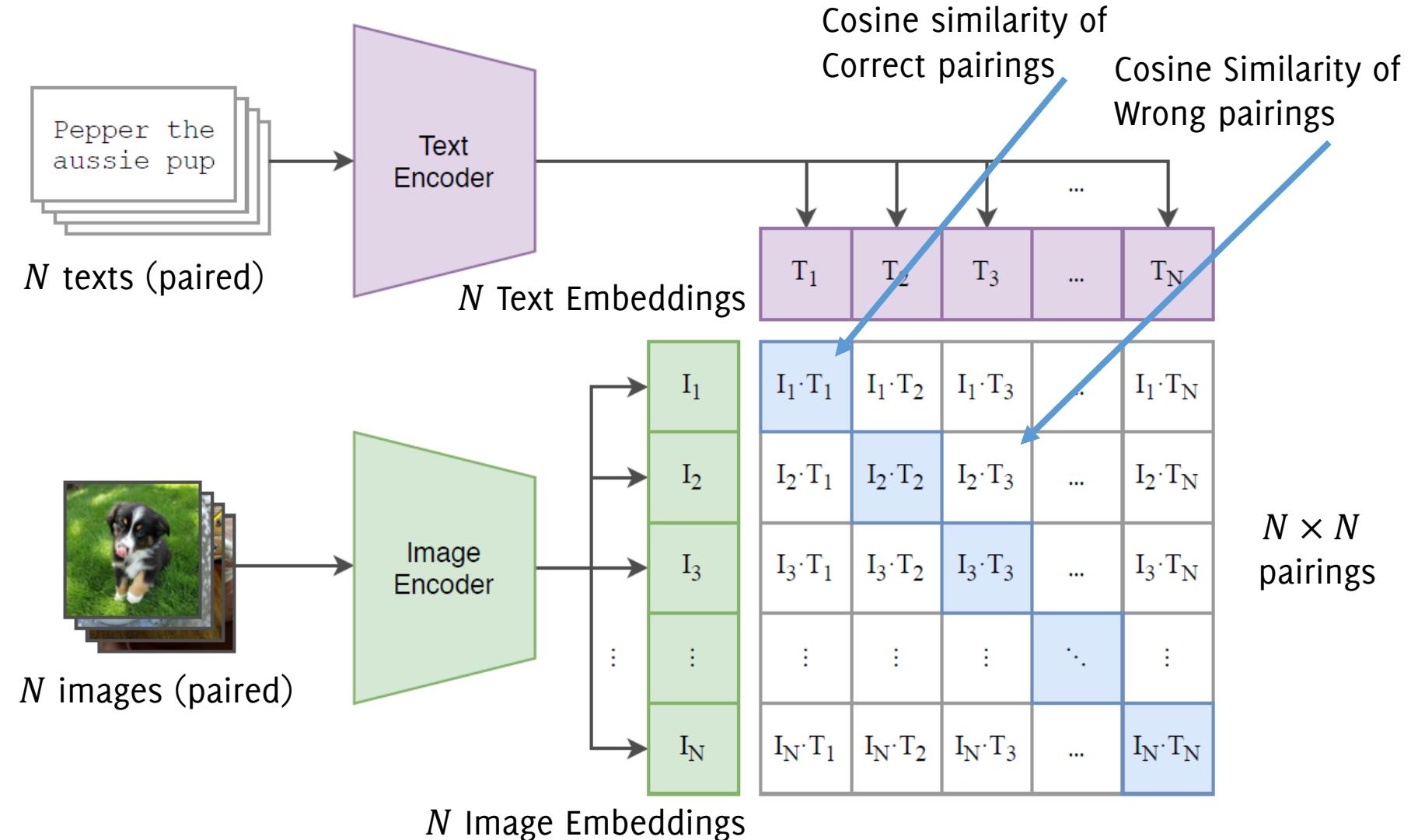
to predict the correct pairings of (image, text) examples



CLIP: Contrastive Language Image Pre-Training

Maximize the cosine similarity among the N correct pairings (i.e. the diagonal entries of the matrix)

Minimize the cosine similarity at the $N^2 - N$ wrong pairings (i.e. the off-diagonal entries of the matrix)



CLIP: Contrastive Language Image Pre-Training

- no pre-training needed,
- the only form of augmentation is image cropping

Two learnable feature extractors.

Extract features from the entire batch.

Two learnable linear projections to compute the embeddings.

Compute cosine similarities

Sum the off-diagonal and diagonal loss using different labels

```
# image_encoder - ResNet or Vision Transformer
# text_encoder  - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l]       - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t              - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T)  #[n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)

# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss   = (loss_i + loss_t)/2
```

CLIP: Contrastive Language Image Pre-Training

Ideally, we would like to have a matrix like this.

Entries are always in [0,1]:

- 0 when embeddings are orthogonal,
- 1 when they are parallel

Entries do not sum to 1

- Both columns and rows of the perfect association can be seen as 1-hot encoding
- Departure from this ideal solution can be assessed by categorical cross-entropy.
- This is how you implement in Py

$I_1 \cdot 1 T_1$	$I_1 \cdot 0 T_2$	$I_0 \cdot T_3$	$0 \dots$	$0 \cdot T_N$
$I_2 \cdot 0_1$	$I_2 \cdot 1 T_2$	$I_2 \cdot 0_3$	$0 \dots$	$I_0 \cdot T_N$
$I_3 \cdot 0_1$	$I_0 \cdot T_2$	$I_3 \cdot 1 T_3$	$0 \dots$	$I_3 \cdot 0_N$
0	0	0	1	0
$I_N \cdot 0 T_1$	$I_0 \cdot T_2$	$I_0 \cdot T_3$	$0 \dots$	$I_N \cdot 1 T_N$

```
# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss = (loss_i + loss_t)/2
```

Zero-Shot Classifier

CLIP is pre-trained to predict if an image and a text are paired together in WIT... that's not very useful by itself!

However, it enables **Zero-Shot / Few-Shot learning**, say having a model to predict classes of unknown events with zero/very little annotated training examples.

Testing zero-shot learning is a way to study the generalization capabilities of a model to unseen datasets. That's a quantitative assessment of task-learning capabilities.

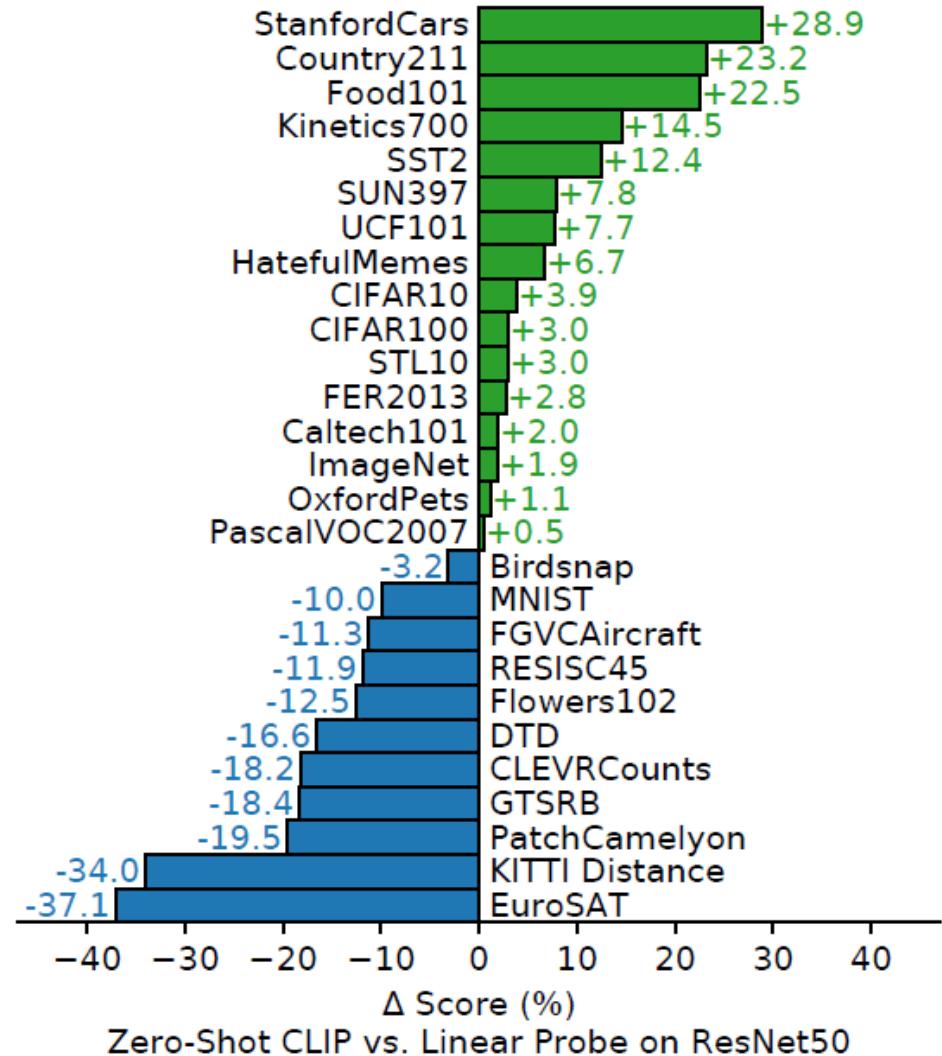
Zero-Shot CLIP

- Take pre-trained CLIP
- Consider the class names as text $\{t_i\}$
- Classify an image I by associating it to the text $\{t_i\}$ that is closest to I according to CLIP

Zero-Shot Classification Performance

Baseline: fully supervised classifier (logistic regression) fitted on the ResNet50 features-

Zero-shot CLIP classifier outperforms a fully supervised classifier on 16 datasets, including ImageNet.

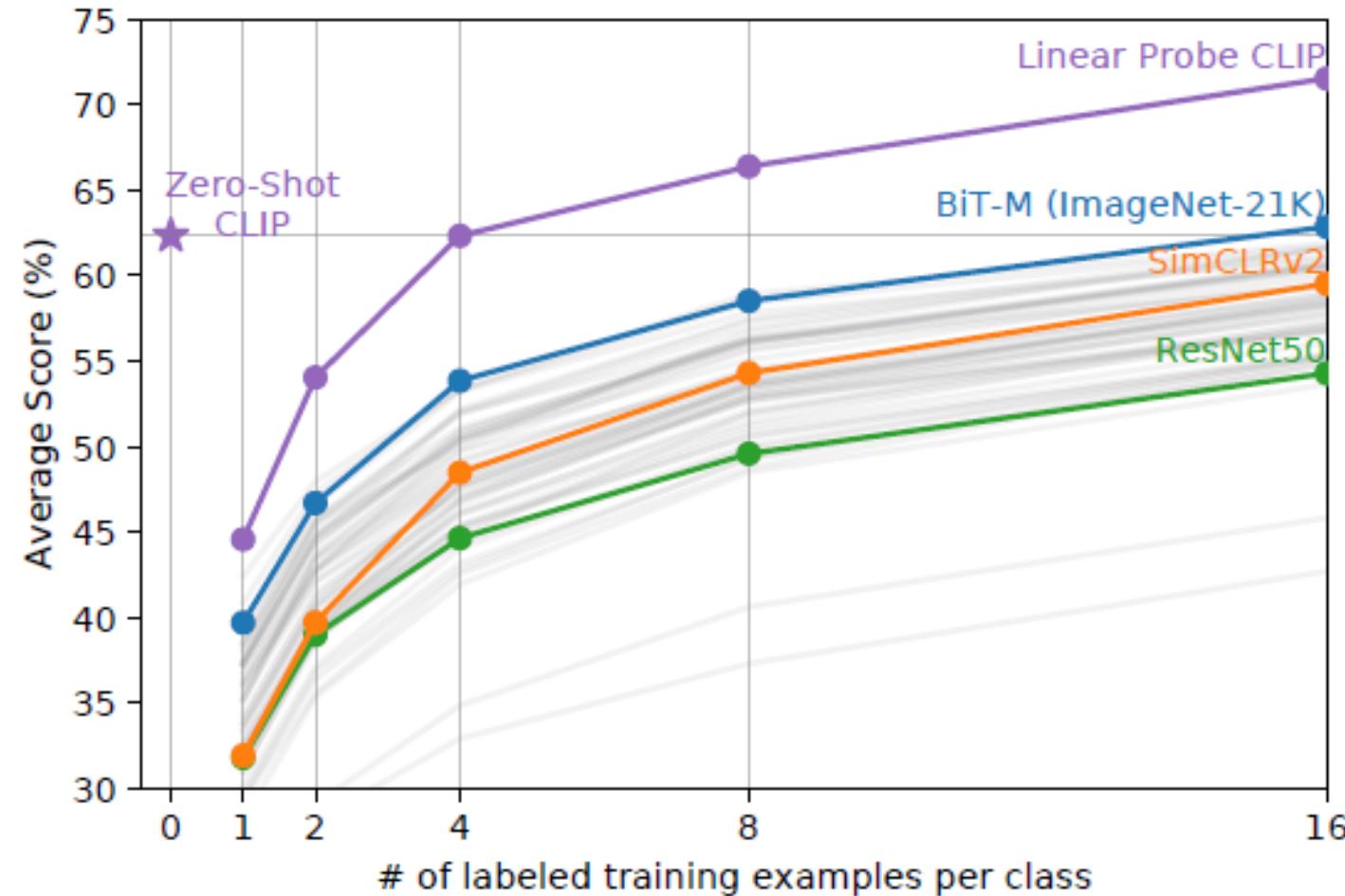


Few-Shot Classification Performance

Comparison against few shot learning solution, where a logistic regression is trained on features from other models.

Zero-shot CLIP matches the performance of 4-shot logistic regression on the same feature space

- Zero-shot CLIP classifier is generated via natural language, and allows to directly “communicate” visual concepts
- Supervised learning must infer concepts indirectly from training examples



Clip Limitations

- Around **1000x increase in compute is required** for zero-shot CLIP to reach overall SOTA performance in all the dataset.
- Many complex visual recognition tasks are difficult to specify just through text
- There is a counter-intuitive drop in performance when transitioning from a zero-shot to a few-shot setting

More on Zero-Shot Learning by CLIP

Published as a conference paper at ICLR 2022

LANGUAGE-DRIVEN SEMANTIC SEGMENTATION

Boyi Li

Cornell University, Cornell Tech

Kilian Q. Weinberger

Cornell University

Serge Belongie

University of Copenhagen

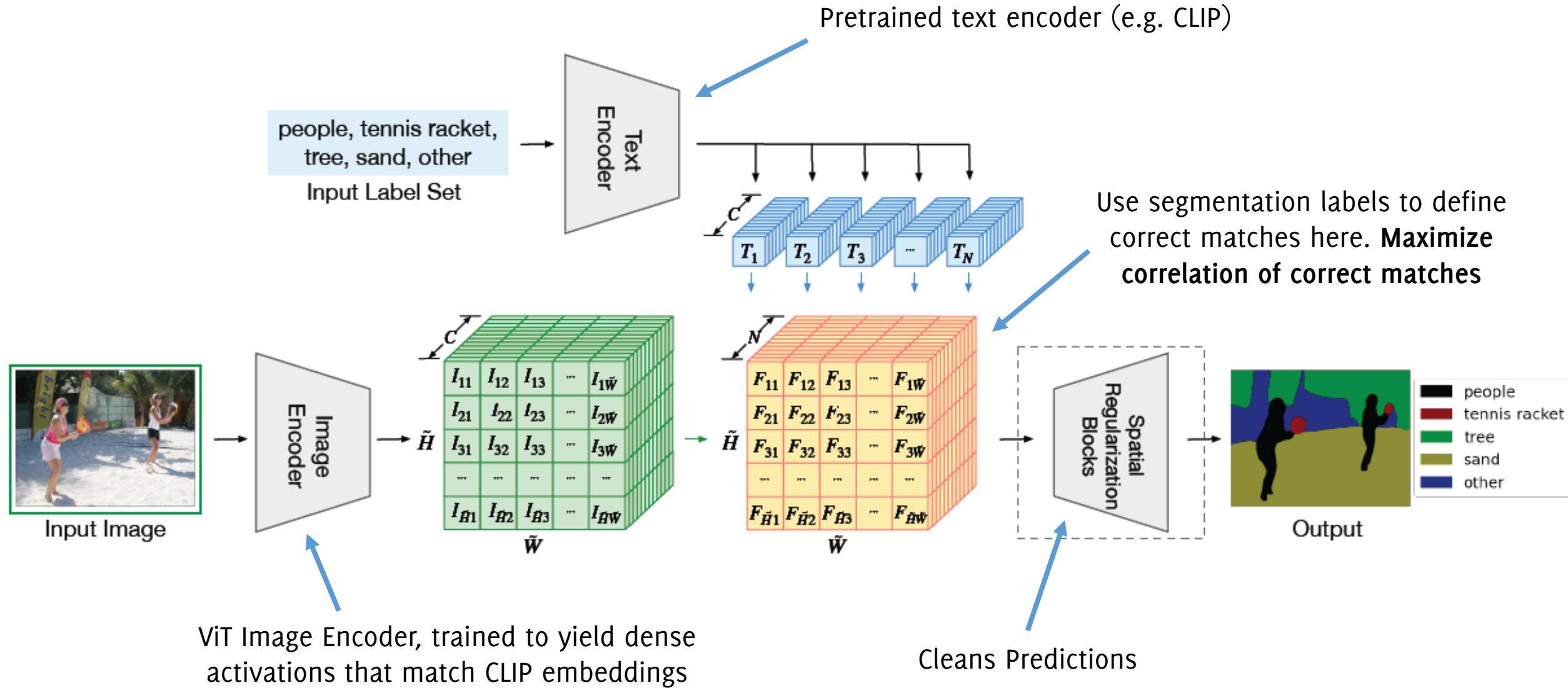
Vladlen Koltun

Apple

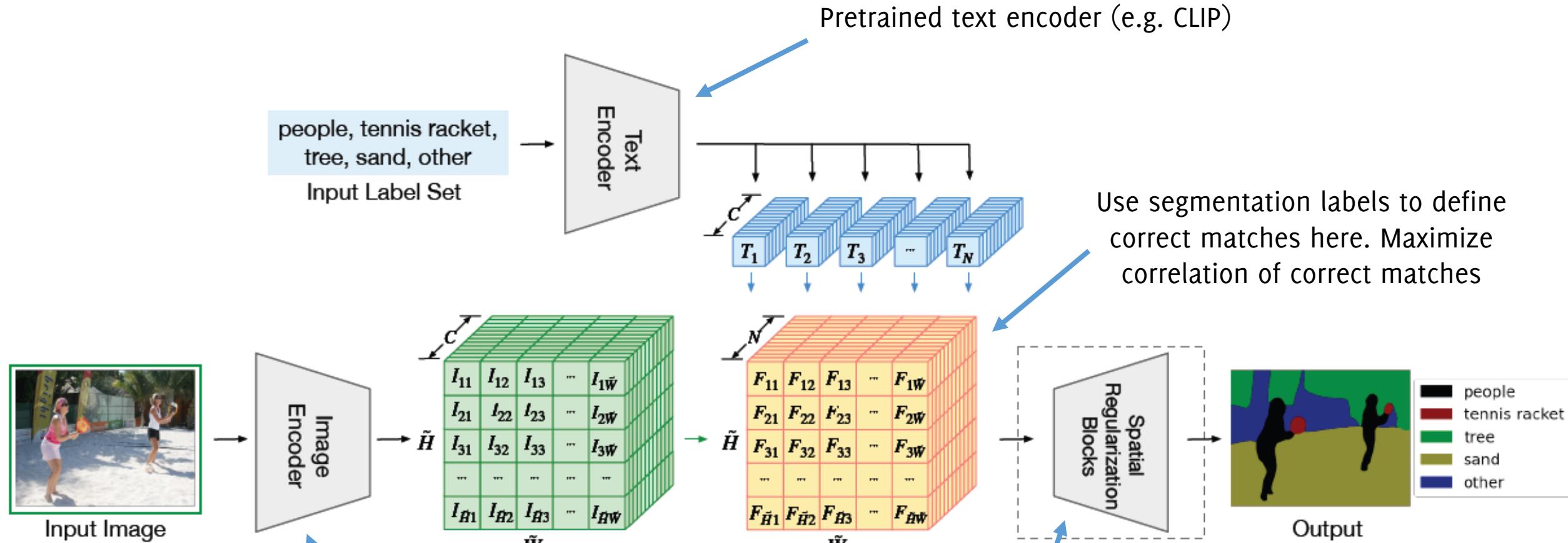
René Ranftl

Intel Labs

LSeg: Language Driven Semantic Segmentation



LSeg: Language Driven Semantic Segmentation

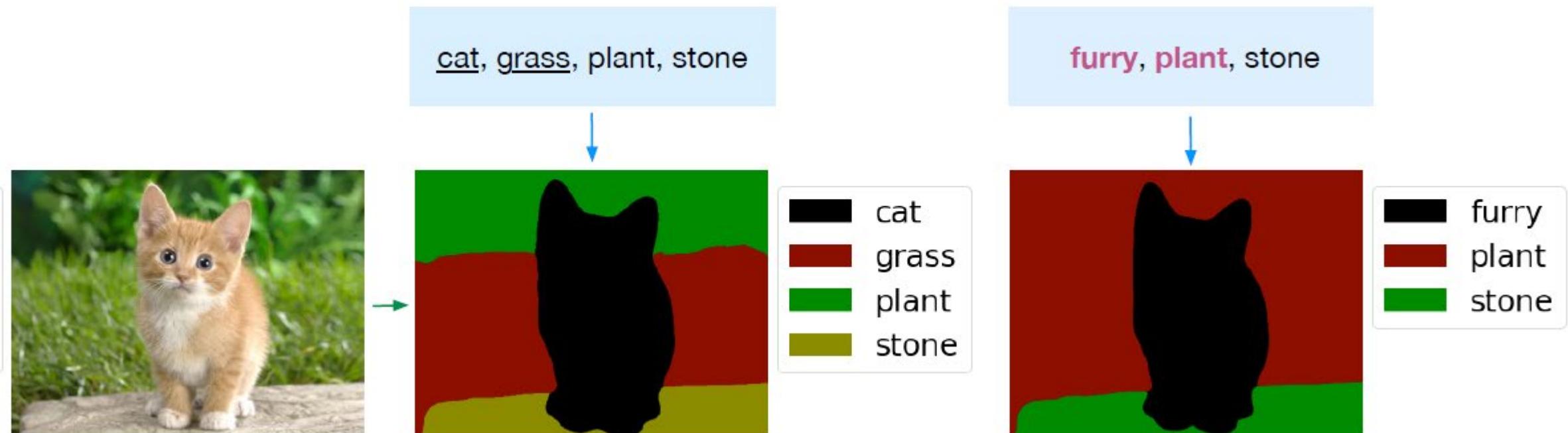


This model is trained on few labels for semantic segmentation. However, since **CLIP text embedding brings words referring to the same visual concepts nearby**, we can perform zero-shot learning in semantic segmentation

Zero-Shot Semantic Segmentation

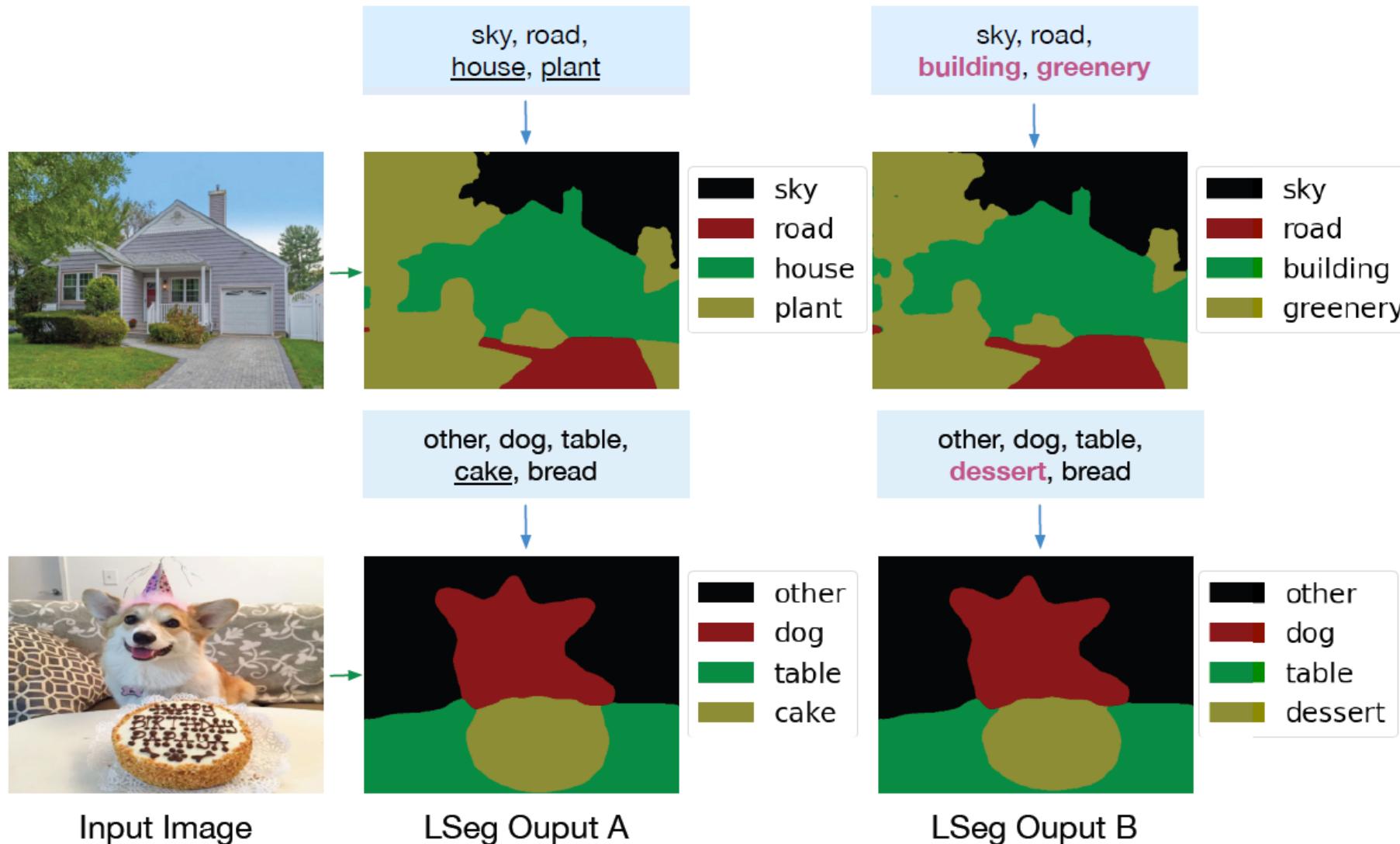
Red classes are added at test time (never annotated)

- You can arbitrarily expand, shrink, or reorder the label set for any image at test time
- No need of additional training samples



Zero-Shot Semantic Segmentation

Red classes are added at test time (never annotated)



Hierarchical Text-Conditional Image Generation with CLIP Latents

Aditya Ramesh*

OpenAI

aramesh@openai.com

Prafulla Dhariwal*

OpenAI

prafulla@openai.com

Alex Nichol*

OpenAI

alex@openai.com

Casey Chu*

OpenAI

casey@openai.com

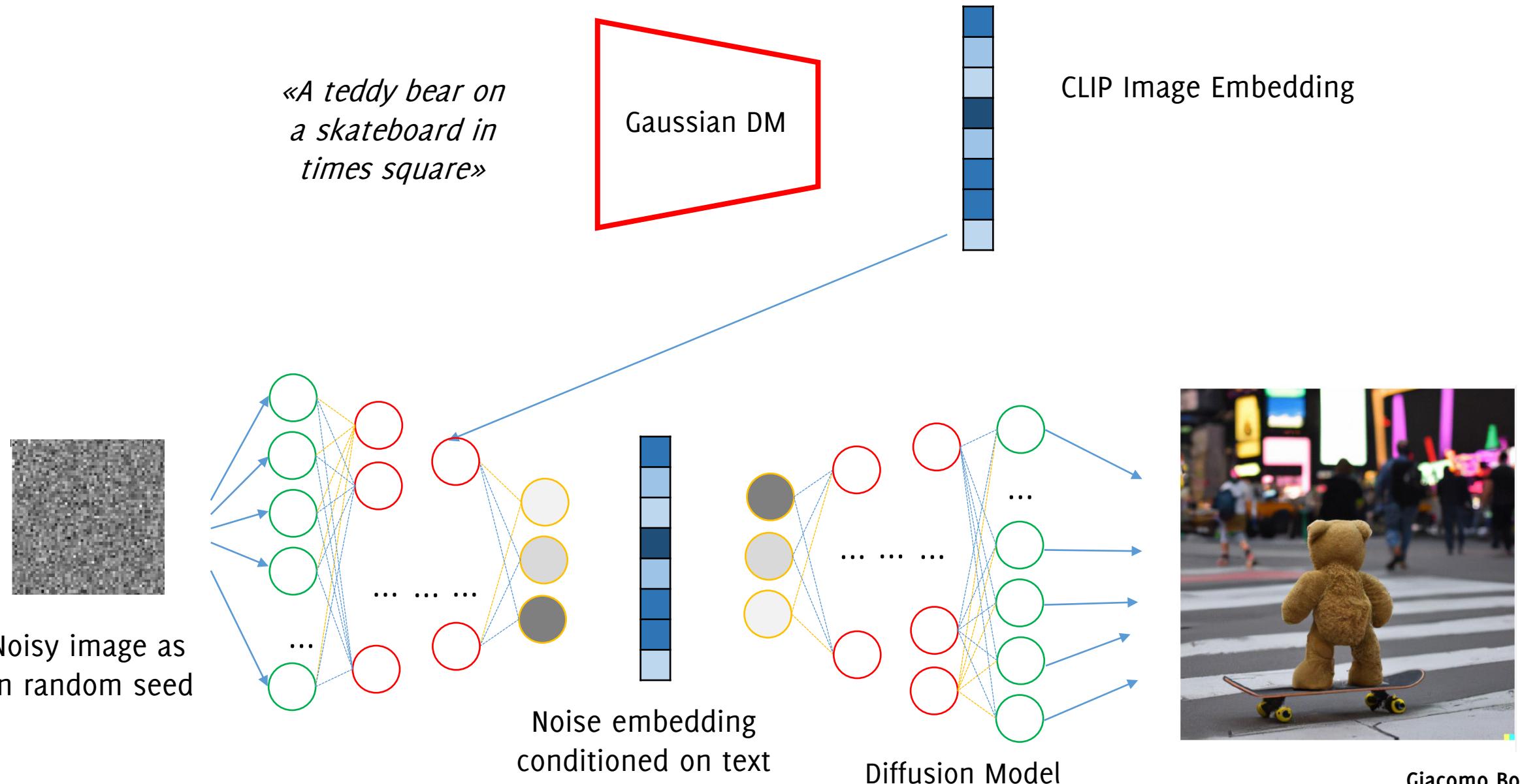
Mark Chen

OpenAI

mark@openai.com

Dall-e 2 overview

Text → CLIP Image Embedding





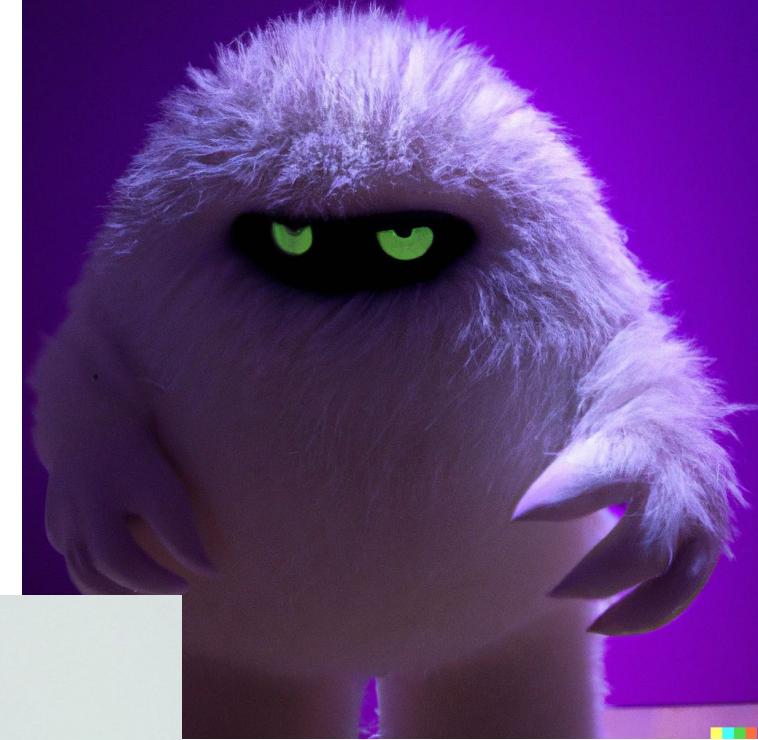
A hand drawn sketch of a Porsche 911



A van Gogh style painting of an American football player



A handpalm with a tree growing on top of it



A photo of a white fur monster standing in a purple room

Image Generation

- Image generation was considered the «holy grail» of imaging research up to less than 10 years ago
- Different architectures of neural networks made this possible.
- Still, the **practical impact** of generative models was **kind of limited**.
- Text embedding and superior quality in image generation has lead to astonishing performance, opening **new perspective applications**
- Behind these models there is no black magic or «human-like feelings», but rather expert training from **a huge amount of data...** it is important to know how these work!

Concluding Remarks

ConvNeXt



This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.

Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

A ConvNet for the 2020s

Zhuang Liu^{1,2*} Hanzi Mao¹ Chao-Yuan Wu¹ Christoph Feichtenhofer¹ Trevor Darrell² Saining Xie^{1†}

¹Facebook AI Research (FAIR) ²UC Berkeley

Code: <https://github.com/facebookresearch/ConvNeXt>

Bring Imaging-based Inductive Bias Back!

... many of the advancements of Transformers for computer vision have been aimed at bringing back convolutions. These attempts, however, come at a cost: a naive implementation of sliding window selfattention can be expensive [...] or more sophisticated in design.

On the other hand, it is almost ironic that a ConvNet already satisfies many of those desired properties, albeit in a straightforward, no-frills way.

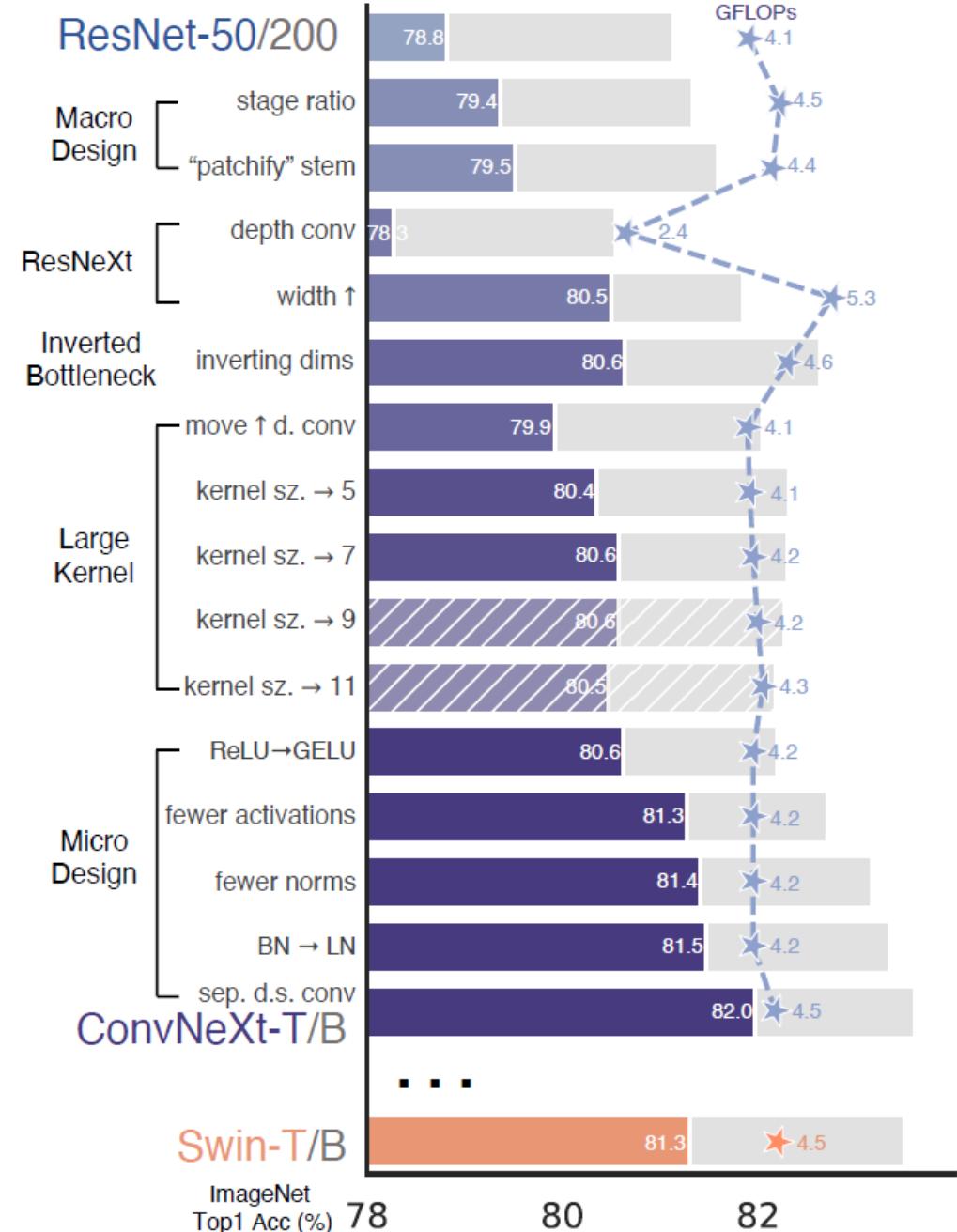
[..]

Our research is intended to bridge the gap between the pre-ViT and post-ViT eras for ConvNets, as well as to test the limits of what a pure ConvNet can achieve.

ConvNeXt

To do this, we start with a standard ResNet (e.g. ResNet-50) trained with an improved procedure. We gradually “modernize” the architecture to the construction of a hierarchical vision Transformer (e.g. Swin-T).

As a result, we propose a family of pure ConvNets dubbed ConvNeXt



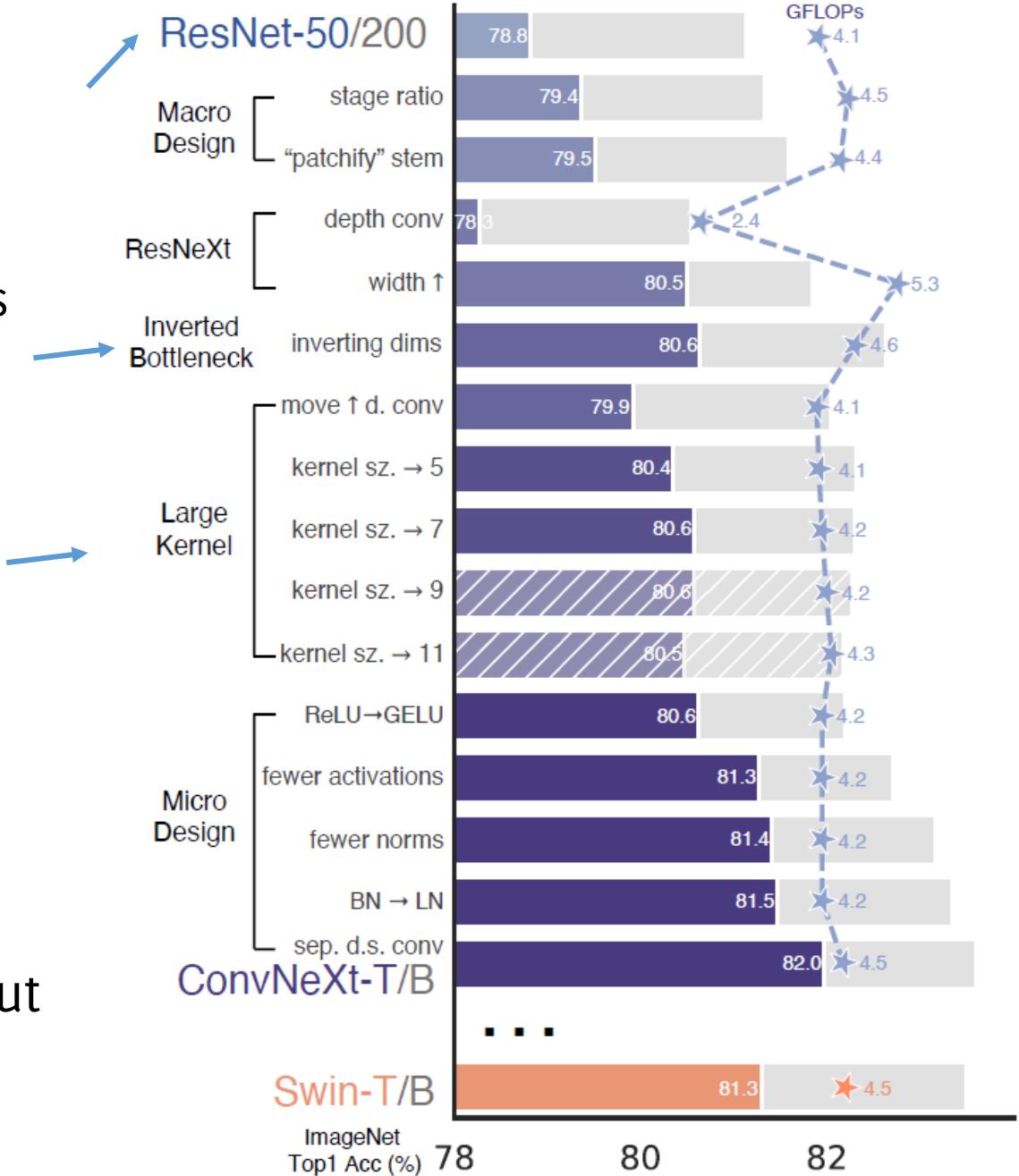
ConvNeXt

AdamW optimizer (used in Transformers) and modern data augmentation

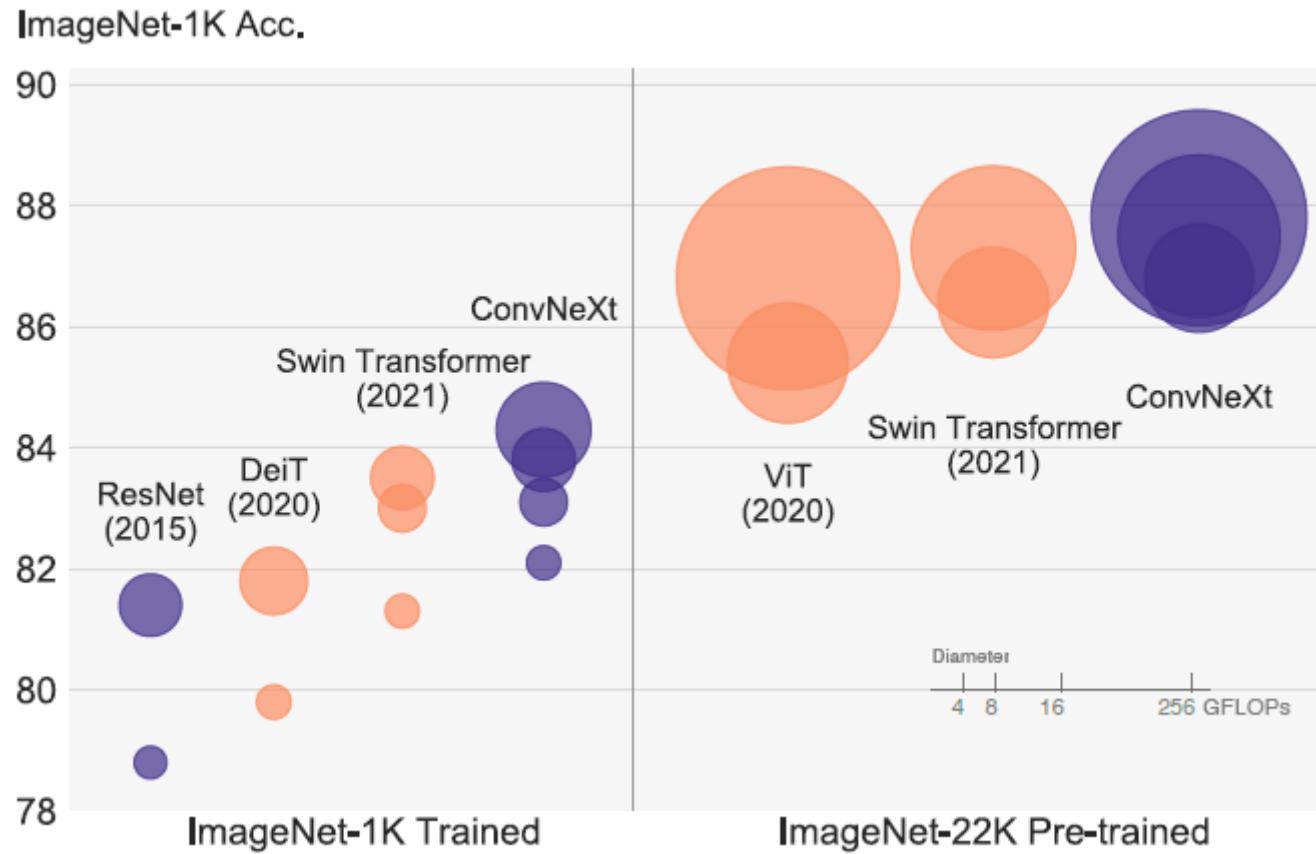
Expand the latent dimension among residual connections (separable convolutions as in MobileNet)

Gold Standard in CNN is to use 3×3 kernel size to get efficient processing and more nonlinearities (from VGG)
Hierarchical transformers use larger windows

All these variants are inspired from Transformers but uses methods that were already present in CNN.
Simply, these were never collectively used



ConvNeXt



Concluding Remarks

- The Advent of transformers represent a breakthrough in visual recognition
- It is surprising that they do not rely on “inductive biases” typical of images (and quite depressing from a computer-vision perspective). Huge training set seems enough
- Still, they seem to revert to implement “principles of vision” to solve advanced visual recognitition tasks
- They motivated a restyling of CNN that achived surprising results: ConvNeXt (nice to see inductive biases are back!)

- Transformers and Vision models (either CNNs or ViT) opens for new surprising applications combining text and vision