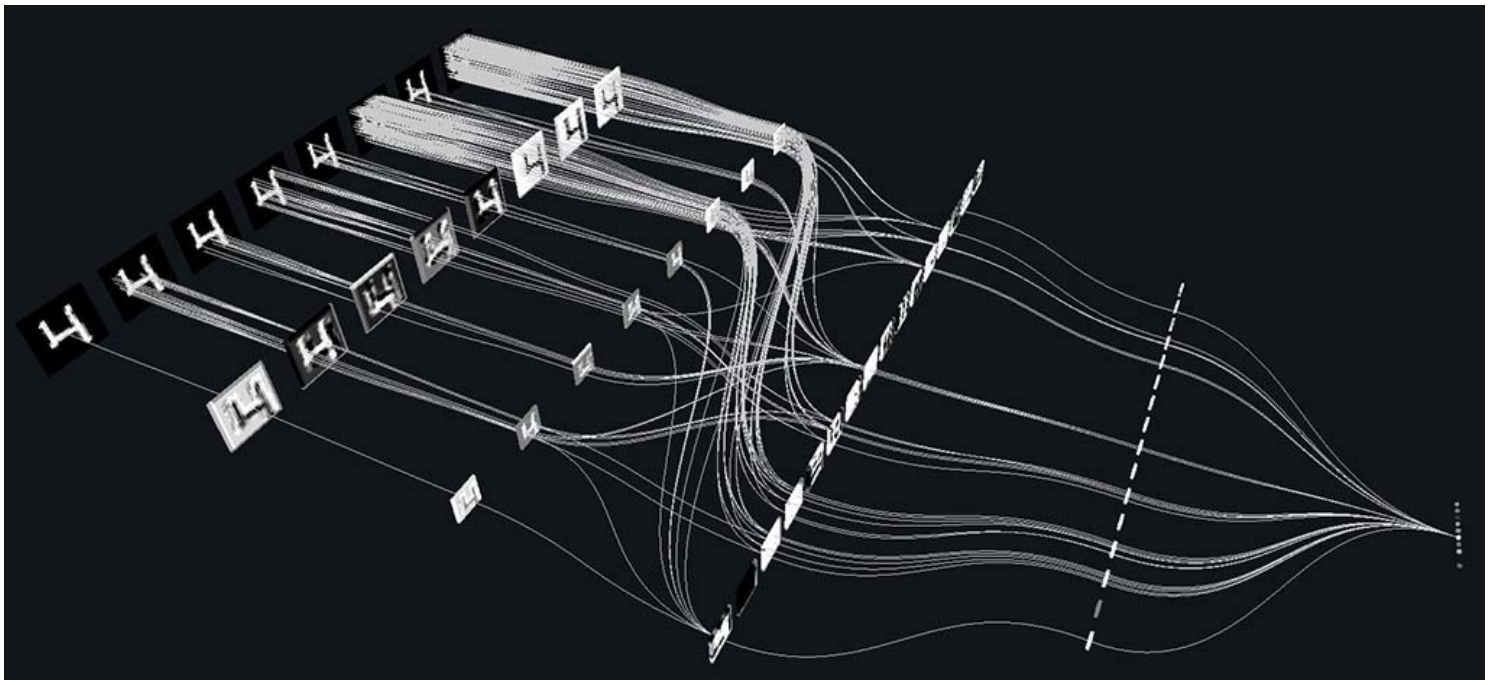


Advancements of convolutional neural networks part 2: Influential network architectures

Dashanka Nadeeshan Follow

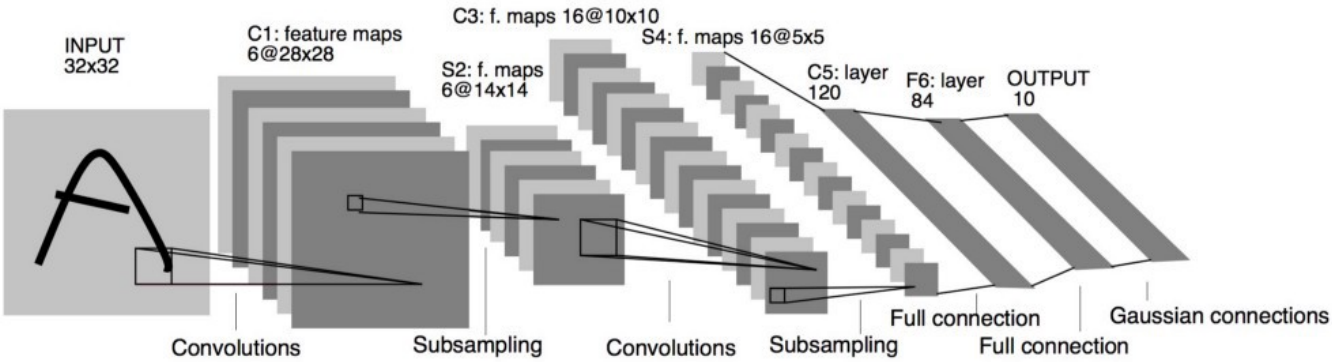
Jul 4, 2018 · 11 min read

Today deep learning has become a powerful and popular class of algorithms in artificial intelligence. Convolutional neural network(CNN)s are one of the major deep learning architectures that is successfully used in many different applications and continuously evolving field of study.

From the previous article, the part 1, we discussed main operations of and their recent advancements briefly. This article briefly discusses most influential CNN architectures.

The learning of CNN is mainly based on the process that starts with local features directly from inputs to gradually building of high-level features. This idea of **hierarchical learning** first introduced by Hubel and Wiesel in 1959. They discovered the concept of the learning process of the primary visual cortex (The so called cat experiment) which is later becoming the inspiration to the CNNs. Then in 1980, Fukushima *et al* built the first learning artificial neural network called Neocognitron.

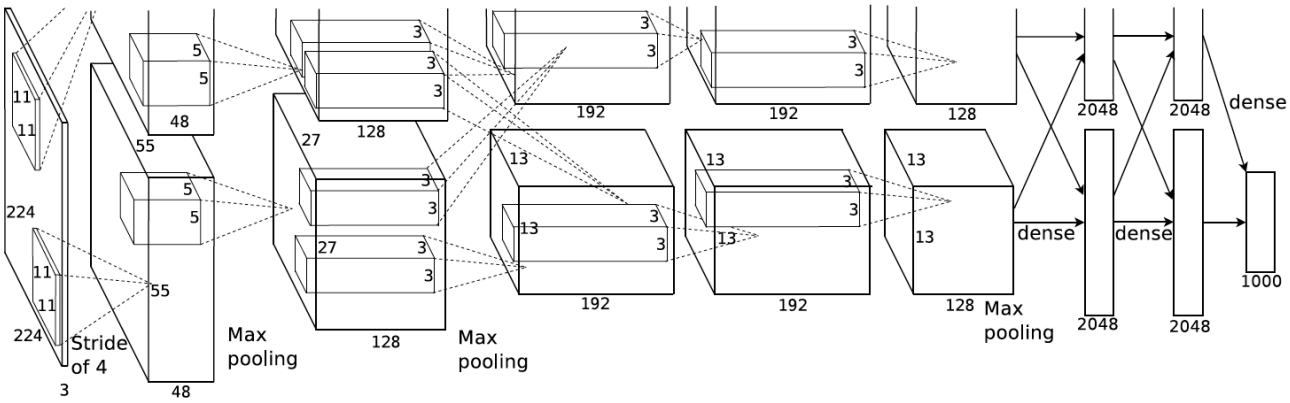
LeNet 5 (1994)



One of the very first CNN introduced in 1994 developed by Yann LeCun, one of the pioneering scientists in the field of deep learning. The LeNet 5 was able to extract the feature insight over the image (input) using convolution with filter kernels who are composed of learnable parameters. Since these filters only look at local patch during a single convolution which is also known as receptive field, the network needed a fewer number of trainable parameters compared to conventional neural network models. However, convolution over whole input area lets layers of the network to map distributed spatial features across the entire input area which are known as feature maps.
One of the main difference between conventional neural networks and conventional artificial neural networks is that the connections between the neurons are not fully connected. Instead only the local patches are taken into account per convolution. This helps to save a large computational cost. The LeNet 5 uses a sequence of 3 types of layers: convolution, non-linearity, and pooling which becomes the standard order of layer layout of CNNs later on. This architecture comprises 7 layers, 3 convolution layers and 2 fully connected layers (or we can say conventional neural network connectivity). For the non-linearity, the hyperbolic tangent squashing function has been used. Altogether the LeNet 5 laid the basic foundation to CNNs.

One of the main drawbacks is that there was not much computational power available on those days like GPUs till most recently and the CPUs were not faster enough. This fact made people to not to pay much attention to this discovery for a while till 2012 (However, there were minor advancements like Dan Ciresan Net in 2010). From 1998 to 2012 there were many advancements and developments in the field of computer science. Especially the fast growth of internet usage and rapid increase of usage of consumer electronics generated more and more data like images and other forms of data. Furthermore, the introduction of GPUs to general purpose computing and CPUs becoming more and more powerful made the deep learning problems solving feasible and efficient.

AlexNet (2012)



The network model that can consider as the first modern successful CNN architecture, Developed by Alex Krizhevsky in 2012. AlexNet is much deeper than the LeNet and able to won the ImageNet challenge in 2012 by a large margin. This was the first CNN model compete for the ImageNet challenge and the achieved success made AlexNet famous. The classic AlexNet architecture comprises 5 convolution layers and 2 fully connected layers. There are several aspects that make this network so powerful. Krizhevsky et al have used rectified linear units which provide better performance over tanh and sigmoids. In order to introduce generalization to or to prevent from over-fitting the network, a technique called dropout has been used which is basically dropping the neuron connections between adjacent layers during the training phase. An important fact is that the network is trained using Nvidia GTX 580 GPUs. This has made training process 10 times faster than using CPUs because of much larger number of cores in GPUs

compared to CPUs. Since AlexNet is deeper than the LeNet, the concept of going deeper with network model came into the play after this. So some of the researchers started building deeper networks.

VGG (2014)

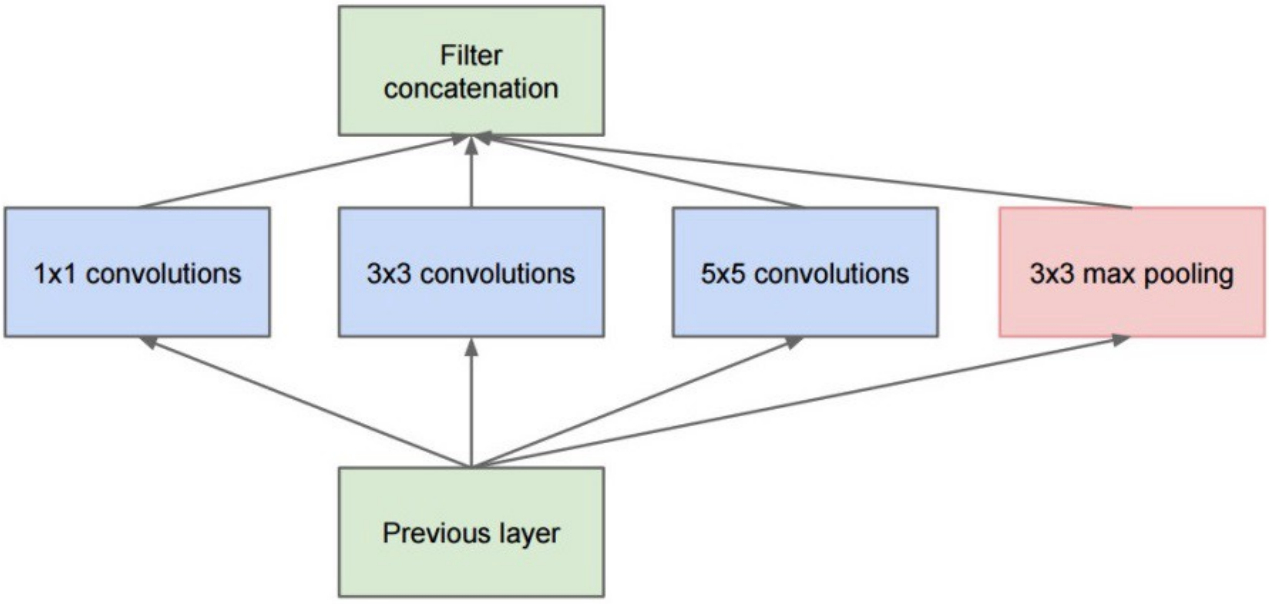
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	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					

With the concept of deeper CNNs, VGG networks are introduced by Simonyan *et al*. There are two main factors that the VGG networks differ from AlexNet: depth of the network and size of the convolution filter. Several network depth configurations are introduced in the paper. Among them, VGG 16 and VGG 19 are the most famous architectures with 13 and 16 convolution layers respectively, and 3 fully connected layers for each network. Size of the convolution filter or the size of the receptive filed used in AlexNet is 11×11 for ImgeNet. But in the VGG networks, authors have used 3×3 receptive fields instead of larger respective fields used by previous top performing entries including 11×11 with stride 4 in (Krizhevsky et al., 2012) and 7×7 with stride 2

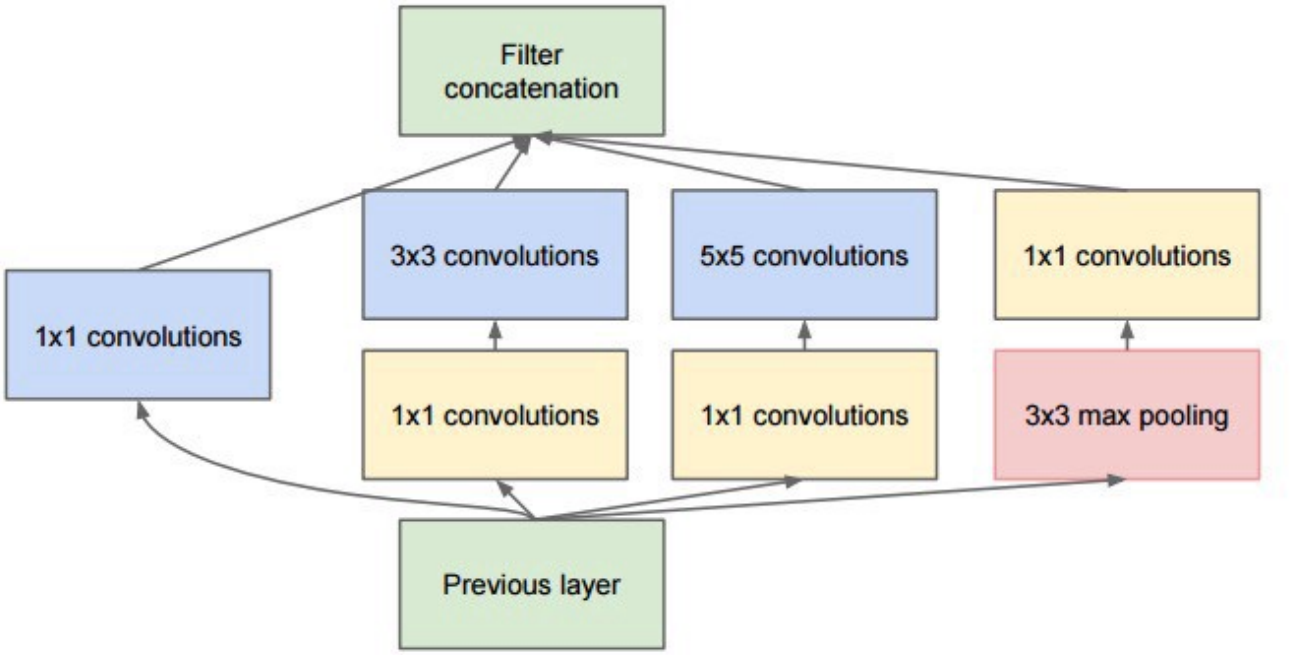
in (Zeiler & Fergus, 2013; Sermanet et al., 2014). The idea is to emulate the larger receptive field by a stack of 3×3 in sequence. As explained in the paper, two 3×3 convolutional layers cover an effective receptive field of 5×5 and three 3×3 such layers have a 7×7 effective receptive field. Why do they use such methodology rather than sticking with the old larger receptive fields? There are two main reasons. The first one is this enables the network to incorporate forward flow with three non-linear rectifications instead of a single. The additional non-linearity introduction makes the decision function more discriminative. The second one is this method reduces the number of parameters significantly. If we understand this in a numerical perspective; a three-layer 3×3 convolution stack with n number of input and output channels has $3(n\times3\times3\times n) = 27n^2$, but for single 7×7 convolution layer with same channels would require $(n\times7\times7\times n) = 7^2n^2 = 49n^2$. This is roughly 81% more parameters which directly effects the learning time and power. But the drawback occurred here is the many numbers of layers, hence a large number of features (maps) developed. This made deploying the VGG networks are computationally expensive, both in terms of memory and time.

GoogLeNet/ Inception V1 (2014)

Also codenamed as Inception (version 1), who established a new state of the art classification in the ImageNet challenge in 2014. GoogLeNet is 22 layer deep CNN but the overall architectural view is quite different from the typical a CNN. The interesting fact is that despite the depth of the network and spread looking architecture, the number of parameters in GoogLeNet is twelve times lesser than the AlexNet. This results in a huge reduction of the computational burden and at the same time, it is more accurate than the state of the art performance that time(ImageNet). The authors made this all possible by coming up with the Inception module.



The figure above from the paper illustrates the basic structure of Inception module which they also named as the naive version. As it seems, the Inception module is a bunch of parallel connections between two layers. These connections operate 1×1, 3×3, and 5×5 convolutions. The 1×1 convolutions are able to detect correlations in local patches while larger convolutions like 3×3 and 5×5 detect relatively larger spatial insights. But the problem occurred is that this large number of filters make the whole process computationally expensive. It is solved by the introduction of so-called bottleneck layer.



The basic idea of bottleneck layer is to apply 1×1 convolutions before 3×3, and 5×5 convolutions as shown in the Inception module in the figure above. As is explained in the paper, "In general, an Inception network is a network consisting of modules of the above type stacked upon each other, with occasional max-pooling layers with stride 2 to halve the resolution of the grid". What bottleneck layer does is, instead of the convolution of all input feature to outputs using 3×3 or 5×5 which is computationally expensive, first perform 1×1 convolutions with a reduced number of input features. Then perform convolutions with Inception module branches which are 3×3 and 5×5. Finally the concatenation of output features. If we understand this using an example (as explained in the paper), let us take an Inception module (the first module) which has 192 input channels. This module has 128, 3×3 filters and 32, 5×5 filters. The number of computations for each filter will be $192\times3\times3\times128=221{,}184$ and $192\times5\times5\times32=153{,}600$ which are a lot and may blow as it goes deeper into the network too. Therefore as explained before the Inception module applies 1×1 convolutions before applying larger convolutions. In the paper they have used 16, 1×1 convolutions, so $192\times1\times1\times16$ computations will take place. Furthermore, the total number of computations with 3×3 will be $192\times1\times1\times16+16\times3\times3\times128=21{,}504$ and with 5×5 will be $192\times1\times1\times16+16\times5\times5\times32=15{,}872$. These evaluations show that the Inception module is able to reduce the number of operations by nearly 10 times compared to the normal CNN connections.

In addition to all these modifications in convolution layers, GoogLeNet has replaced the fully connected layer with a simple global average pooling from the last convolution layer. This is also helping to reduce the number of parameters significantly. Another important architectural modification is that the authors have connected auxiliary classifiers to intermediate layers to successfully propagate the gradients back through every layer to tweak the learnable parameters and prevent the vanishing gradient problem.

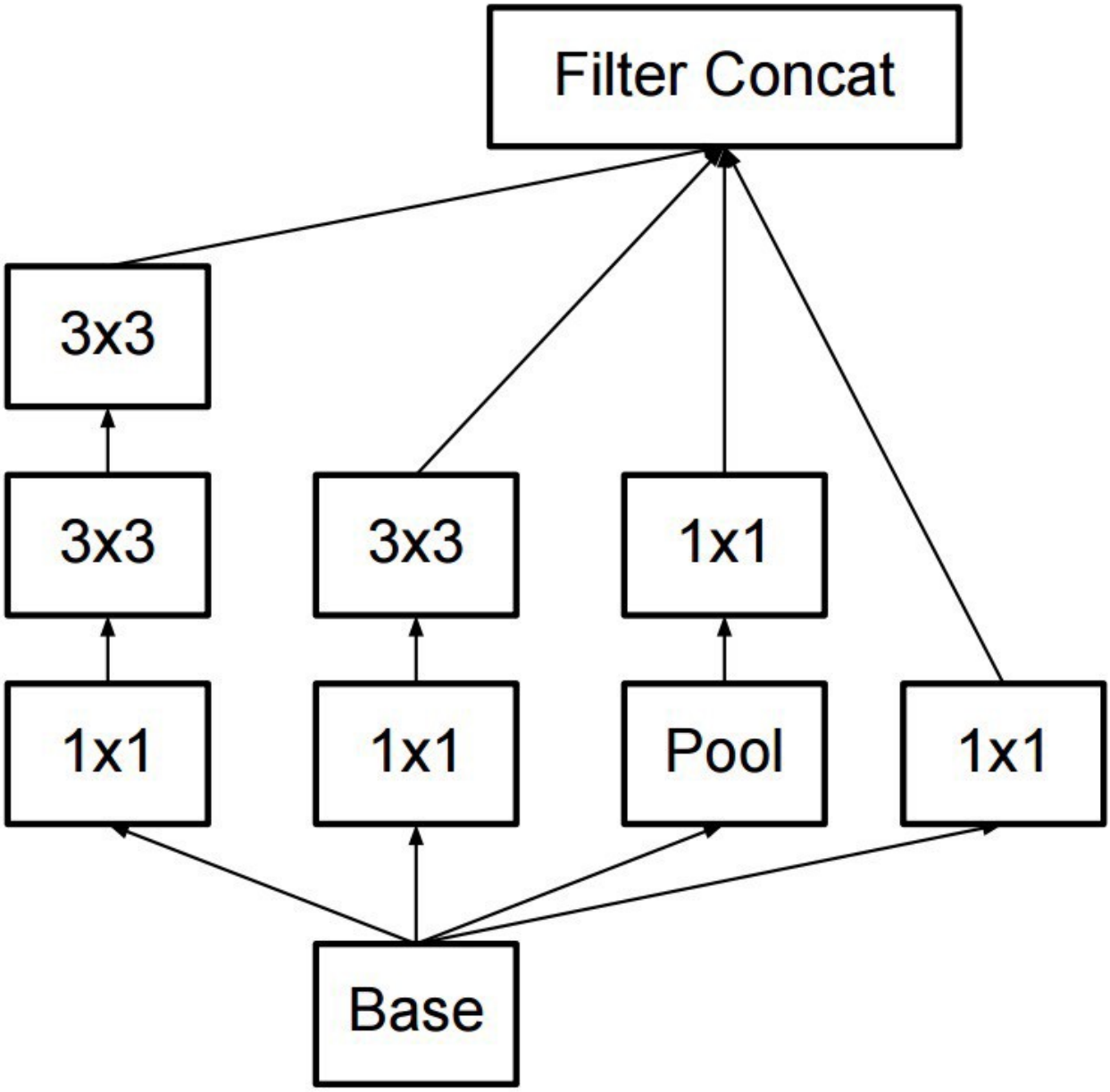
Inception V2 (Early 2015)

In early 2015, Szegedy et al came with the second version of the Inception, Inception V2. As per the novelty, batch normalization is introduced to the Inception. Generally, batch normalization introduces a data whitening process to learned features. It is done by computing the mean and the standard deviation of all feature maps and normalize the

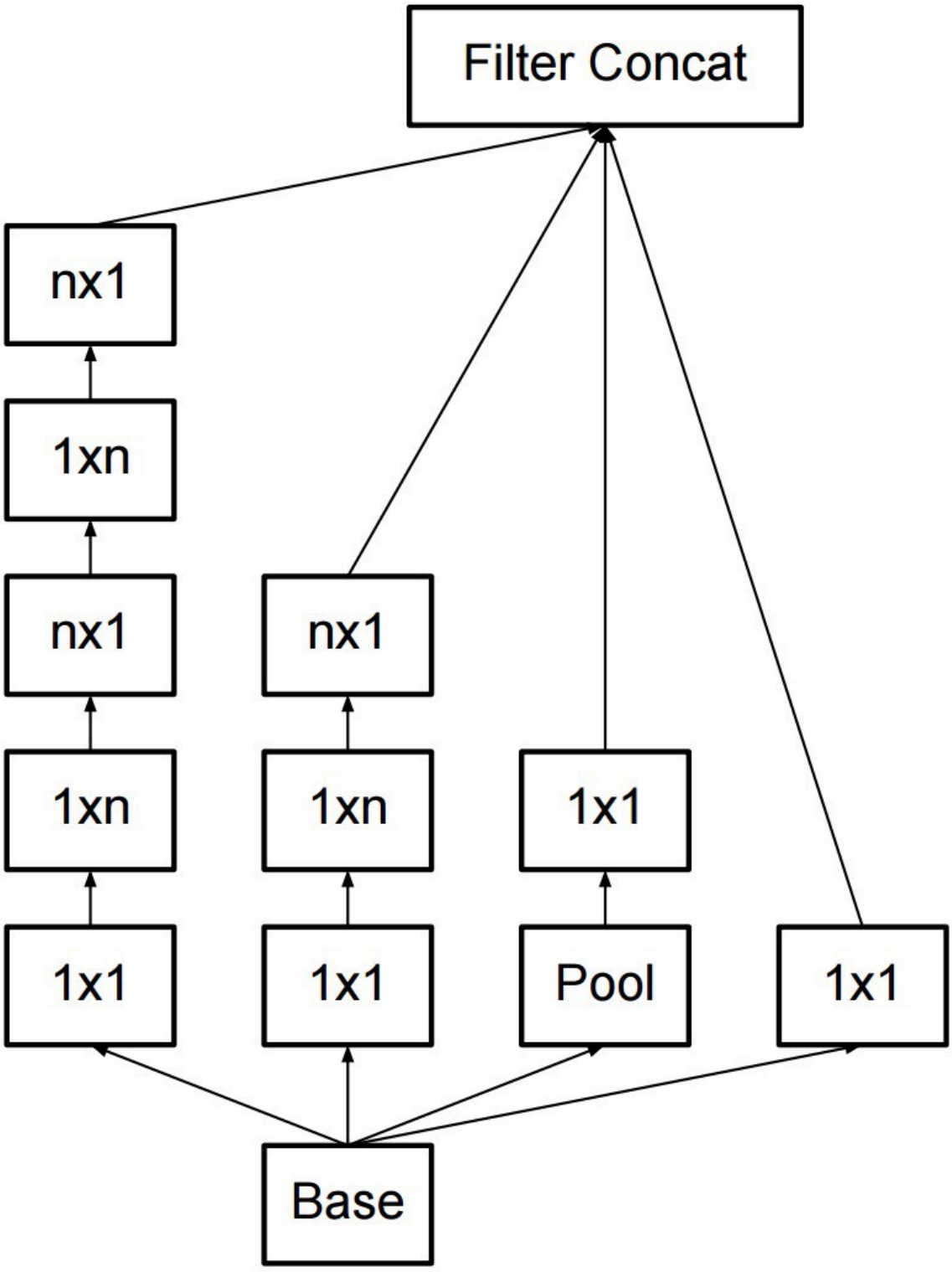
responses by making them zero mean and a have same range. This helps the training process by, to not to worry about learning data offsets of the inputs at the following layer.

Inception V3 (Late 2015)

The next version came out at the end of the same year with a new version of the Inception, Inception V3. There are several modifications they have performed. Decomposed the 5×5 and 7×7 filters with multiple 3×3s. This process is also known as the factorization of convolutions.



The second one is to factorization of n×n convolutions by a 1×n convolution followed by n×1 convolution. In theory, this method reduces the computational cost as *n* grows, but in practice, factorization works well on medium grid-sizes. Therefore the authors have used 1×7 followed by 7×1 convolutions.

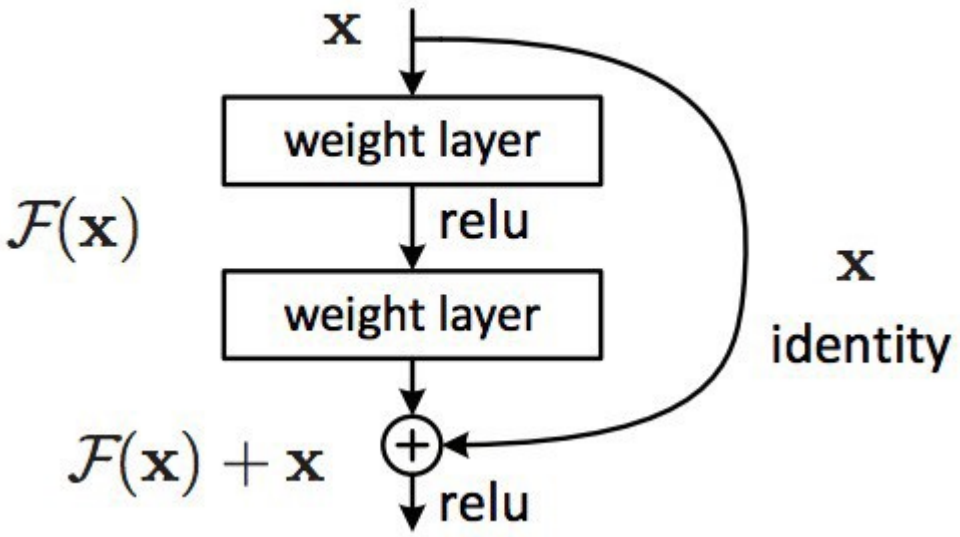


inception modules can also decrease the size of the data by providing pooling while performing the inception computation. This is basically identical to performing a convolution with strides in parallel with a simple pooling layer.

The first Inception network introduced with the auxiliary classifiers to make sure gradient flowing through the whole network and improve the convergence during the training phase by fighting with the vanishing gradient problem. The authors from the 3rd version (paper) claim that the advantage of auxiliary branches occurs near the end of the training and argue that the auxiliary classifiers act more likely as a regularizer.

ResNet (2015)

Theoretically, deeper the network is, the ability to learn complex feature representations and hence the model accuracy gets better. But in practice, these networks tend to get over-fitted and suffer from the problem of vanishing gradients. Another problem occurs when training such deeper networks is the degradation problem. During the training phase, deeper networks have a large number of parameter to optimize. Adding more layers or increasing the depth of the network will lead to a generation of higher training error. This is known as the degradation problem. The authors have addressed this problem by introducing the deep residual framework.



Let's say we added a one more layers to the network. The training error to not to go beyond the previous error (before adding the layer) there can be two options. The newly

added layer does not perform learning or it only performs identity mapping. By formulating this idea the authors have introduced the residual framework. This framework comprises residual modules who provide the solution to the degradation problem. If we consider a single module, it creates a direct path between the input and the output to the module (as shown in the image above). These shortcut connection perform identity mapping and do not contain any parameters or computational complexity. Since this identity mapping provides the original conditions of the network before adding the extra layer, the additional layer only has to learn the residuals. This is called as residual mapping and it does not affect the training error undesirably. If we look at the image above, *F(x)* is the residual mapping while *x* provides the identity mapping. The residual module finally concatenates the mapping as element-wise addition and pass through a non-linearity to generate the output. Furthermore, the paper claims that the residual block may contain two or three layers as the residual (*F*) function is flexible to do so. Therefore, the presented the residual network and the plain network (without residual connections) fairly have the same number of parameters, depth, width and computational cost except for the negligible element-wise additions. The authors have evaluated and shown that the ResNet architecture with 152 layers depth has achieved better classification accuracy and computationally efficient than VGG networks and GoogLeNet.

MobileNet (2017)

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$

Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

CNNs are inherently resources consuming and cumbersome for real-time applications with restricted hardware capabilities like mobile applications or embedded systems. Therefore, in 2017 MobileNet is introduced by Howard *et al*. to address this issue. The core idea is to make the network more efficient by replacing the standard convolutions with depth wise separable convolutions. The basic idea of the standard convolution is to combine the output of a local patch across the input area (2 dimensional) and across the whole input depth. Depth wise separable convolutions are a form of factorized convolutions. It basically split the standard convolution process into two distinct steps. Firstly a depthwise convolution and secondly a pointwise convolution. The depthwise convolution performs convolution over the square region within a single depth slice. The pointwise convolution merges the information obtained from the previous step across the whole depth using 1×1 convolution. The MobileNet architecture is built using discussed depth wise separable convolutions and a single full convolution for the first layer.

. . .

Over the past decade, there has been a lot of research conducted and many CNN architecture variations have been introduced. But the core principle and the framework stays similar. In order to thoroughly understand the intuition and algorithms behind the discussed architectures, it is highly recommended to go through the relevant published paper (mentioned here too). When we inspect the general development over time, it can clearly see how fast the field is advancing. Generally speaking, each year the classification error (from ILSVRC) for CNN classification becomes lesser and in 2016 it has beaten the human error of 5%. This article only discusses the most influential CNN architectures who have shaped the current state of the art. But yet, there are many other architectural developments and algorithms available in the literature that to be discussed.