GoogleNet

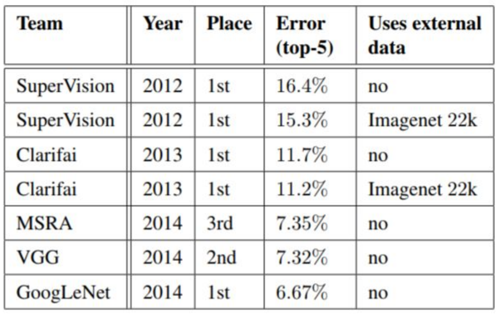
Key contribution of the GoogleNet is reduced error and improved utilization of computational resources, which is very important in mobile devices. How we know mobile devices are very popular nowadays, but they have significantly less computational power and battery life compared to PC's. But applications, using ML becomes more popular, for example, Prisma (makes photo as popular paintings), Google Translator (to translate from camera in realtime).

Differences with other solutions: uses 12 times fewer parameters and more accurate than in paper of A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks, which was best solution in 2012.

To get better performance, we need to increase size (depth - number of layers, width - number of units at each layer) of neural network, but this introduce problems as overfitting and increased use of computational resources. GoogleNet tries to solve this issues by introducing sparsity and replacing the fully connected layers by the sparse.

The fundamental idea of the architecture is to consider how an optimal local sparse structure of a convolutional vision network can be approximated and covered by readily available dense components. In order to avoid patch-alignment issues, current incarnations of the Inception architecture are restricted to filter sizes 1×1, 3×3 and 5×5.

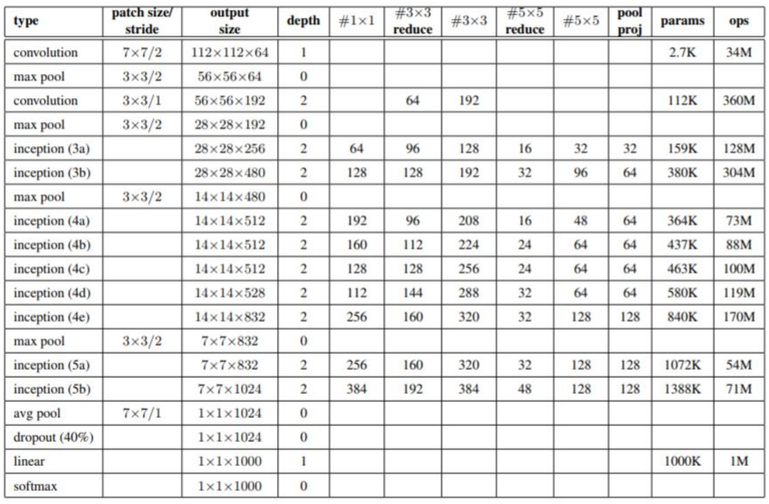
Another important idea of the Inception architecture: judiciously reducing dimension wherever the computational requirements would increase too much otherwise. In general, an Inception network consists of special layers stacked upon each other, with occasional max-pooling layers with stride 2 to divide to 2 equal parts the resolution of the grid. All the convolutions use rectified linear activation.



Team	Year	Place	Error (top-5)	Uses external data
SuperVision	2012	1st	16.4%	no
SuperVision	2012	1st	15.3%	Imagenet 22k
Clarifai	2013	1st	11.7%	no
Clarifai	2013	1st	11.2%	Imagenet 22k
MSRA	2014	3rd	7.35%	no
VGG	2014	2nd	7.32%	no
GoogLeNet	2014	1st	6.67%	no

Table 2: Classification performance.

The network is 22 layers deep when counting only layers with parameters (or 27 layers if we also count pooling). They found that a move from fully connected layers to average pooling improved accuracy by about 0.6%, but using of dropout remained important even after removing the fully connected layers.



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

Table 1: GoogLeNet incarnation of the Inception architecture.

In training they used asynchronous stochastic gradient descent with 0.9 momentum, fixed learning rate schedule (decreasing the learning rate by 4% every 8 epochs) and Polyak averaging to create the final model used at inference time.

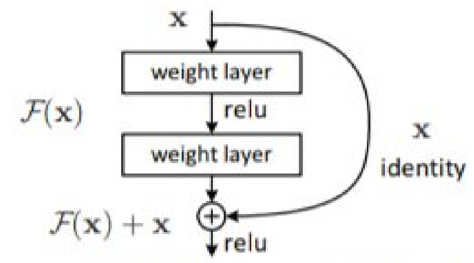
ResNet

Main problem, which they try to solve it is to ease training of networks, which are really deep (deeper than those used previously). They explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. They proved that residual networks are easier to optimize, and can gain accuracy from considerably increased depth. For example, on the ImageNet dataset they estimate residual nets with a depth of up to 152 layers (8 times deeper than VGG nets) but still have lower complexity.

Key problems, they faced: problem of vanishing/exploding gradients and when deeper networks are able to start converging, a degradation problem has been exposed: with the network depth increasing, accuracy gets saturated and then degrades rapidly.

They try to solve them by introducing a deep residual learning framework.

Instead of hoping each few stacked layers directly fit a desired underlying mapping, they explicitly let these layers fit a residual mapping. If we denote the desired underlying mapping as H(x), then let the stacked nonlinear layers fit another mapping of F(x) := H(x)−x. The original mapping is recast into F(x)+x. They hypothesize that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping


x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu
Figure 2. Residual learning: a building block.

How it works: the image is resized with its shorter side randomly sampled for scale augmentation. A 224×224 crop is randomly sampled from an image or its horizontal flip, with the per-pixel mean subtracted. The standard color augmentation is used. They adopt batch normalization right after each convolution and before activation. Also they initialize the weights and train all plain/residual nets from scratch. They use SGD with a mini-batch size of 256. The learning rate starts from 0.1 and is divided by 10 when the error plateaus, and the models are trained for up to 60 × 104 iterations. We use a weight decay of 0.0001 and a momentum of 0.9. Thet do not use dropout.