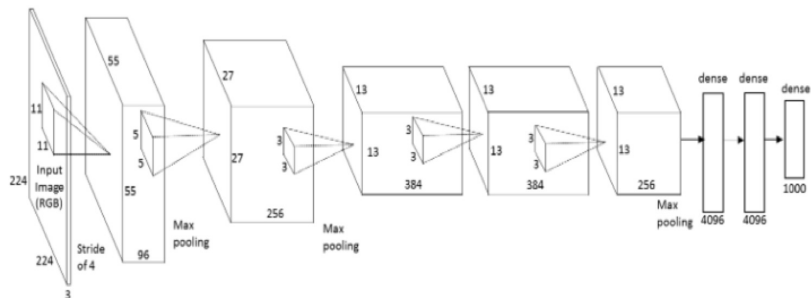


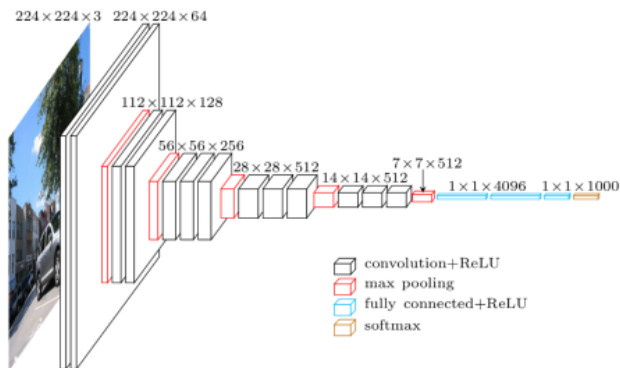
- Examples of real CNNs
- Transfer Learning



AlexNet Architecture (Krizhevsky, Sutskever e Hinton), winner of a NIPS contest in 2012.



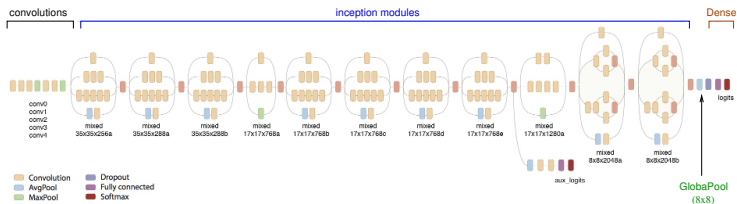
VGG 16 (Simonyan e Zisserman). 92.7 accuracy (top-5) in ImageNet (14 millions images, 1000 categories).



Picture by Davi Frossard: VGG in TensorFlow



Inception V3

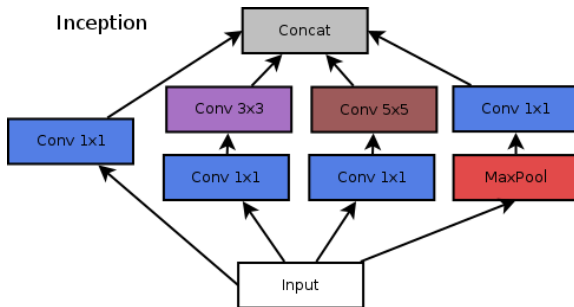


The convolutional part is a long composition of
inception modules



Inception modules

The networks is composed of inception modules (towers of nets):

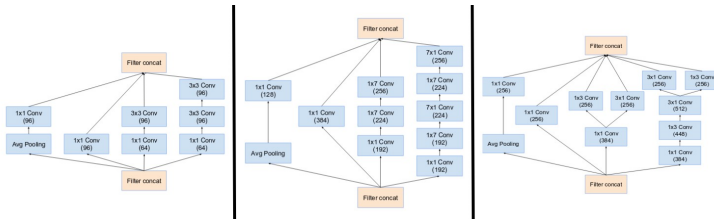


Video from the Udacity course "Deep Learning"

Variants

The point is to induce the net to learn different filters.

Many variants proposed and used over years:



The inception hypothesis

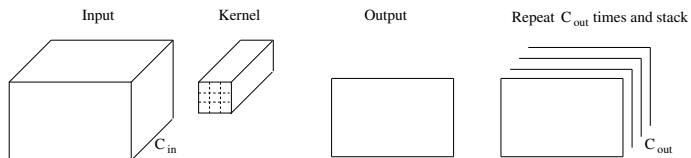
Remember that normal convolutional kernels are 3D, simultaneously mapping **cross-channel** correlations and **spatial** correlations.

It can be better to decouple them, independently looking for cross-channel correlations (via 1×1 convolutions), and spatial 2D convolutions.

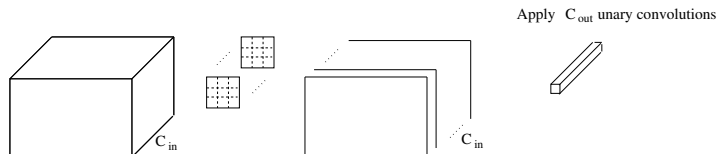
Inception modules can be understood as an intermediate step between a regular convolution and a depthwise separable convolution (a depthwise convolution followed by a pointwise convolution).

Depthwise separable convolutions

Traditional Convolutions



Depthwise Separable Convolutions



Depthwise separable convolutions

Suppose we have a convolutional layer with a 3×3 kernel, 16 input channels and 32 output channels. The input is convolved 32 times with different kernels of dimension $3 \times 3 \times 16 = 144$: we have a total of $32 \times 144 = 4608$ parameters.

In a **depthwise separable convolution** on the same example, we first traverse the 16 channels with a different 3×3 kernel, and then we apply 32 different kernels with dimension $1 \times 1 \times 16$. The total number of parameters is $16 \times 3 \times 3 + 32 \times 1 \times 1 \times 16 = 656$.

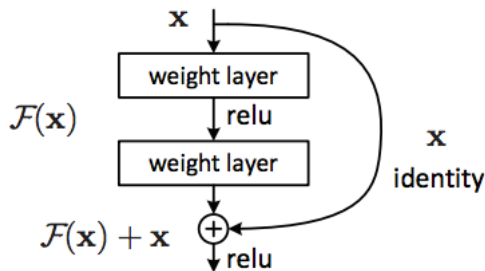


Depthwise separable convolutions have been made popular by their adoption in

- **Xception**
- **MobileNet** - a class of “light” models conceived to be deployed on mobile devices.

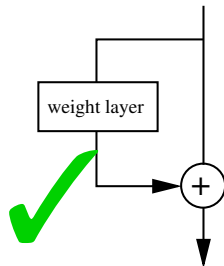
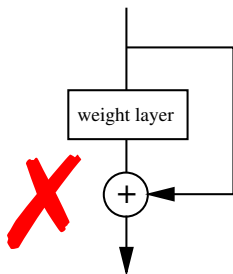
Residual Learning

Another recent topic is residual learning.

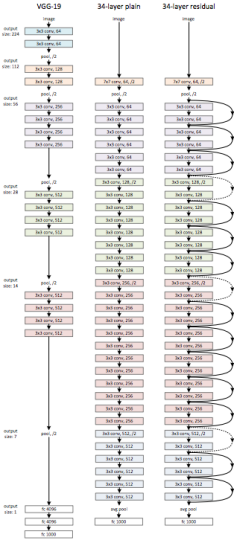


Instead of learning a function $\mathcal{F}(x)$ you try to learn $\mathcal{F}(x) + x$.

The right intuition



Residual networks



you add a residual shortcut connection every 2-3 layers

Inception Resnet is an example of a such an architecture



Why Residual Learning works?

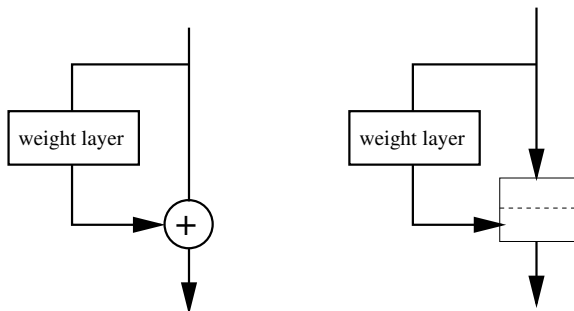
Not well understood yet.

The usual explanation is that during back propagation, **the gradient at higher layers can easily pass to lower layers**, without being mediated by the weight layers, which may cause vanishing gradient or exploding gradient problem.



Sum or concatenation?

The “sum” operation can be interpreted in a liberal way.
A common variant consists in concatenating instead of adding
(usually along the channel axis):



ConvNets essentially grow in three directions:

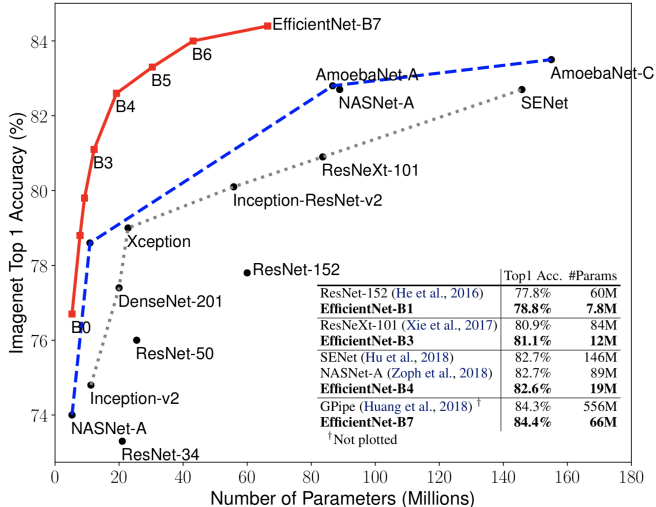
- **Layers:** the number of layers
- **Channels:** the number of channels for layers
- **Resolution:** the spatial width of layers

Is there a principled method to scale up ConvNets that can achieve better accuracy and efficiency?

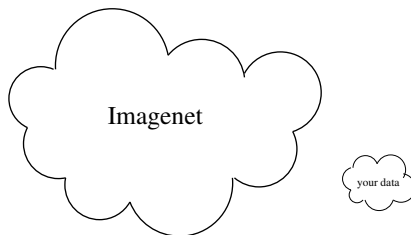
Question addressed in [EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks](#)



Efficient Net



Transfer Learning



Reusing Knowledge

We learned that the first layers of convolutional networks for computer vision compute feature maps of the original image of growing complexity.

The filters that have been learned (in particular, the most primitive ones) are likely to be **independent from the particular kind of images they have been trained on.**

They have been trained on a **huge amount of data** and are probably very good.

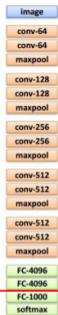
It is a good idea to try to *reuse them* for other classification tasks.

Transfer Learning with CNNs

Transfer Learning with CNNs



1. Train on ImageNet



2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

i.e. swap the Softmax layer at the end



3. If you have medium sized dataset, “**finetune**” instead: use the old weights as initialization, train the full network or only some of the higher layers

retrain bigger portion of the network, or even all of it.

When Transfer Learning makes sense

transferring knowledge from problem A to problem B makes sense if

- the two problems have “similar” inputs
- we have much more training data for A than for B



What we may expect

Faster and more accurate training

