Convolutional Neural Networks

Cognitive Robotics

Marco Ciccone

Dipartimento di Informatica Elettronica e Bioingegneria
Politecnico di Milano

Computer Vision tasks

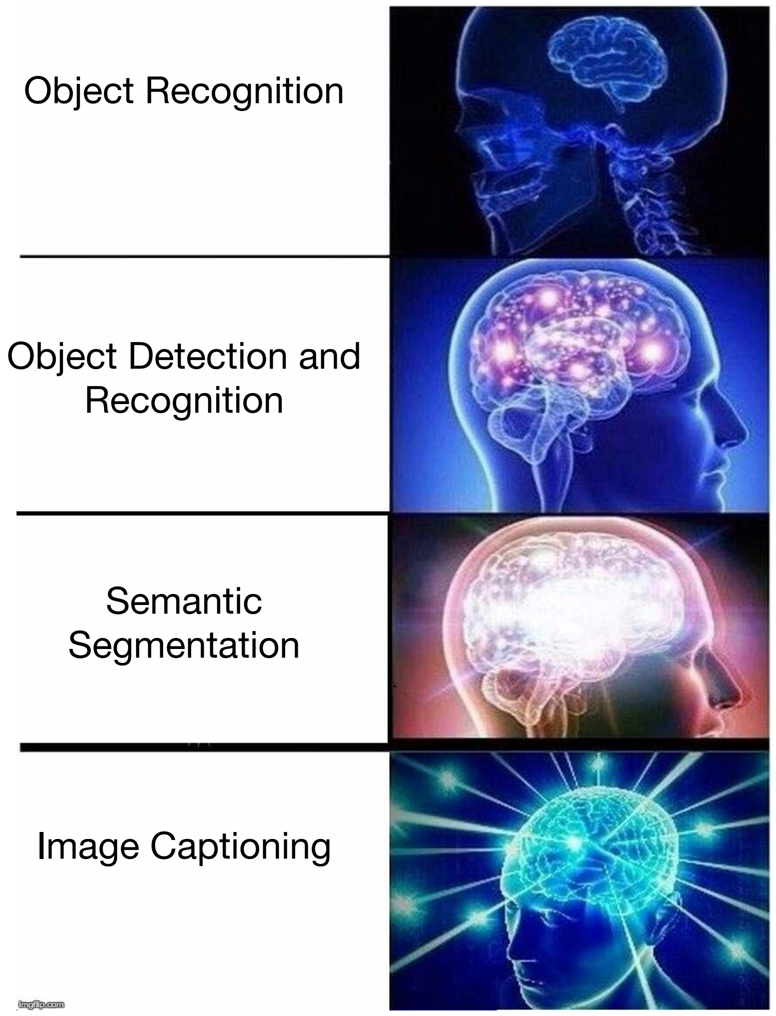
Computer Vision goal:

Processing visual data to solve a given task

Each problem has its own complexity depending on the level of details that we want to achieve in the visual understanding task

For simplicity in this course we will only focus on **Object Recognition**:

- given an input image identify which object it contains



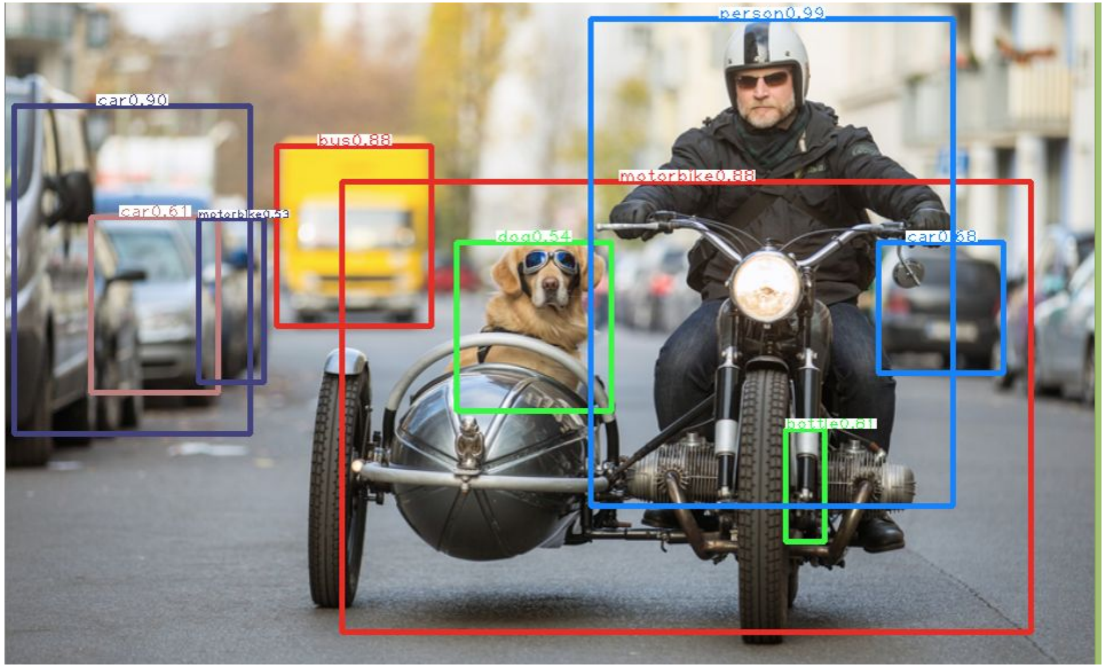
Object Recognition

Pug

Cat

Object Detection

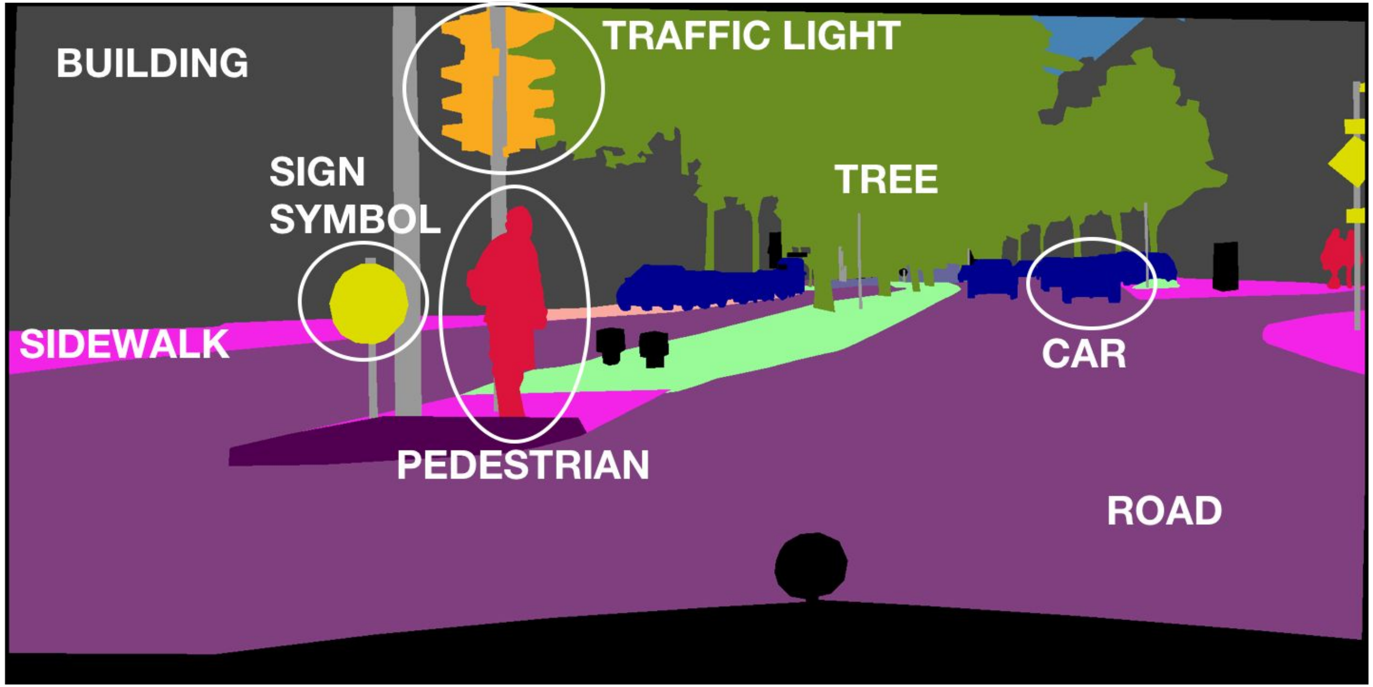
Determine the bounding box around an object and classify it



Semantic Segmentation

Assign to each pixel of the image a semantic class

From Cityscapes Dataset

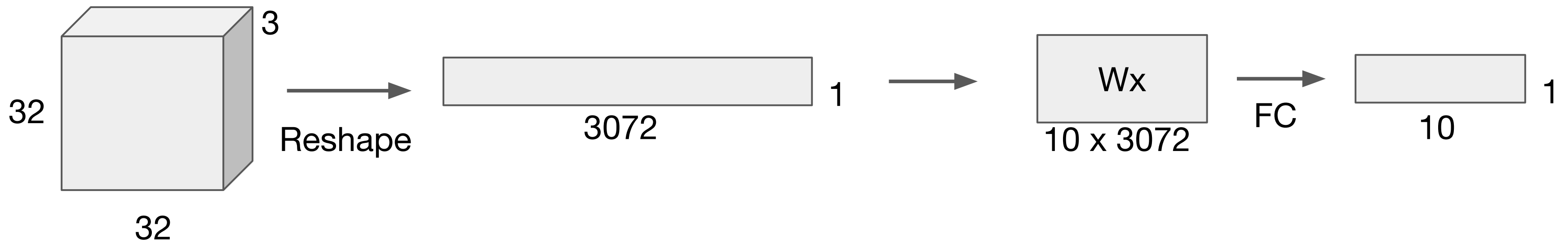


Fully Connected Layer

Using the tools that we already know…

Given a small image 32 x 32 x 3:

- we need to reshape the input features in order to apply a FC Layer
- 32 x 32 x 3 => 3072 x 1



Fully Connected Layer

Problems:

- Cannot deal with high dimensional inputs
    - e.g. 224 x 224 x 3 => 150528 inputs
    - Too many parameters
        - Risk of overfitting
        - High computational time
- No spatial structure
    - Reshaping the image we disregard the spatial informations
    - We want to exploit the 2D topology of the pixels
- We need invariance to
    - Translation
    - Illumination
    - …

We need a new type of network:
Convolutional Neural Networks (CNNs)

Convolutional Layer

We can design neural networks that are specifically well suited for computer vision:

- Can deal with very high dimensional inputs
- Take into account spatial informations
- Can build invariance to certain transformations

Convolutional Networks leverage these ideas through 3 properties:

- *Local Connectivity*
- *Parameter Sharing*
- *Pooling / Subsampling operations*

Local connectivity

- Traditional FC layers use a matrix of multiplication to describe the interaction between each input and output unit

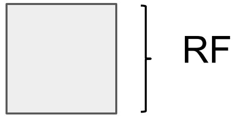
  => each input unit interacts with each output unit

  <u>But pixels in an image are only correlated with their closest pixels!</u>

- Convolutional networks (CNNs) have **“local (sparse) interactions”**
  - Each hidden unit is connected only to a subregion (patch) of the input
  - We define local interactions using **“kernels”** much smaller than the input
  - An hidden unit will be connected to all channels
    - 1 if grayscale image
    - 3 if RGB color image

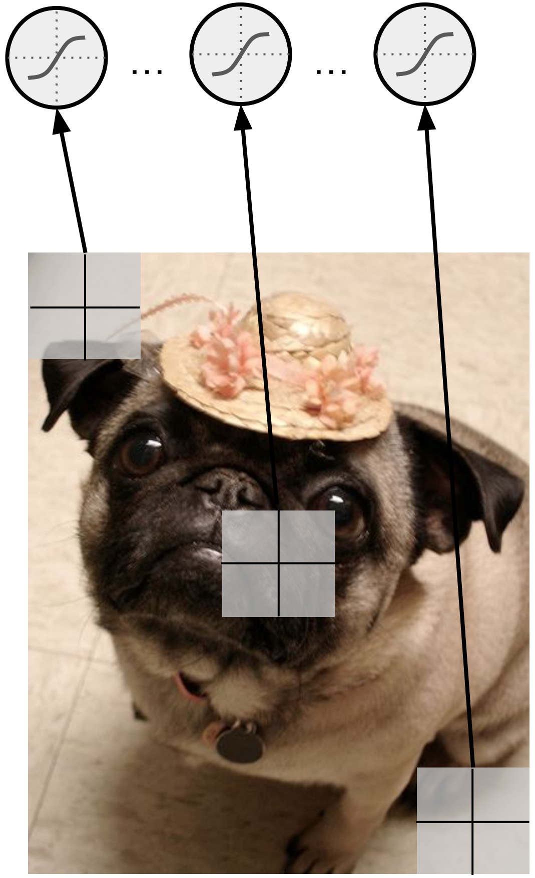
Local Connectivity

- **Receptive Field (RF):** portion of the input that affects the output



This solves two problems:

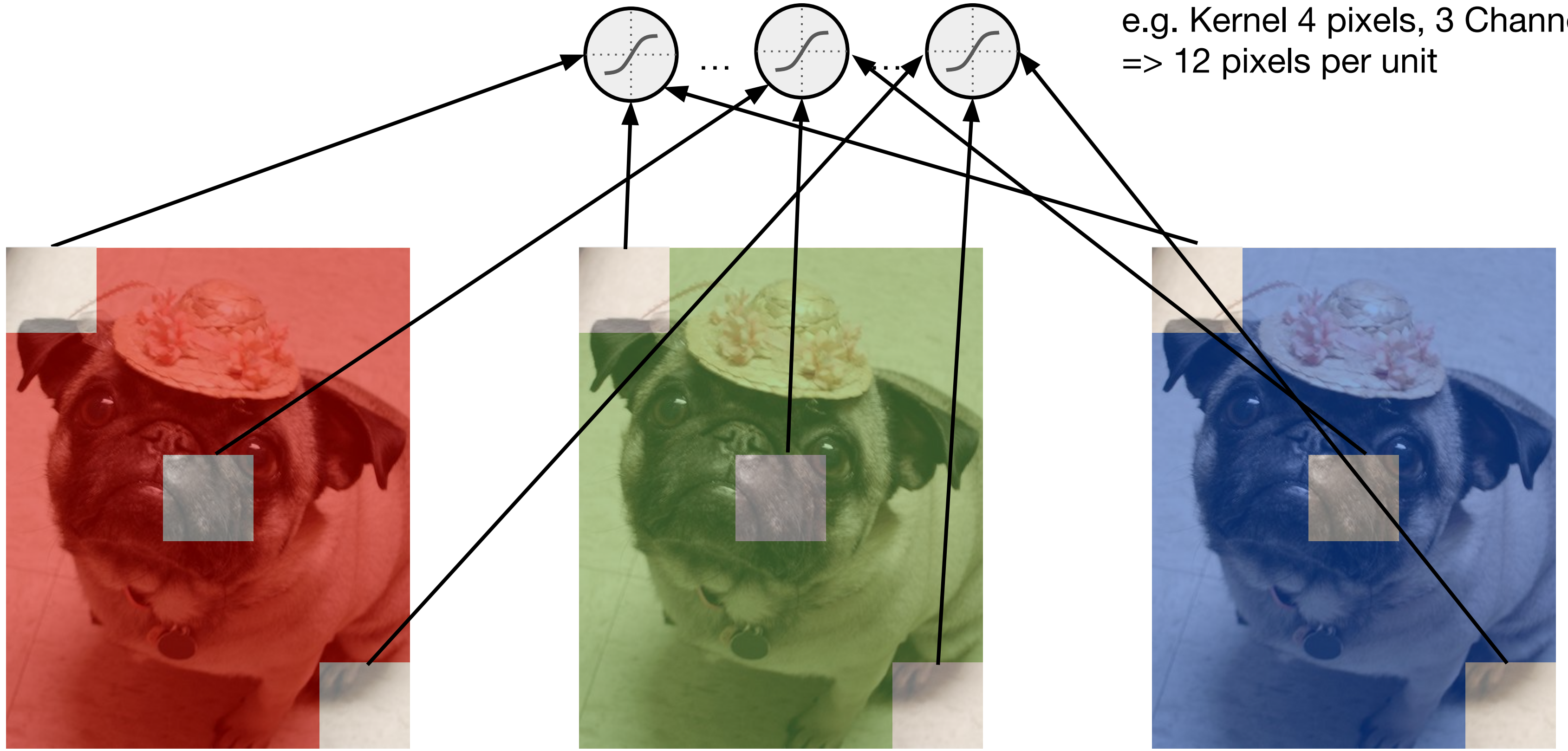
- FC layers would have an unmanageable number of parameters
  - Overfitting
  - Memory
- Computing the linear pre-activation of the hidden units it’s less expensive because we have reduced the number of connections (and so multiplications)



Local Connectivity

Each hidden unit is connected to a subregion for all of the channels (RGB).

e.g. Kernel 4 pixels, 3 Channels
=> 12 pixels per unit



Parameter sharing

In a FC Layer each element of the weight matrix is used exactly once when computing the output.

In CNN, we add a further simplification in order to reduce the number of parameters:

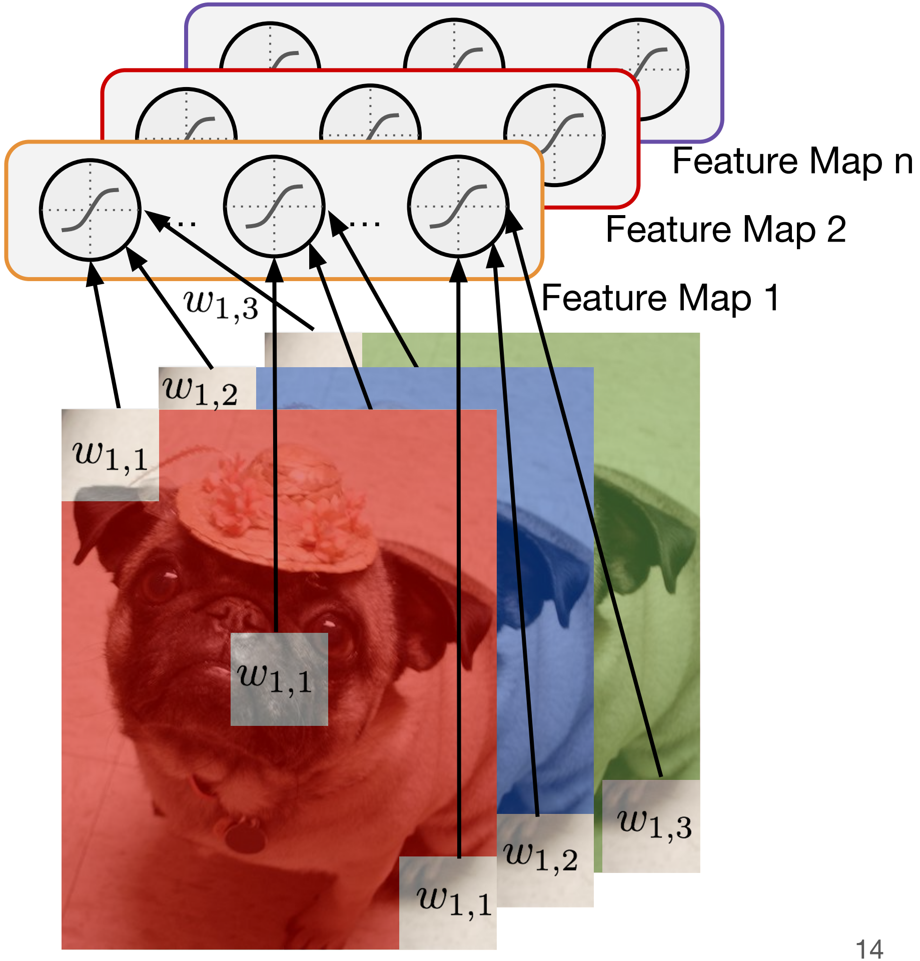
- Each weight of the kernel is used at *different positions of the input*
- Rather that learning a separate set of parameters (kernel) for each location, we learn only one set.
- We say that the network has "*tied weights*" because the value of the weight applied to one input is tied to the value of a weight applied elsewhere.

Parameter Sharing

- The weights are shared over the location for each feature map

Note:

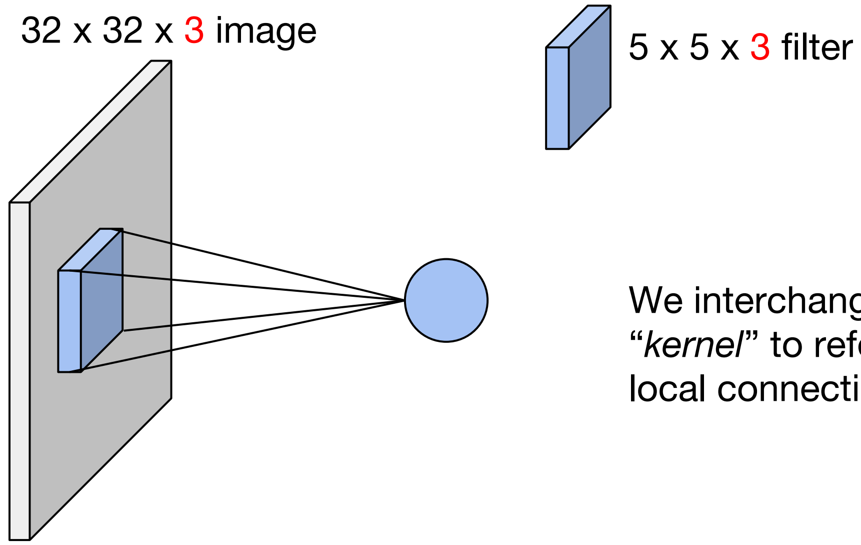
- For clarity, in the figure are shown only the parameters of one single feature map.
- Each feature map has different parameters $w_{i,j}$, where:
  - **i**: feature map, **j**: input channel



Local Connectivity + Parameter Sharing
=
Convolution operator

Parameter sharing allows to learn filters that are able to extract particular features independently by the position in the image

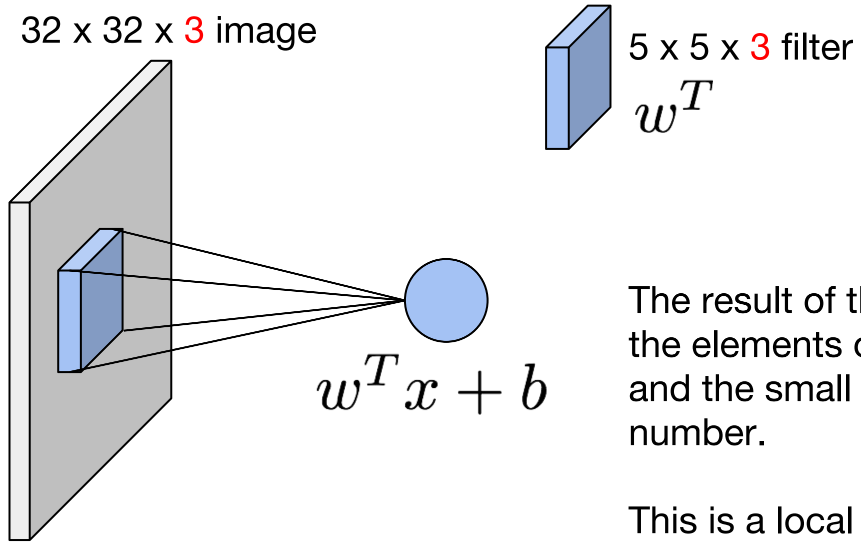
Convolutional Neural Network



NOTE: Input and filters always match the number of channels

We interchange the terms “*filter*” and “*kernel*” to refer to the weights of the local connections

Convolutional Neural Network

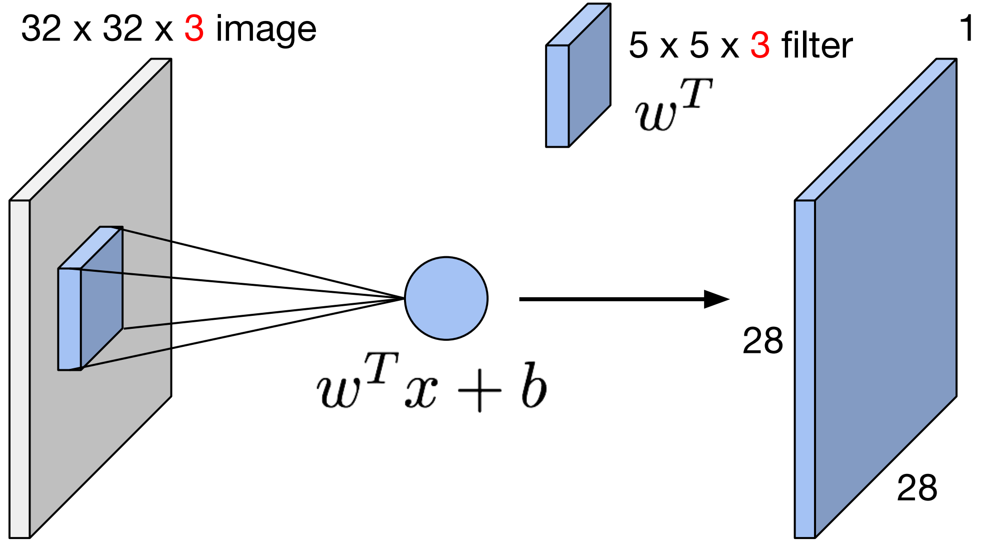


We **Convolve** the kernel with the image i.e. slide the filter over all the input locations and compute the dot product

The result of the dot product between the elements of the kernel (weights) and the small region of the input is one number.

This is a local linear combination of the input features.

Convolutional Neural Network

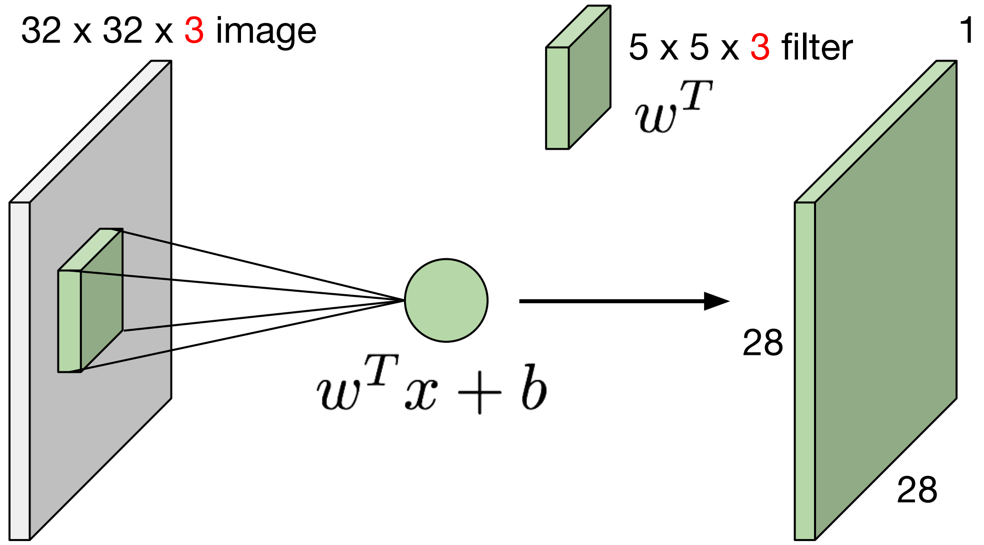


The result of the convolution between the input and the kernel is called **“feature map”** or **“activation map”**

Note:
Depending on the type of convolution the feature map will have a different size (see slides on convolution arithmetic):

- VALID (n - k + 1)
- SAME
- FULL (n + k -1)

Convolutional Neural Network

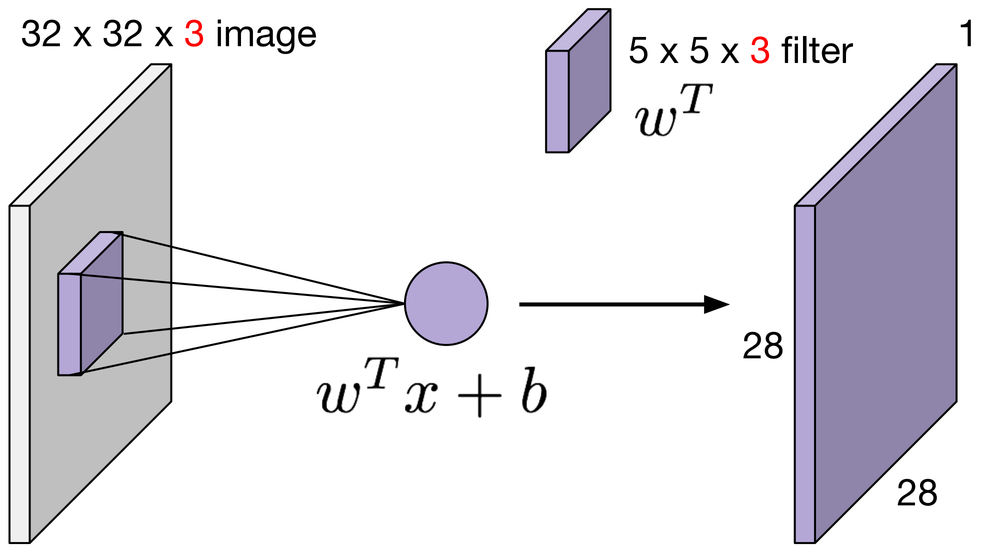


Each color corresponds to a different filter that:

- Shares its parameters among all the locations of the input
- And extract a different feature independently on the location in which is applied

The result of the convolution between the input and a kernel gives a **feature map.**

Convolutional Neural Network

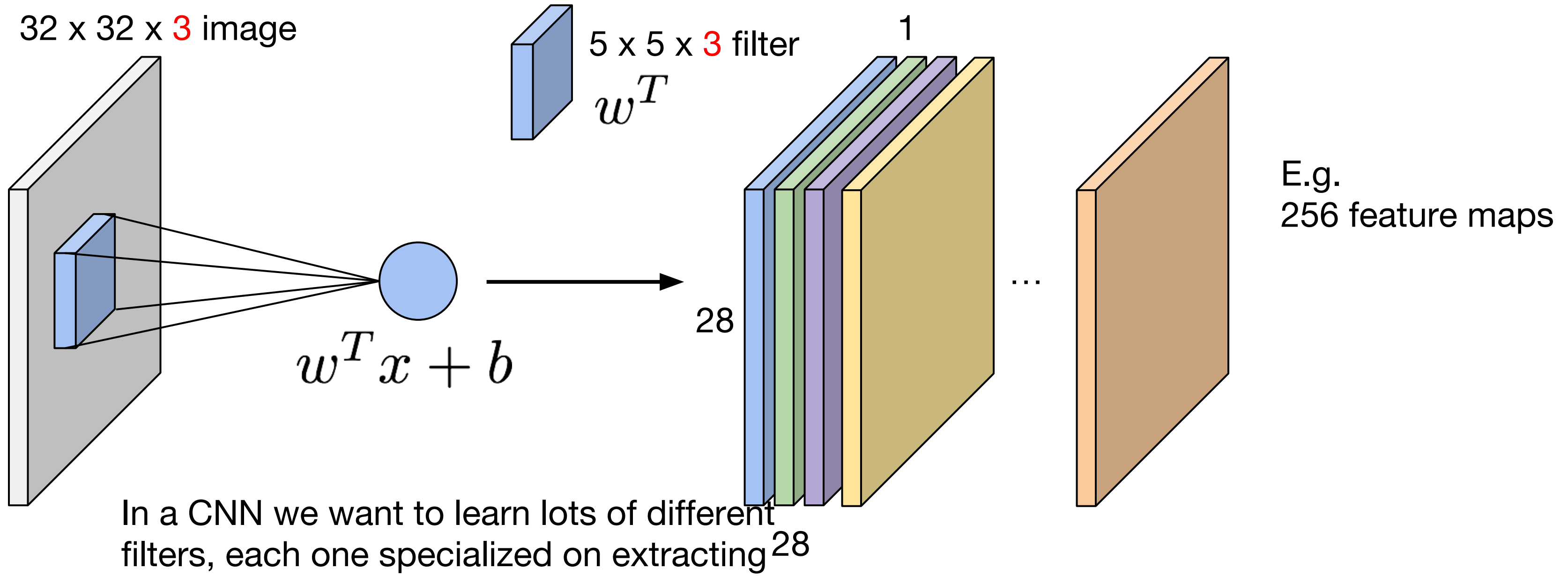


Each color corresponds to a different filter that:

- Shares its parameters among all the locations of the input
- And extract a different feature independently on the location in which is applied

The result of the convolution between the input and a kernel gives a **feature map.**

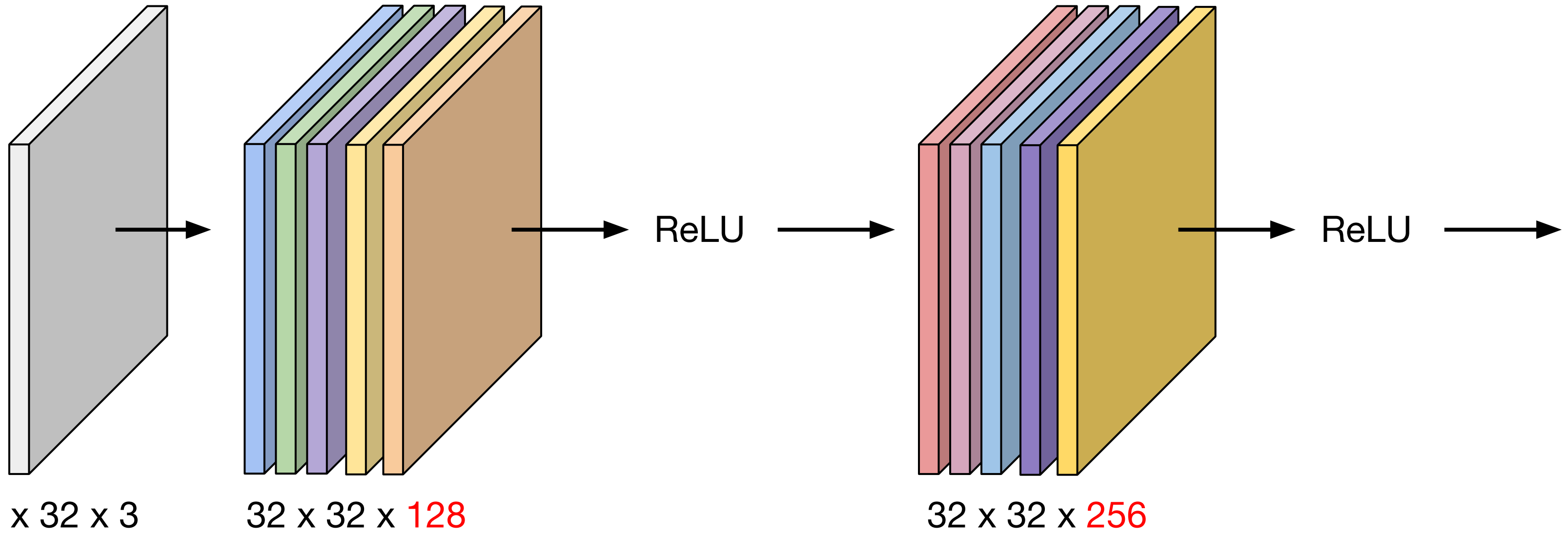
Convolutional Neural Network



In a CNN we want to learn lots of different filters, each one specialized on extracting a specific feature in the image.

So we are going to have several feature maps!

Convolutional Network



Remark:
CNNs learn a nonlinear mapping between input images and classes adding constraints on the model parameters

Number of parameters

Example:

Input volume: 32 x 32 x 3

10 filters 5x5

Number of parameters in this layer?

Number of parameters

Example:

Input volume: 32 x 32 x 3

10 filters 5x5

Number of parameters in this layer?

each filter has 5*5*3 + 1 = **76 params** => 76*10 = **760** (+1 because of the bias)

Number of parameters

Example:

Input volume: 32 x 32 x 3

10 filters 5x5

Number of parameters in this layer?

each filter has 5*5*3 + 1 = **76 params** => 76*10 = **760** (+1 because of the bias)

NOTE:

FC Layer would have 32*32*3*10 = **30720 parameters!** (because we have a matrix)

Pooling - Subsampling

Idea: take a set of hidden units and aggregate their activations

- Operates over each activation map independently
- Makes the representations smaller and more manageable
    - Reduce the dimension of the input to the next layer
- It can be any aggregation function (AVERAGE, MAX)
- It is performed over *non-overlapping patches* (neighbors)
- Pooling is a form of nonlinear downsampling

Convolutional neural network often alternates convolutional and pooling layers

Max Pooling

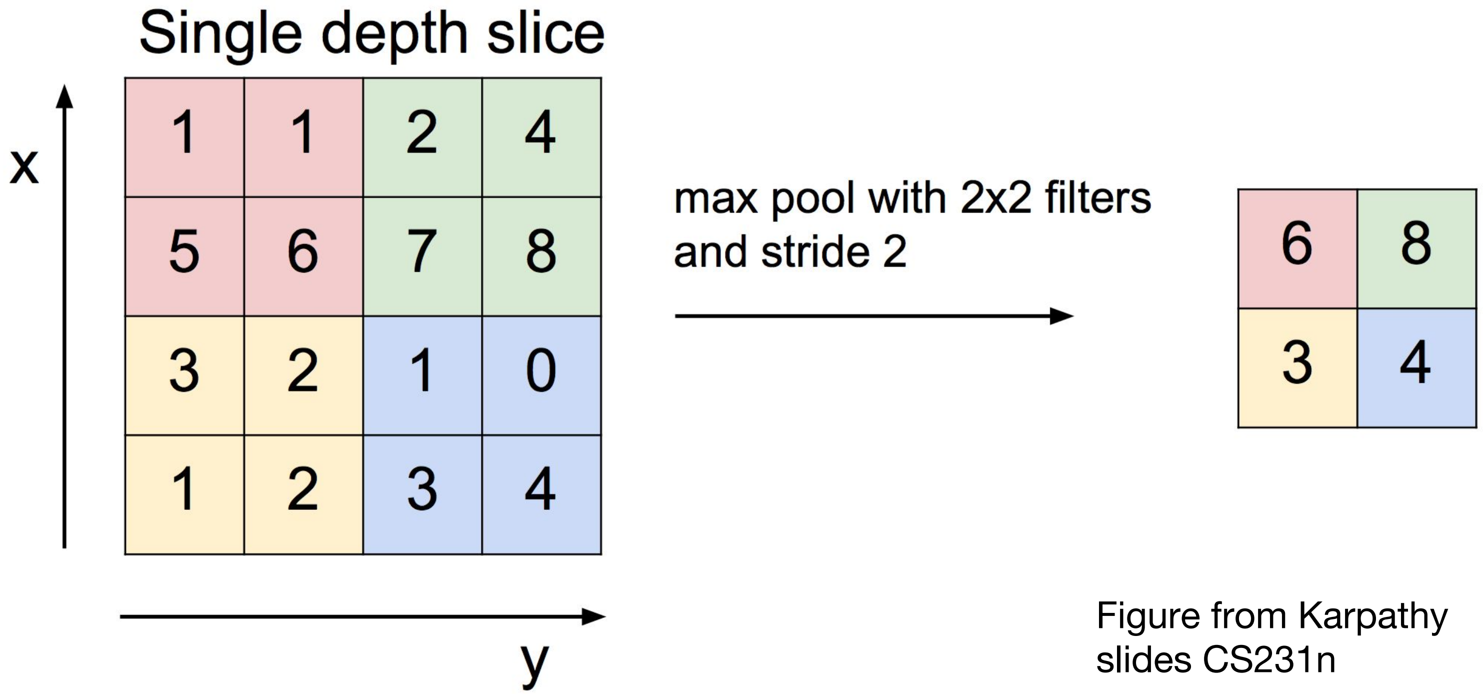
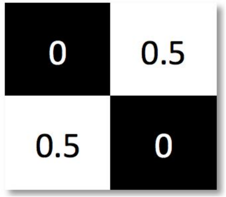


Figure from Karpathy slides CS231n

Local Translation Invariance



Two slightly translated images gives the same feature map after max pooling!

They are independent on the position (given a small translation)

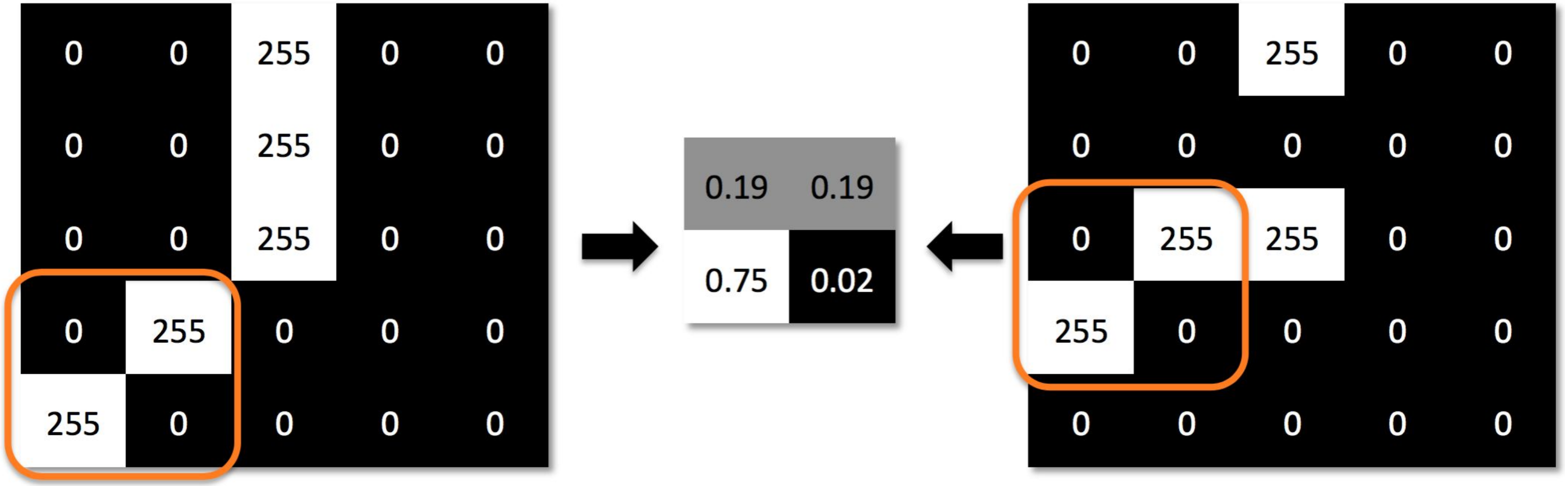
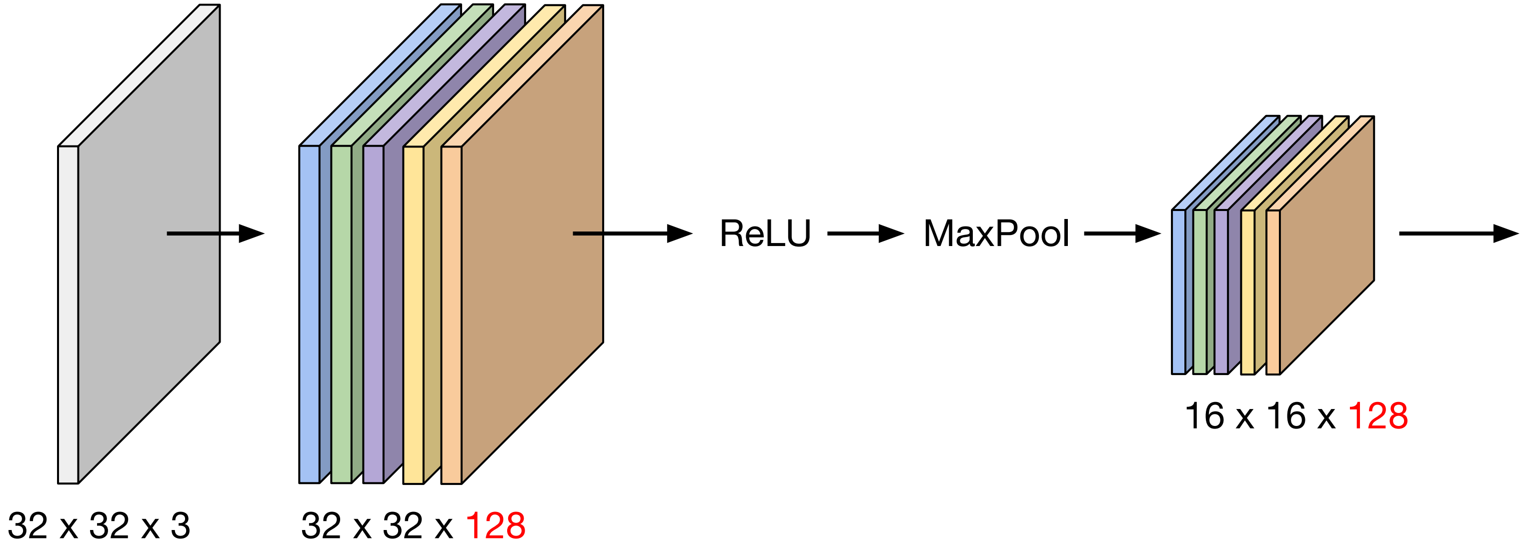


Figure from Hugo Larochelle slides

Convolution + Pooling



Spatial Prior

We can think of convolution as introducing an *"infinitely strong prior"* probability distribution over the parameters of the layer.

The prior says that the functions the layers should learn contains only *local interactions* and it's ***equivariant to translation.***

The use of convolution constraints the class of functions that the layer can represent, meaning that we are putting zero probability on some type of parameters.

Transformation invariance

- Invariances built-in in convolutional network:
    - small translations: due to convolution and max pooling

- It is **NOT** invariant to other important variations such as *rotations* and *scale*

- However, it's easy to artificially generate data with such transformations
    - could use such data as additional training data
    - neural network **will learn to be invariant to such transformations**

Data Augmentation

Augment the dataset with several transformation
It can be done online at every minibatch to add robustness to the network

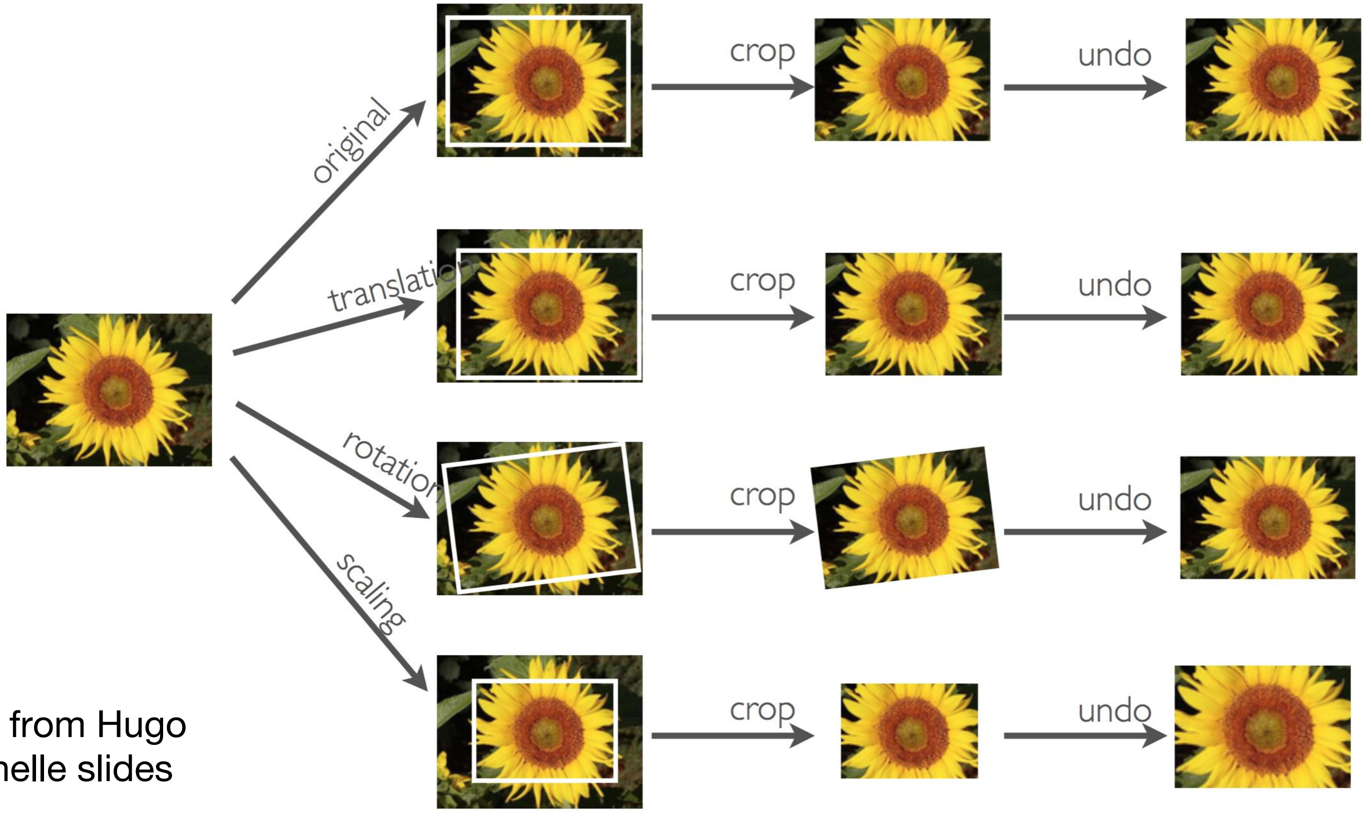


Figure from Hugo Larochelle slides

Training is performed through Backpropagation!

We just need to know how to compute the gradient of:

- Convolution
- Pooling

Optional: Backpropagation in CNN

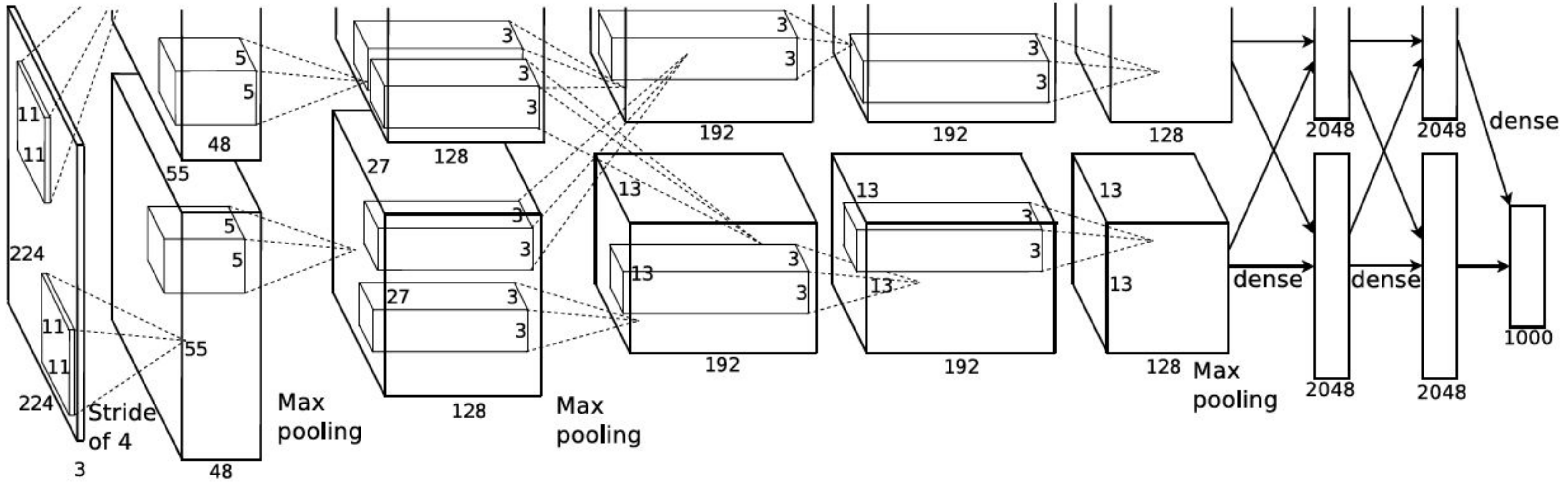
CNNs Architectures

See this article for a detailed review of different CNNs architectures and more

AlexNet (2012)

First Deep CNN actually working

- 60 Millions of parameters
- Much of them are in the FC layers (classifier)
- Use of Dropout
- Use of RELU
- Minibatch Gradient Descent with Momentum
- Data Augmentation
- Trained for about a week on NVIDIA GTX 580

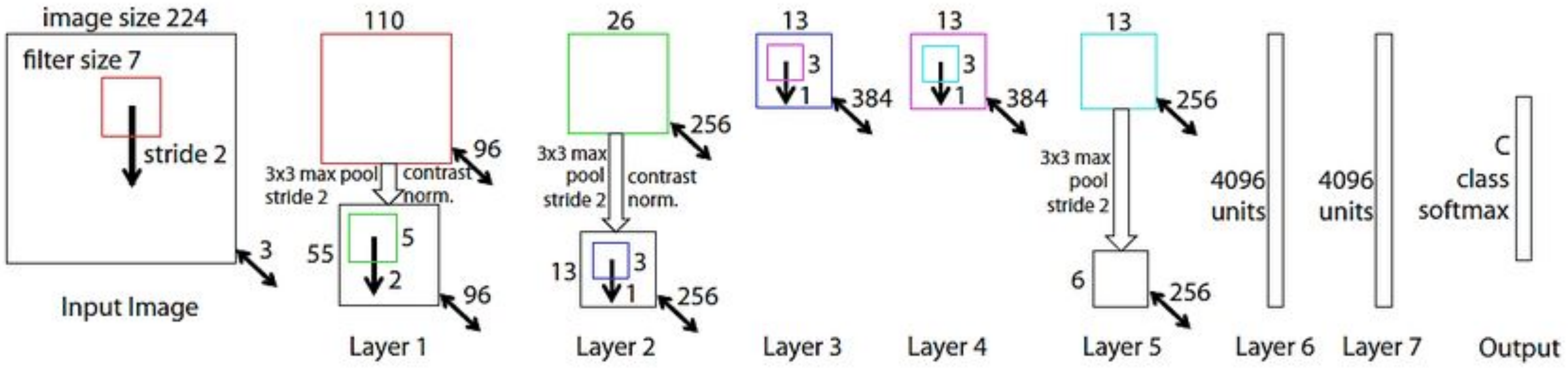


AlexNet architecture may look weird because there are two different “streams”. This is because the training process was so computationally expensive that they had to split the training onto 2 GPUs.

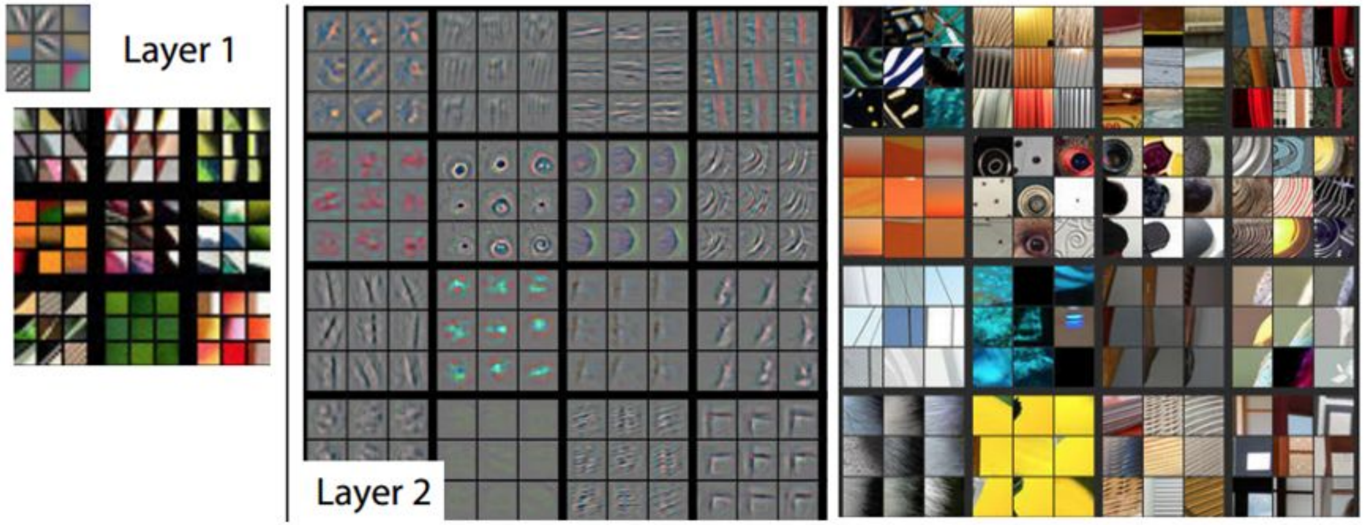
ZF-Net (2013)

Improvement of AlexNet with accurate tuning of the hyperparameters

In the same paper Zeiler and Fergus present a method to visualize the filter learned by the Network (*DeconvNet*).

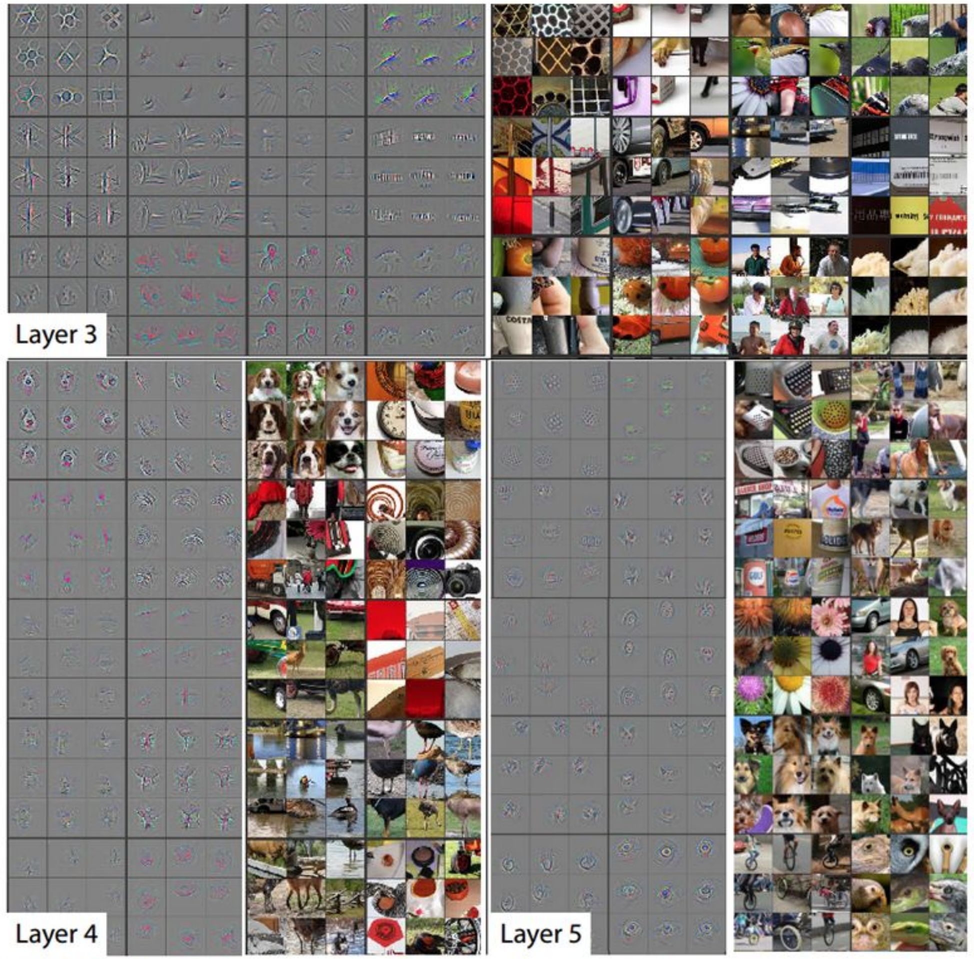


DeconvNet



Visualizations of Layer 1 and 2. Each layer illustrates 2 pictures, one which shows the filters themselves and one that shows what part of the image are most strongly activated by the given filter. For example, in the space labled Layer 2, we have representations of the 16 different filters (on the left)

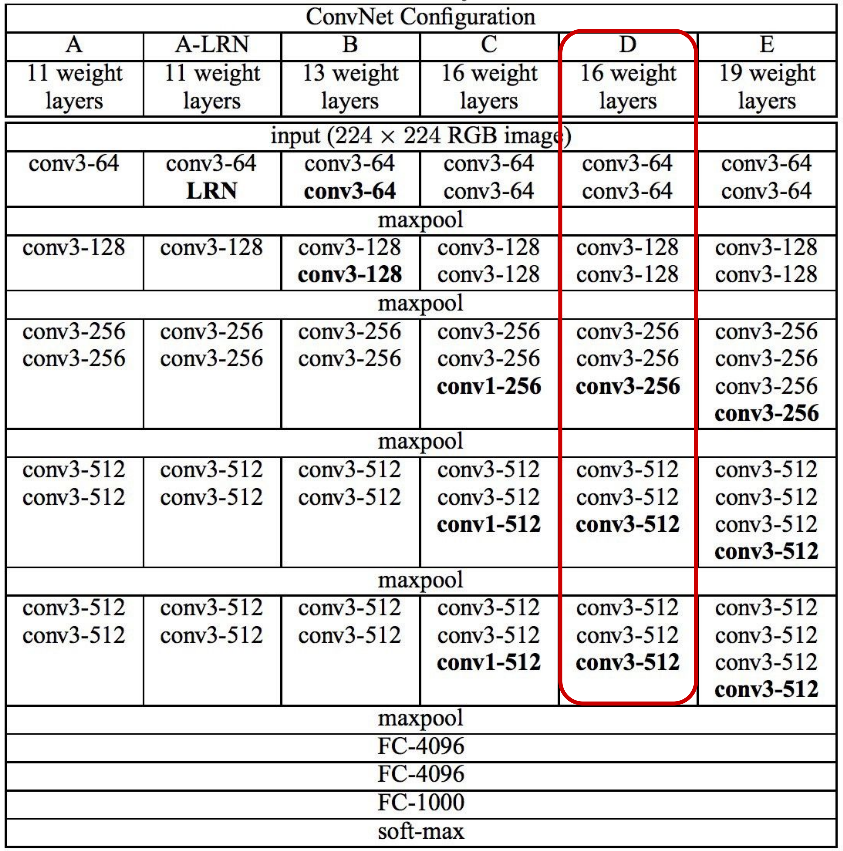
DeconvNet



Visualizations of Layers 3, 4, and 5

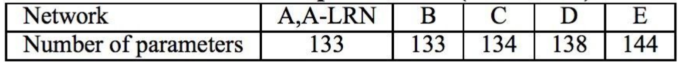
VGG-16 (2014)

- Use 3x3 kernels
- Block of stacked 3 conv layers have an effective receptive field of 7x7 but less parameters (+3 nonlinearities)
- Today we have a nice efficient 3x3 convolution (Winograd)
- Lots of parameters **140 M** (Most of them are in the FC Layers)
- Extracts very general and transferable features
- FC Layers are redundant and can be removed (without loss of accuracy)
- The structure can be used for other tasks such as Semantic Segmentation



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 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

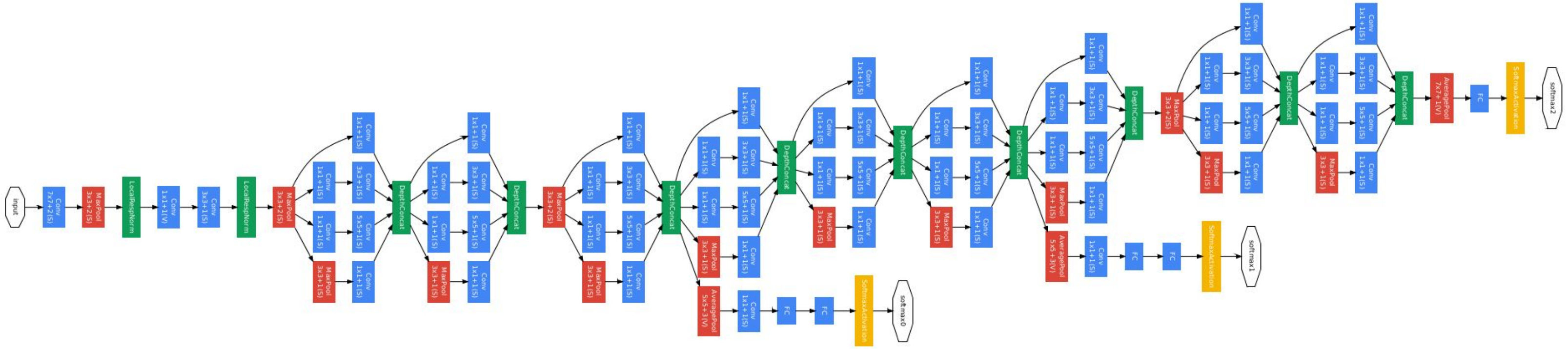
Table 2: **Number of parameters** (in millions).



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

GoogleNet (2014)

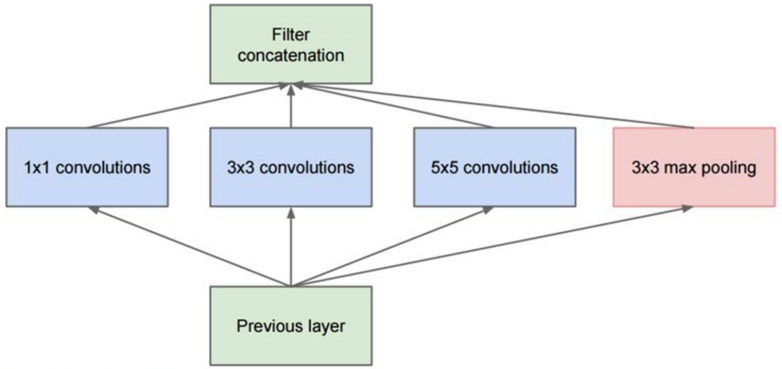
- Used 9 Inception modules in the whole architecture, with over 100 layers in total! Now that is deep…
- No use of fully connected layers! They use an average pool instead, to go from a 7x7x1024 volume to a 1x1x1024 volume. This saves a huge number of parameters.
- Uses 12x fewer parameters than AlexNet (4 Millions)
- At test time, multiple crops of the same image were created, fed into the network, and the softmax probabilities were averaged to give us the final solution.
- There are several updated versions to the Inception module (Versions 6 and 7).



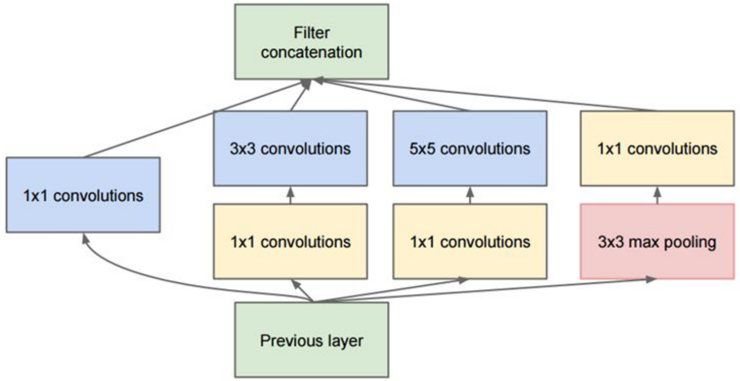
Inception module

GoogLeNet was one of the first models that introduced the idea that CNN layers didn’t always have to be stacked up sequentially.

Coming up with the Inception module, the authors showed that a creative structuring of layers can lead to improved performance and computationally efficiency.



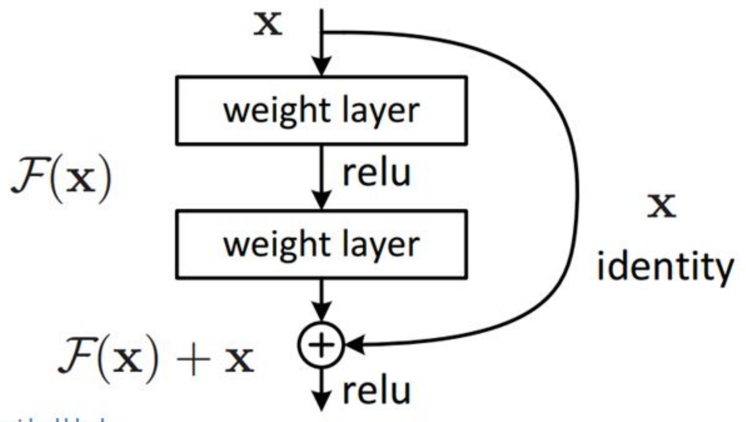
Naïve idea of an Inception module



Full Inception module

ResNet (2015)

- Add a skip connection from the input to the output of the block bypassing the layer.
- The authors claimed that is easier to optimize the residual mapping
- The gradient will easily flow through the graph because of the skip connection
- State of the art



A residual block

Acknowledgements

This slides are highly based on material taken from:

- Hugo Larochelle
- Andrej Karpathy
- Laurent Dinh