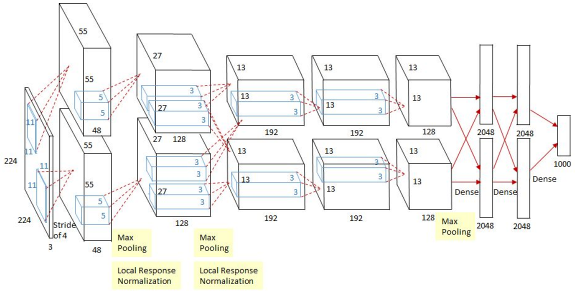
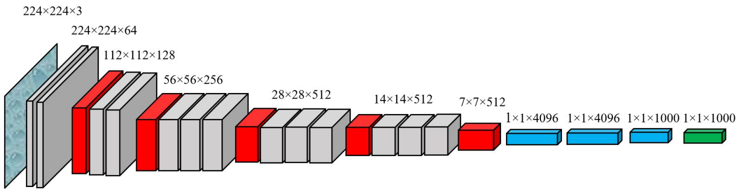
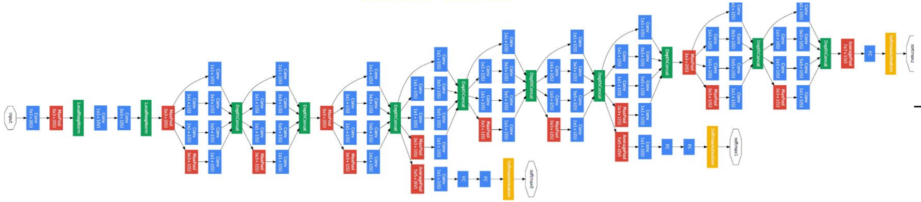
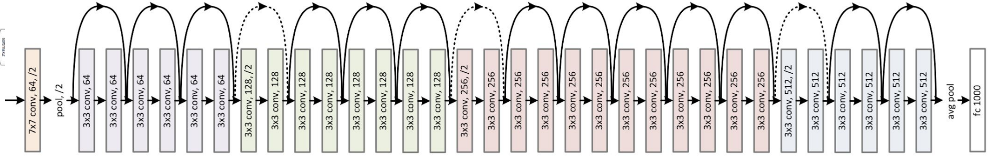
CSEP 576: Advanced CNNs









Jonathan Huang (jonathanhuang@google.com)
University of Washington 17 May 2020
Google Research

Lecture Outline

May 19

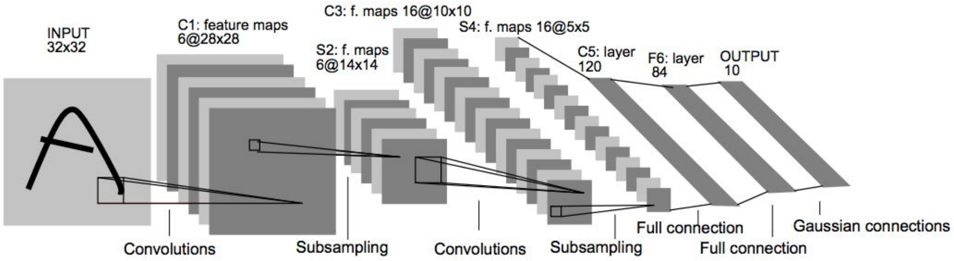
- Part 1: Advanced CNNs (Focusing on classification)
  - Reusable higher level building blocks of modern convnet architectures
    - Dropout, Batch Norm, Factorized Convolutions, Residual Connections, etc.
  - Tour through “popular” classification architectures
    - E.g., AlexNet, VGG, GoogLeNet, Resnet, MobileNet, SE-Net
- Part 2: Object Detection
  - Motivation, Applications
  - Anchor based detection methodology:
  - Single stage and Two stage meta-architectures
  - Evaluation metrics
  - Practical Tips

LeNet-5 Review

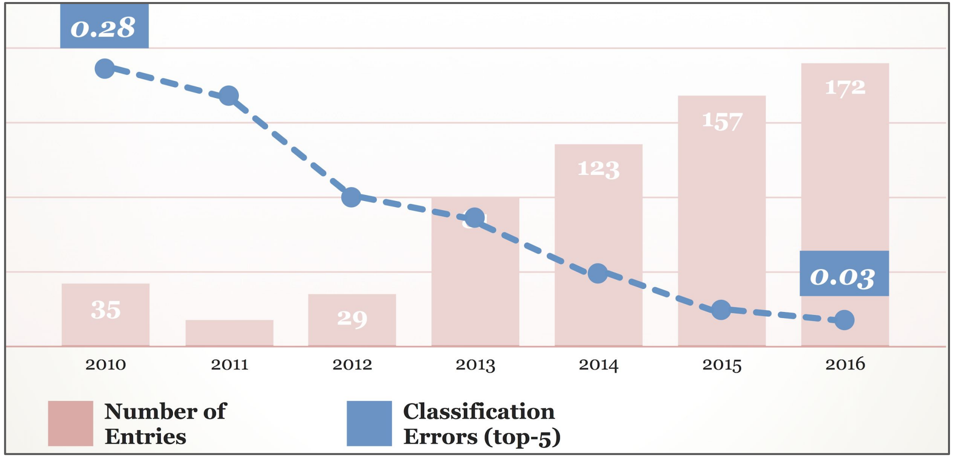
- Input 32x32
- Conv(5x5, 1->6) -> Tanh
- MaxPool(2, 2)
- Conv(5x5, 6->16) -> Tanh
- MaxPool(2, 2)
- Flatten
- FC(400 -> 120) -> Tanh
- FC(120 -> 84) -> Tanh
- FC(84 -> 10)

Two Convs w/valid padding, Three FCs
Params: 25*6 + 25*6*16 + 400*120 + 120*84+84*10 = 61470

FLOPS:
28^2 * 5*5*6+14^2 * 4 * 20+10^2 * 5 *5 * 6*16+5^2 * 4 * 16+400*120+120*84+84*10 =433800



Timeline of Events



- 1958 Perceptron (Rosenblatt et al)
- 1985 Backprop (Hinton et al)
- 1989 LeNet (LeCun et al)
- 1998 LeNet-5 (LeCun et al)
- Late aughts - rekindled interest in neural nets, deep learning
- 2009 - Imagenet
- 2012 - AlexNet - a turning point!
- Post-AlexNet = Deep Learning revolution

Focus of Today's lecture

Our focus today

- AlexNet and LeNet (from 1980s) very similar; What's changed?
    - More data...
    - **Deeper models**
    - **More efficient**

- Example details that will be covered today
    - ReLU
    - Batch Normalization
    - Factored convolutions
    - Residual connections
    - Squeeze-and-excitation layers
- We won't cover efficiency coming from hardware advances over the years

Let’s take a tour through the AlexNet paper…

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

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called “dropout” that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

1 Introduction

Current approaches to object recognition make essential use of machine learning methods. To improve their performance, we can collect larger datasets, learn more powerful models, and use better techniques for preventing overfitting. Until recently, datasets of labeled images were relatively small — on the order of tens of thousands of images (e.g., NORB [16], Caltech-101/256 [8, 9], and CIFAR-10/100 [12]). Simple recognition tasks can be solved quite well with datasets of this size, especially if they are augmented with label-preserving transformations. For example, the current-best error rate on the MNIST digit-recognition task (<0.3%) approaches human performance [4]. But objects in realistic settings exhibit considerable variability, so to learn to recognize them it is necessary to use much larger training sets. And indeed, the shortcomings of small image datasets have been widely recognized (e.g., Pinto et al. [21]), but it has only recently become possible to collect labeled datasets with millions of images. The new larger datasets include LabelMe [23], which consists of hundreds of thousands of fully-segmented images, and ImageNet [6], which consists of over 15 million labeled high-resolution images in over 22,000 categories.

To learn about thousands of objects from millions of images, we need a model with a large learning capacity. However, the immense complexity of the object recognition task means that this problem cannot be specified even by a dataset as large as ImageNet, so our model should also have lots of prior knowledge to compensate for all the data we don’t have. Convolutional neural networks (CNNs) constitute one such class of models [16, 11, 13, 18, 15, 22, 26]. Their capacity can be controlled by varying their depth and breadth, and they also make strong and mostly correct assumptions about the nature of images (namely, stationarity of statistics and locality of pixel dependencies). Thus, compared to standard feedforward neural networks with similarly-sized layers, CNNs have much fewer connections and parameters and so they are easier to train, while their theoretically-best performance is likely to be only slightly worse.

Influential over many many later papers w/over 60K cites on Google Scholar (as of May 2020):

- ReLU
- Multi-GPU
- Data augmentation
- Push to go deeper
- 224x224

AlexNet Architecture

Much bigger input than LeNet!
Important design consideration;
Too small, hard to recognize;
Too large, computational challenges

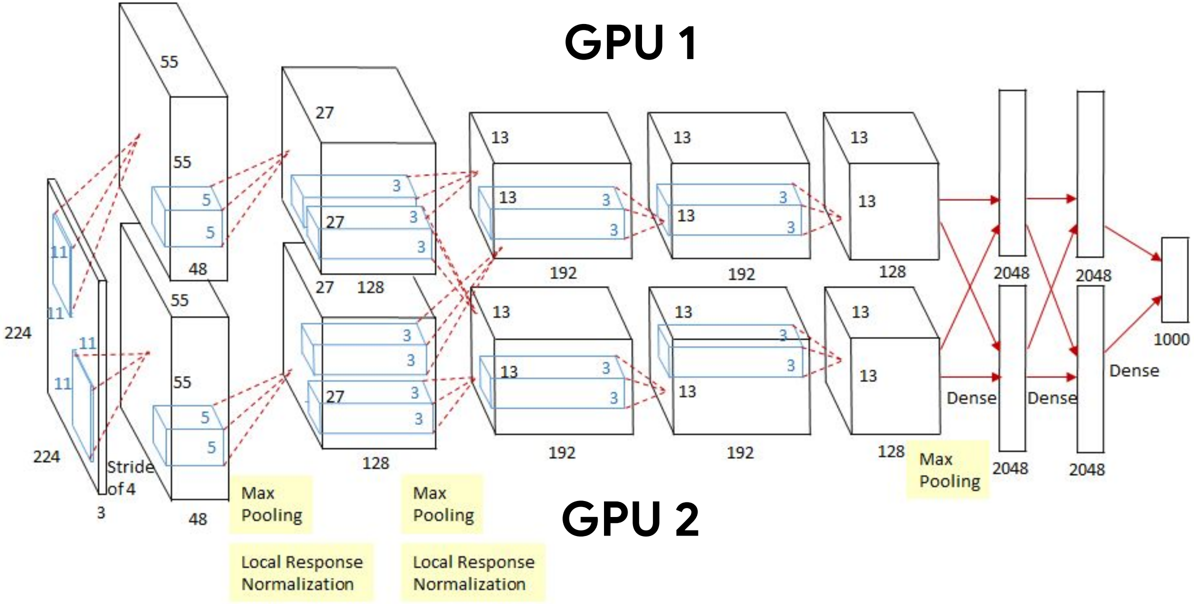
- Input 224x224 (or 227x227)
- Conv(11x11, 3->96, stride 4) -> ReLU -> LRN
- Pool (3, 2)
- Conv(5x5, 96->256, stride 1) -> ReLU -> LRN
- Pool (3, 2)
- Conv(3x3, 256->384, stride 1) -> ReLU
- Conv(3x3, 384>384, stride 1) -> ReLU
- Conv(3x3, 384>256-, stride 1) -> ReLU
- Pool(3, 2)
- FC(9216 -> 4096)
- FC(4096 -> 4096)
- FC(4096 -> 1000)

Deeper than LeNet
5 Convs, 3 FC

LRN mostly not used these days; we won't talk about it

Overlapping pooling - we also won't cover this

Multi GPU training



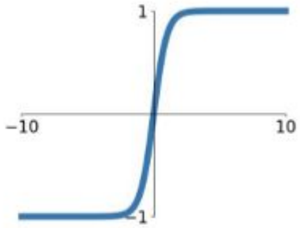
This is *model parallelism* --- these days *data parallelism* more common

See AlexNet paper for details; also [One weird trick for parallelizing convolutional neural networks](#)
(also by Alex Krizhevsky)

ReLU vs Tanh nonlinearities

tanh

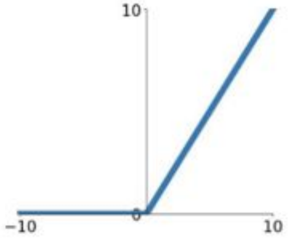
$\tanh(x)$



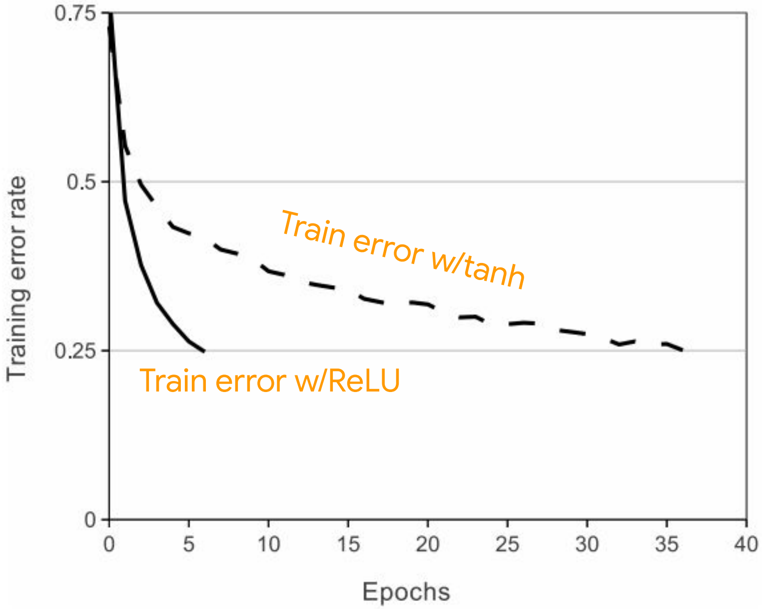
Problem with tanh is that signal *saturates* easily *(w/gradient magnitudes becoming extremely small)* leading to slow training

ReLU

$\max(0, x)$



In positive region, ReLU doesn't saturate (constant gradient!)

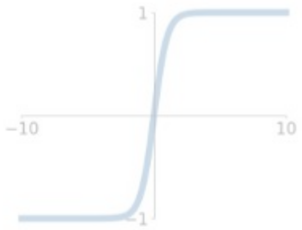


Example on CIFAR-10 (this is not with AlexNet)

ReLU vs Tanh nonlinearities

tanh

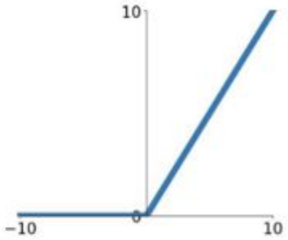
$\tanh(x)$



Problem with tanh is that signal *saturates* easily *(w/gradient magnitudes becoming extremely small)* leading to slow training

ReLU

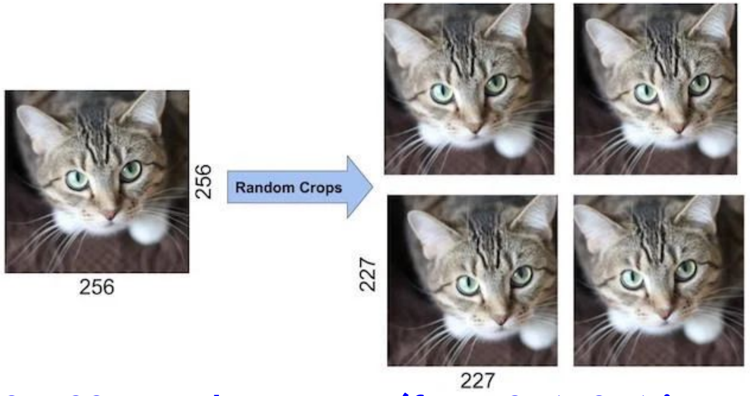
$\max(0, x)$



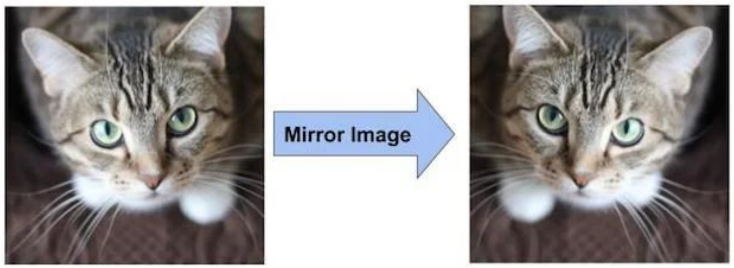
In positive region, ReLU doesn't saturate (constant gradient!)

- Almost universally adopted
- Very fast computationally
- Still saturates in negative region
  - Needs good initialization (or batch norm, as we will discuss later)
- Competitors:
  - PReLU, ELU, Leaky ReLU, SELU, Swish
- <u>Can lead to overconfident predictions far away from training data</u>

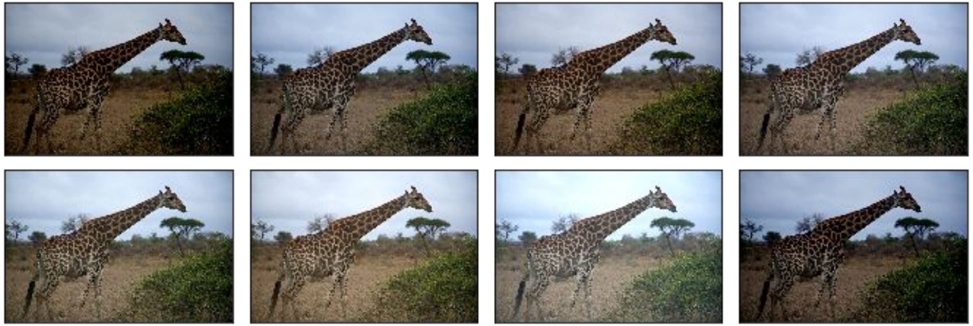
Data Augmentation - Training time



224x224 random crops (from 256x256 inputs)



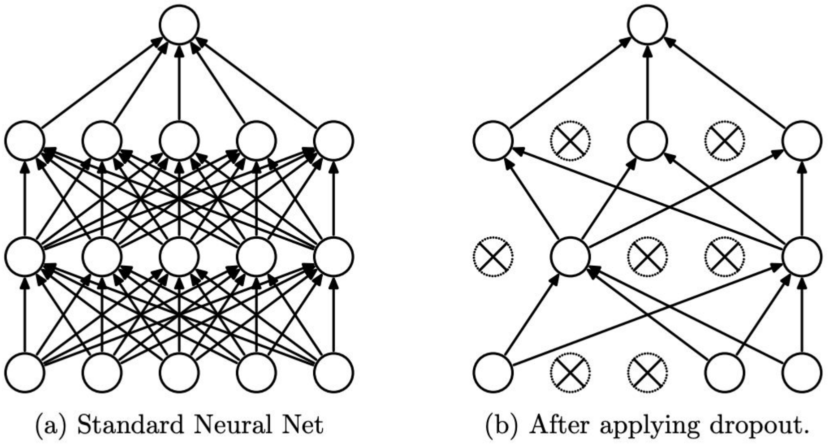
Random horizontal flips



Random color distortions based on PCA applied to RGB pixels

Figure credit: https://www.learnopencv.com/understanding-alexnet/, https://mxnet.apache.org/versions/1.5.0/tutorials/gluon/data_augmentation.html

Dropout Regularization



"Drop" neurons w/probability p

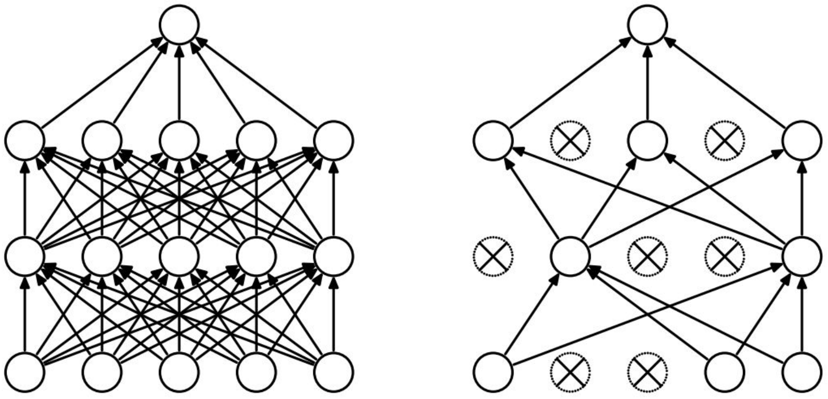
Idea:

- **Training time:**
  - Scale layer by (*1/p*)
  - Set each neuron in layer to zero with probability *p*

- **Test time:**
  - Don't do dropout

Dropout: A Simple Way to Prevent Neural Networks from Overfitting by Srivastava et al

Dropout Regularization



(a) Standard Neural Net (b) After applying dropout.

"Drop" neurons w/probability p

Idea:

- **Training time:**
  - Scale layer by ($1/p$)
  - Set each neuron in layer to zero with probability p

- **Test time:**
  - Don't do dropout

Why scale by $1/p$?
If x is value of neuron and w is its weight, under Dropout, we have:

$$E[w * (x/p)] = w * x$$

Dropout: A Simple Way to Prevent Neural Networks from Overfitting by Srivastava et al

Dropout Regularization

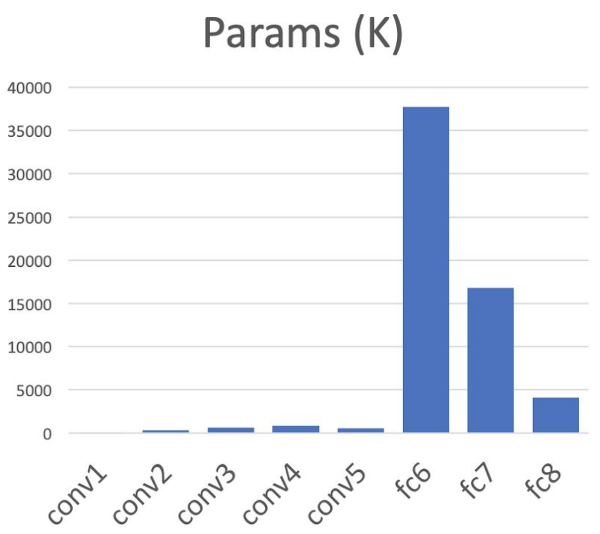
- Reduces “co-adaptation” of neurons and leads to more robust/redundant features
- Tends to be used with large FC layers

- Usually requires longer training
- Less ubiquitous these days (but still used) --- the idea of randomly perturbing something at training time and averaging over the randomness at test time is *very* common

Parameter counting

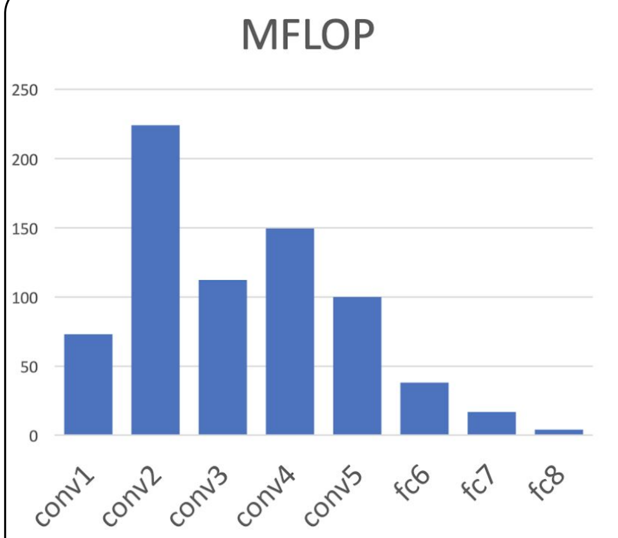
Note: 60 M parameters trained on ~1 M images!

Most memory in early convs



Most parameters in FC layers

We will see a move not to use FC



Most compute in mid conv layers

We will see factored conv layers and bottleneck layers as a solution to this in later papers

But mostly... we will see that things will just get more compute intensive :)

Figure credit: Justin Johnson

ImageNet experiments

Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**

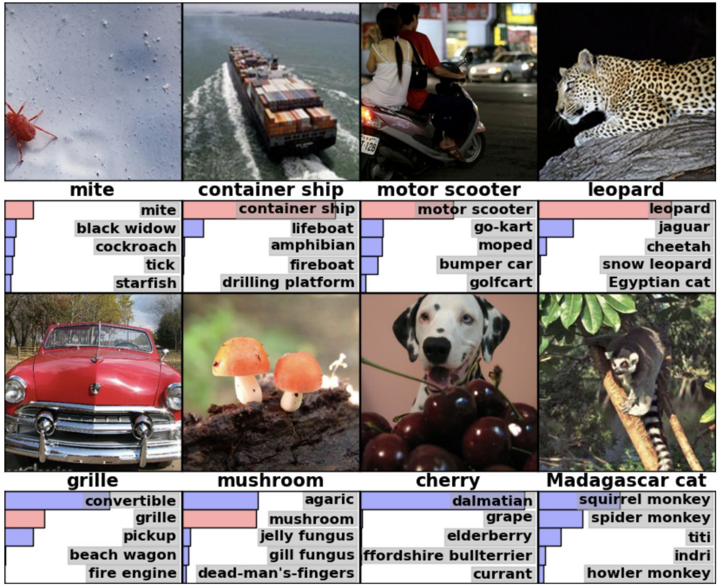
Comparison against ImageNet SOTA at the time

Preprocessing:

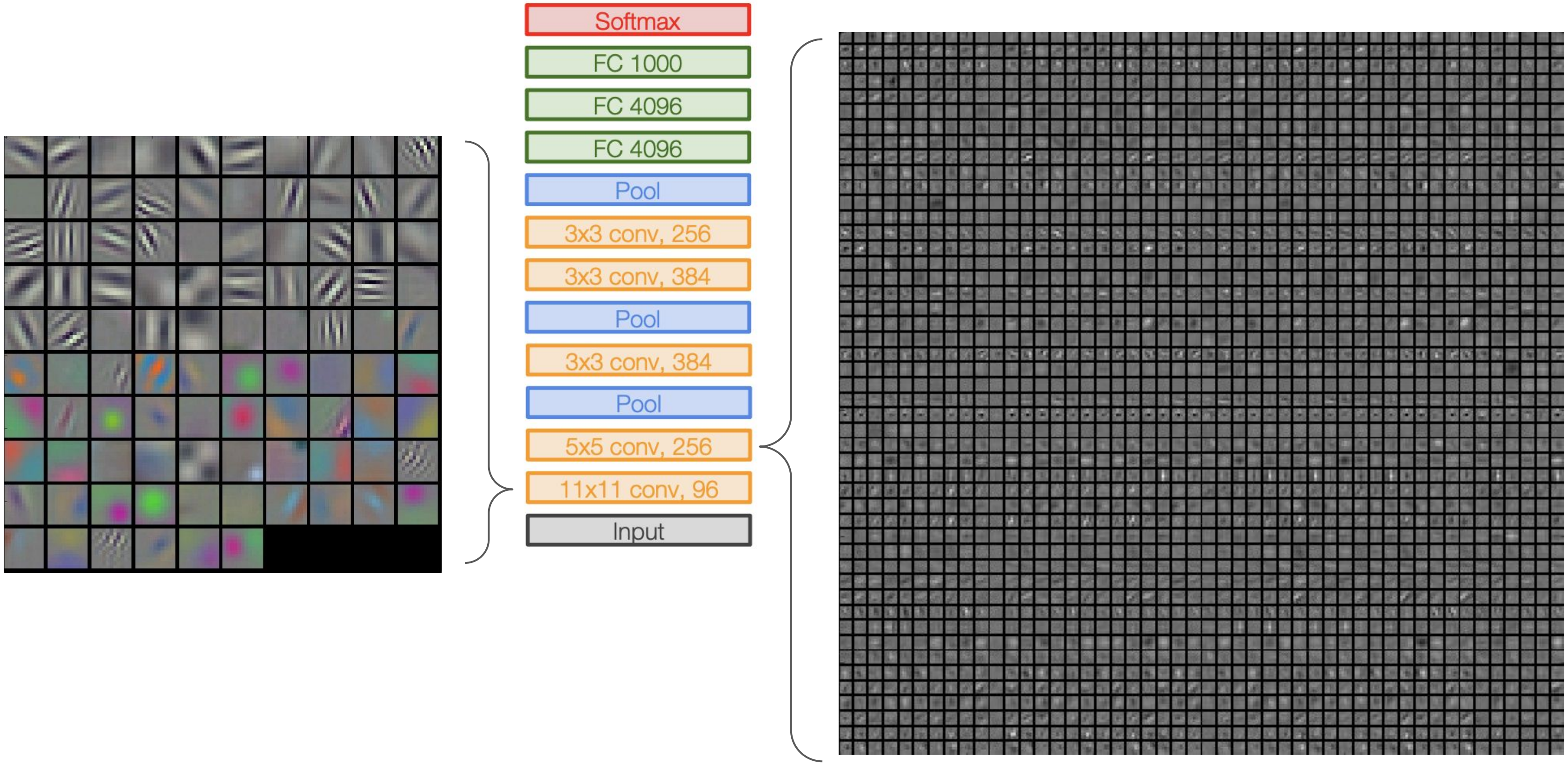
- Subtract mean RGB from each pixel

Optimization:

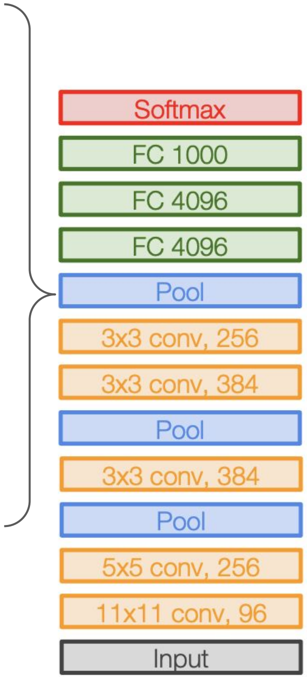
- SGD momentum
- Batch size 128
- 5-6 days of training



What are those layers doing?



What are those layers doing?

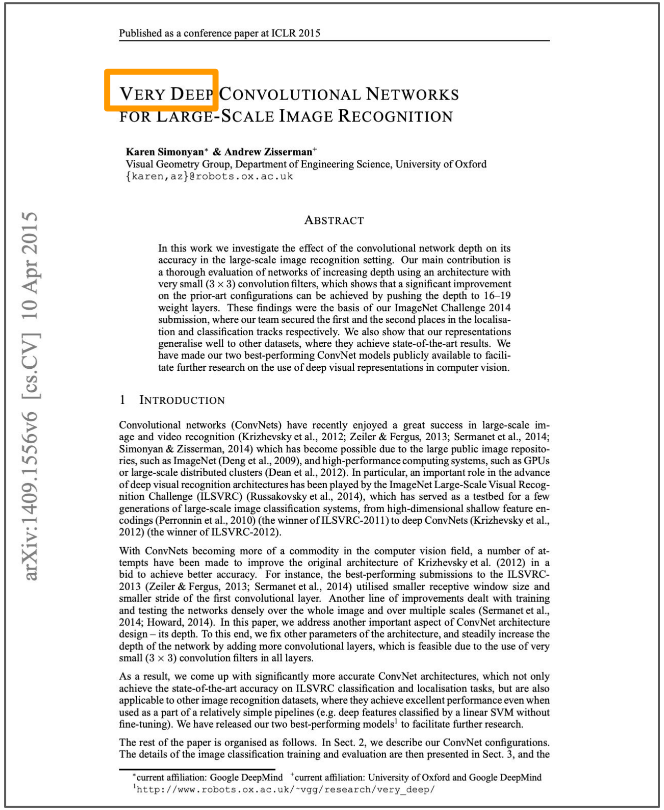


Rich feature hierarchies for accurate object detection and semantic segmentation by Girshick et al.

AlexNet Recap

- Deeper than LeNet-5!
  - 5 Conv, 3 FC vs 2 Conv + 3 FC
  - 60 M vs 60K parameters
- ReLU (vs Tanh)
- DropOut regularization
- 224x224-ish inputs
- Multi GPU training
- Data Augmentation

Case Study (2014): VGG



Published as a conference paper at ICLR 2015

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

Karen Simonyan* & Andrew Zisserman+
Visual Geometry Group, Department of Engineering Science, University of Oxford
{karen,az}@robots.ox.ac.uk

arXiv:1409.1556v6 [cs.CV] 10 Apr 2015

ABSTRACT

In this work we investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. Our main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small (3 × 3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19 weight layers. These findings were the basis of our ImageNet Challenge 2014 submission, where our team secured the first and the second places in the localisation and classification tracks respectively. We also show that our representations generalise well to other datasets, where they achieve state-of-the-art results. We have made our two best-performing ConvNet models publicly available to facilitate further research on the use of deep visual representations in computer vision.

1 INTRODUCTION

Convolutional networks (ConvNets) have recently enjoyed a great success in large-scale image and video recognition (Krizhevsky et al., 2012; Zeiler & Fergus, 2013; Sermanet et al., 2014; Simonyan & Zisserman, 2014) which has become possible due to the large public image repositories, such as ImageNet (Deng et al., 2009), and high-performance computing systems, such as GPUs or large-scale distributed clusters (Dean et al., 2012). In particular, an important role in the advance of deep visual recognition architectures has been played by the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) (Russakovsky et al., 2014), which has served as a testbed for a few generations of large-scale image classification systems, from high-dimensional shallow feature encodings (Perronnin et al., 2010) (the winner of ILSVRC-2011) to deep ConvNets (Krizhevsky et al., 2012) (the winner of ILSVRC-2012).

With ConvNets becoming more of a commodity in the computer vision field, a number of attempts have been made to improve the original architecture of Krizhevsky et al. (2012) in a bid to achieve better accuracy. For instance, the best-performing submissions to the ILSVRC-2013 (Zeiler & Fergus, 2013; Sermanet et al., 2014) utilised smaller receptive window size and smaller stride of the first convolutional layer. Another line of improvements dealt with training and testing the networks densely over the whole image and over multiple scales (Sermanet et al., 2014; Howard, 2014). In this paper, we address another important aspect of ConvNet architecture design – its depth. To this end, we fix other parameters of the architecture, and steadily increase the depth of the network by adding more convolutional layers, which is feasible due to the use of very small (3 × 3) convolution filters in all layers.

As a result, we come up with significantly more accurate ConvNet architectures, which not only achieve the state-of-the-art accuracy on ILSVRC classification and localisation tasks, but are also applicable to other image recognition datasets, where they achieve excellent performance even when used as a part of a relatively simple pipelines (e.g. deep features classified by a linear SVM without fine-tuning). We have released our two best-performing models[1] to facilitate further research.

The rest of the paper is organised as follows. In Sect. 2, we describe our ConvNet configurations. The details of the image classification training and evaluation are then presented in Sect. 3, and the

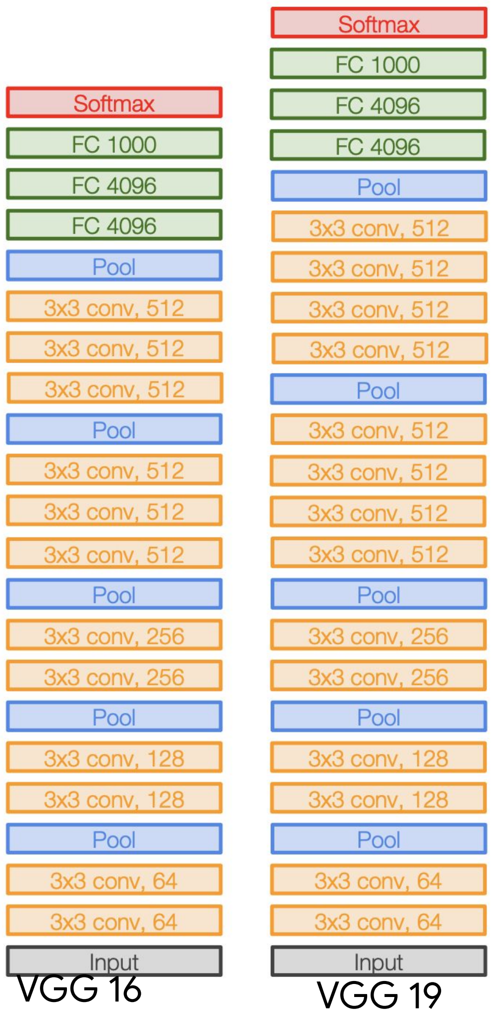
*current affiliation: Google DeepMind +current affiliation: University of Oxford and Google DeepMind
[1]http://www.robots.ox.ac.uk/~vgg/research/very_deep/

From 8 layers to ~20 layers!

VGG Design Pattern

(Influential on many upcoming networks)

- 3x3 convs, stride 1, pad 1
- 2x2 pool, stride 2
- After pool, double channels (until 512)

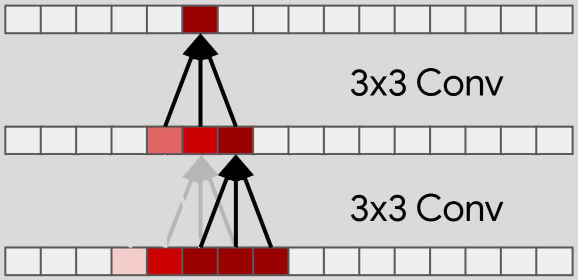


VGG Design Pattern
(Influential on many upcoming networks)

- **3x3 convs, stride 1, pad 1**
- 2x2 pool, stride 2
- After pool, double channels (until 512)

Let's think about two stacked 3x3 Convs vs one 5x5 Conv:
- Same receptive field;
- With intermediate ReLU, stacked version is "deeper";
- Stacked version is more efficient

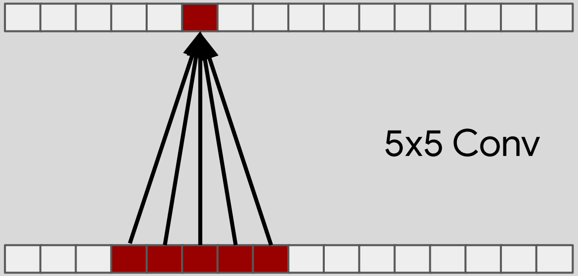
Jon's note: By FLOPS in this slide deck, I actually mean mult-add :P



Stacked 3x3 Convs

Parameters: 2 * 3 * 3 * C^2 = 18 C^2

FLOPS: 18 * C^2 * H * W



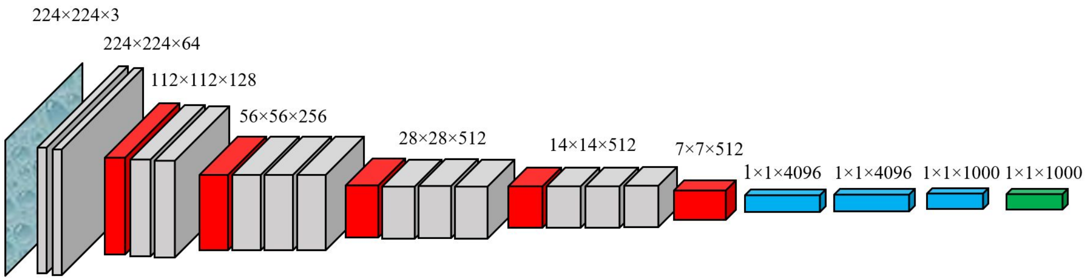
5x5 Conv

Parameters: 5 * 5 * C^2 = 25 C^2

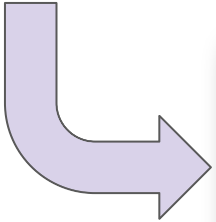
FLOPS: 25 * C^2 * H * W

VGG Design Pattern

(Influential on many upcoming networks)



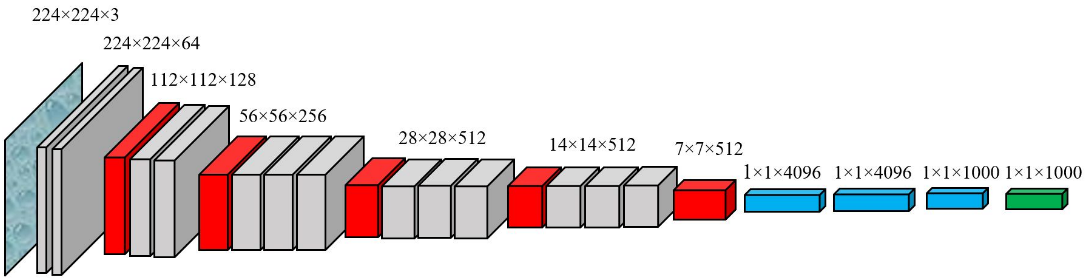
- 3x3 convs, stride 1, pad 1
- **2x2 pool, stride 2**
- **After pool, double channels (until 512)**



- Memory usage halves
  - 2x smaller in {height, width}, 2x larger in depth
- Parameters quadruples
  - Independent of spatial resolution
- FLOPS stays the same!

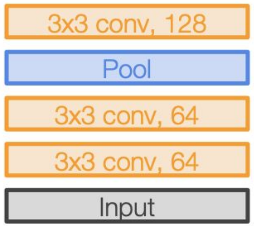
VGG Design Pattern

(Influential on many upcoming networks)



- 3x3 convs, stride 1, pad 1
- **2x2 pool, stride 2**
- **After pool, double channels (until 512)**

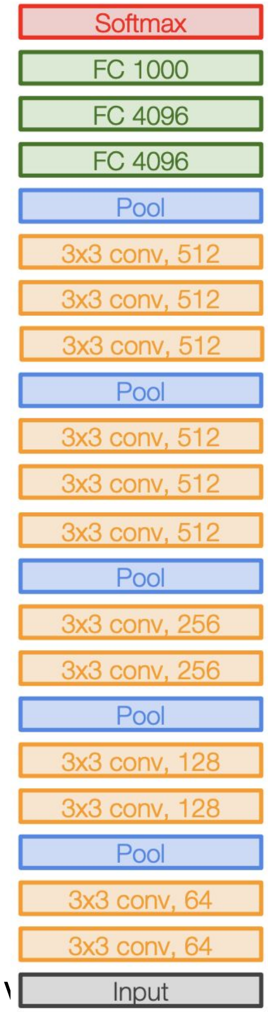
- Memory usage halves
  - 2x smaller in {height, width}, 2x larger in depth
- Parameters quadruples
  - Independent of spatial resolution
- FLOPS stays the same!



224x224

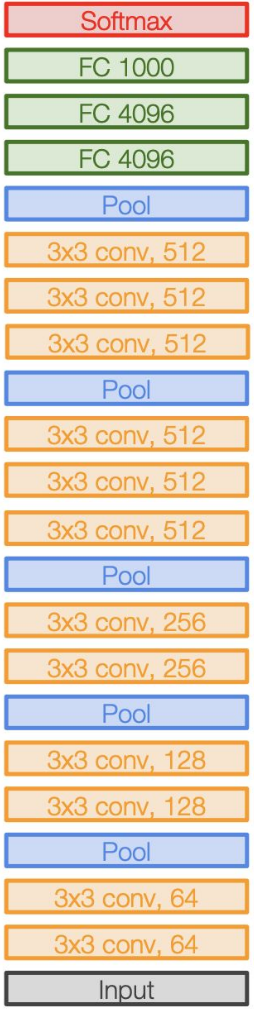
3x3 conv, 128	112x112x128	64*112*3*3 = 64512
Pool	112x112x64	
3x3 conv, 64	224x224x64	64*64*3*3 = 36864
3x3 conv, 64	224x224x64	3*64*3*3 = 1728
Operation	**Output shape**	**# parameters**

Bottom part of VGG



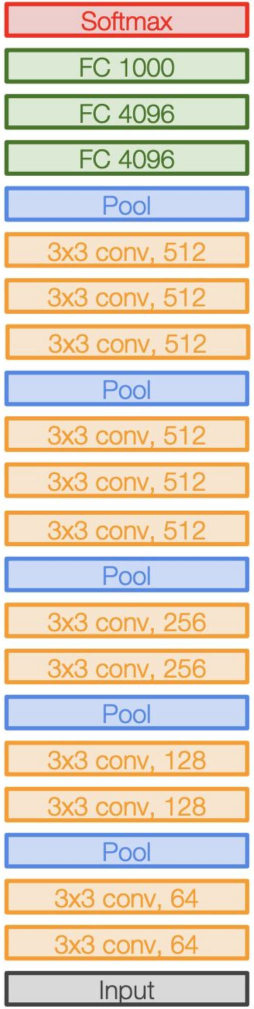
224x224

3x3 conv, 256	56x56x256	294912
Pool	56x56x128	
3x3 conv, 128	112x112x128	147456
3x3 conv, 128	112x112x128	64*112*3*3 = 64512
Pool	112x112x64	
3x3 conv, 64	224x224x64	64*64*3*3 = 36864
3x3 conv, 64	224x224x64	3*64*3*3 = 1728
Operation	**Output shape**	**# parameters**



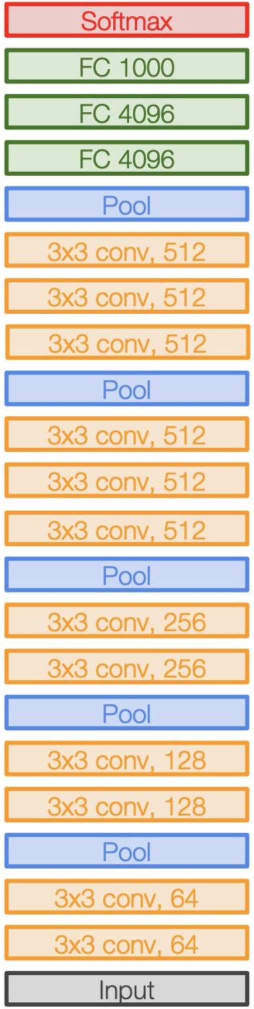
224x224

3x3 conv, 512	28x28x512	1179648
Pool	28x28x256	
3x3 conv, 256	56x56x256	589824
3x3 conv, 256	56x56x256	294912
Pool	56x56x128	
3x3 conv, 128	112x112x128	147456
3x3 conv, 128	112x112x128	64*112*3*3 = 64512
Pool	112x112x64	
3x3 conv, 64	224x224x64	64*64*3*3 = 36864
3x3 conv, 64	224x224x64	3*64*3*3 = 1728
Operation	**Output shape**	**# parameters**



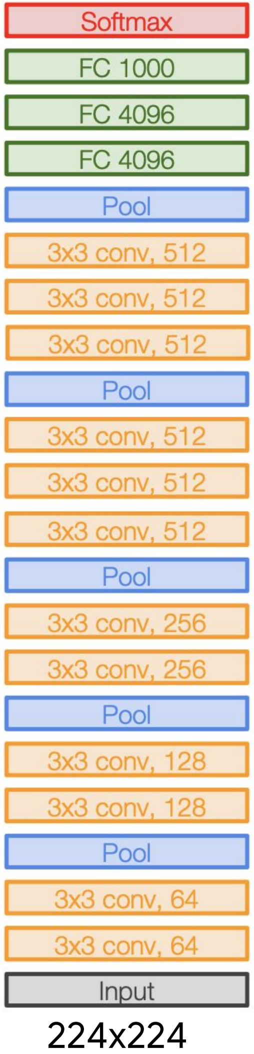
224x224

3x3 conv, 512	14x14x512	2359296
Pool	14x14x512	
3x3 conv, 512	28x28x512	2359296
3x3 conv, 512	28x28x512	2359296
3x3 conv, 512	28x28x512	1179648
Pool	28x28x256	
3x3 conv, 256	56x56x256	589824
3x3 conv, 256	56x56x256	294912
Pool	56x56x128	
3x3 conv, 128	112x112x128	147456
3x3 conv, 128	112x112x128	64*112*3*3 = 64512
Pool	112x112x64	
3x3 conv, 64	224x224x64	64*64*3*3 = 36864
3x3 conv, 64	224x224x64	3*64*3*3 = 1728
Operation	**Output shape**	**# parameters**



224x224

Flatten	25088	
Pool	7x7x512	
3x3 conv, 512	14x14x512	2359296
3x3 conv, 512	14x14x512	2359296
3x3 conv, 512	14x14x512	2359296
Pool	14x14x512	
3x3 conv, 512	28x28x512	2359296
3x3 conv, 512	28x28x512	2359296
3x3 conv, 512	28x28x512	1179648
Pool	28x28x256	
3x3 conv, 256	56x56x256	589824
3x3 conv, 256	56x56x256	294912
Pool	56x56x128	
3x3 conv, 128	112x112x128	147456
3x3 conv, 128	112x112x128	64*112*3*3 = 64512
Pool	112x112x64	
3x3 conv, 64	224x224x64	64*64*3*3 = 36864
3x3 conv, 64	224x224x64	3*64*3*3 = 1728
Operation	**Output shape**	**# parameters**



FC	1000	4096*1000 = 4,096,000
FC	4096	4096*4096 = **16,777,216**
FC	4096	25088*4096 = **102,760,448**
Flatten	25088	
Pool	7x7x512	
3x3 conv, 512	14x14x512	2359296
3x3 conv, 512	14x14x512	2359296
3x3 conv, 512	14x14x512	2359296
Pool	14x14x512	
3x3 conv, 512	28x28x512	2359296
3x3 conv, 512	28x28x512	2359296
3x3 conv, 512	28x28x512	1179648
Pool	28x28x256	
3x3 conv, 256	56x56x256	589824
3x3 conv, 256	56x56x256	294912
Pool	56x56x128	
3x3 conv, 128	112x112x128	147456
3x3 conv, 128	112x112x128	64*112*3*3 = 64512
Pool	112x112x64	
3x3 conv, 64	224x224x64	64*64*3*3 = 36864
3x3 conv, 64	224x224x64	3*64*3*3 = 1728
Operation	**Output shape**	**# parameters**

Even larger FC layers! Largest FC: 25088 -> 4096)

ImageNet experiments

VGG Stronger “single-net” performance than GoogLeNet, but GoogLeNet (next) more efficient

GoogLeNet : Winner of ILSVRC 2014

- Training details similar to AlexNet
- Batch size 256
- 2-3 weeks(!) of training
- 4 GPUs, data parallelism

Method	top-1 val. error (%)	top-5 val. error (%)	top-5 test error (%)
VGG (2 nets, multi-crop & dense eval.)	**23.7**	**6.8**	**6.8**
VGG (1 net, multi-crop & dense eval.)	24.4	7.1	7.0
VGG (ILSVRC submission, 7 nets, dense eval.)	24.7	7.5	7.3
GoogLeNet (Szegedy et al., 2014) (1 net)	-	7.9	
GoogLeNet (Szegedy et al., 2014) (7 nets)	-	**6.7**	
MSRA (He et al., 2014) (11 nets)	-	-	8.1
MSRA (He et al., 2014) (1 net)	27.9	9.1	9.1
Clarifai (Russakovsky et al., 2014) (multiple nets)	-	-	11.7
Clarifai (Russakovsky et al., 2014) (1 net)	-	-	12.5
Zeiler & Fergus (Zeiler & Fergus, 2013) (6 nets)	36.0	14.7	14.8
Zeiler & Fergus (Zeiler & Fergus, 2013) (1 net)	37.5	16.0	16.1
OverFeat (Sermanet et al., 2014) (7 nets)	34.0	13.2	13.6
OverFeat (Sermanet et al., 2014) (1 net)	35.7	14.2	-
Krizhevsky et al. (Krizhevsky et al., 2012) (5 nets)	38.1	16.4	16.4
Krizhevsky et al. (Krizhevsky et al., 2012) (1 net)	40.7	18.2	-

After VGG: Trend is to go even deeper...

But to do so requires computational efficiency.

Next: “Factored” Convolutions -- rewrite convs as a (series or parallel) network of more efficient convs (think of low rank matrix factorizations!).

Examples:

- Sequence of (spatially) smaller convolutional kernels
    - Already saw this a bit with VGG
- Lower dimension then raise again (like low rank decomposition)
- Separable Convolutions

Case Study (2014): GoogLeNet

This CVPR2015 paper is the Open Access version, provided by the Computer Vision Foundation.
The authoritative version of this paper is available in IEEE Xplore.

Going Deeper with Convolutions

Christian Szegedy[1], Wei Liu[2], Yangqing Jia[1], Pierre Sermanet[1], Scott Reed[3],
Dragomir Anguelov[1], Dumitru Erhan[1], Vincent Vanhoucke[1], Andrew Rabinovich[4]
[1]Google Inc. [2]University of North Carolina, Chapel Hill
[3]University of Michigan, Ann Arbor [4]Magic Leap Inc.
[1]{szegedy,jiayq,sermanet,dragomir,dumitru,vanhoucke}@google.com
[2]wliu@cs.unc.edu, [3]reedscott@umich.edu, [4]arabinovich@magicleap.com

Abstract

We propose a deep convolutional neural network architecture codenamed Inception that achieves the new state of the art for classification and detection in the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14). The main hallmark of this architecture is the improved utilization of the computing resources inside the network. By a carefully crafted design, we increased the depth and width of the network while keeping the computational budget constant. To optimize quality, the architectural decisions were based on the Hebbian principle and the intuition of multi-scale processing. One particular incarnation used in our submission for ILSVRC14 is called GoogLeNet, a 22 layers deep network, the quality of which is assessed in the context of classification and detection.

1. Introduction

In the last three years, our object classification and detection capabilities have dramatically improved due to advances in deep learning and convolutional networks [10]. One encouraging news is that most of this progress is not just the result of more powerful hardware, larger datasets and bigger models, but mainly a consequence of new ideas, algorithms and improved network architectures. No new data sources were used, for example, by the top entries in the ILSVRC 2014 competition besides the classification dataset of the same competition for detection purposes. Our GoogLeNet submission to ILSVRC 2014 actually uses 12 times fewer parameters than the winning architecture of Krizhevsky et al [9] from two years ago, while being significantly more accurate. On the object detection front, the biggest gains have not come from naive application of bigger and bigger deep networks, but from the synergy of deep architectures and classical computer vision, like the R-CNN algorithm by Girshick et al [6].

Another notable factor is that with the ongoing traction of mobile and embedded computing, the efficiency of our algorithms – especially their power and memory use – gains importance. It is noteworthy that the considerations leading to the design of the deep architecture presented in this paper included this factor rather than having a sheer fixation on accuracy numbers. For most of the experiments, the models were designed to keep a computational budget of 1.5 billion multiply-adds at inference time, so that the they do not end up to be a purely academic curiosity, but could be put to real world use, even on large datasets, at a reasonable cost.

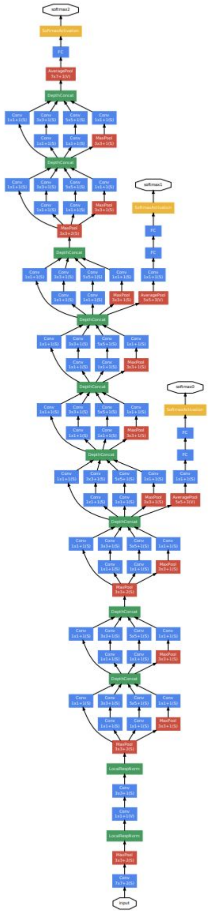
In this paper, we will focus on an efficient deep neural network architecture for computer vision, codenamed Inception, which derives its name from the Network in network paper by Lin et al [12] in conjunction with the famous "we need to go deeper" internet meme [1]. In our case, the word "deep" is used in two different meanings: first of all, in the sense that we introduce a new level of organization in the form of the "Inception module" and also in the more direct sense of increased network depth. In general, one can view the Inception model as a logical culmination of [12] while taking inspiration and guidance from the theoretical work by Arora et al [2]. The benefits of the architecture are experimentally verified on the ILSVRC 2014 classification and detection challenges, where it significantly outperforms the current state of the art.

2. Related Work

Starting with LeNet-5 [10], convolutional neural networks (CNN) have typically had a standard structure – stacked convolutional layers (optionally followed by con-

1

22 Layers



Case Study (2014): GoogLeNet

Going Deeper with Convolutions

Christian Szegedy[1], Wei Liu[2], Yangqing Jia[1], Pierre Sermanet[1], Scott Reed[3],
Dragomir Anguelov[1], Dumitru Erhan[1], Vincent Vanhoucke[1], Andrew Rabinovich[4]
[1]Google Inc. [2]University of North Carolina, Chapel Hill
[3]University of Michigan, Ann Arbor [4]Magic Leap Inc.
[1]{szegedy,jiayq,sermanet,dragomir,dumitru,vanhoucke}@google.com
[2]wliu@cs.unc.edu, [3]reedscott@umich.edu, [4]arabinovich@magicleap.com

Abstract

We propose a deep convolutional neural network architecture codenamed Inception that achieves the new state of the art for classification and detection in the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14). The main hallmark of this architecture is the improved utilization of the computing resources inside the network. By a carefully crafted design, we increased the depth and width of the network while keeping the computational budget constant. To optimize quality, the architectural decisions were based on the Hebbian principle and the intuition of multi-scale processing. One particular incarnation used in our submission for ILSVRC14 is called GoogLeNet, a 22 layers deep network, the quality of which is assessed in the context of classification and detection.

1. Introduction

In the last three years, our object classification and detection capabilities have dramatically improved due to advances in deep learning and convolutional networks [10]. One encouraging news is that most of this progress is not just the result of more powerful hardware, larger datasets and bigger models, but mainly a consequence of new ideas, algorithms and improved network architectures. No new data sources were used, for example, by the top entries in the ILSVRC 2014 competition besides the classification dataset of the same competition for detection purposes. Our GoogLeNet submission to ILSVRC 2014 actually uses 12 times fewer parameters than the winning architecture of Krizhevsky et al [9] from two years ago, while being significantly more accurate. On the object detection front, the biggest gains have not come from naive application of bigger and bigger deep networks, but from the synergy of deep architectures and classical computer vision, like the R-CNN algorithm by Girshick et al [6].

Another notable factor is that with the ongoing traction of mobile and embedded computing, the efficiency of our algorithms – especially their power and memory use – gains importance. It is noteworthy that the considerations leading to the design of the deep architecture presented in this paper included this factor rather than having a sheer fixation on accuracy numbers. For most of the experiments, the models were designed to keep a computational budget of 1.5 billion multiply-adds at inference time, so that the they do not end up to be a purely academic curiosity, but could be put to real world use, even on large datasets, at a reasonable cost.

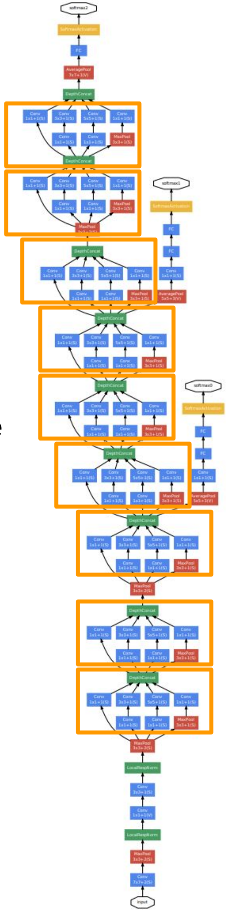
In this paper, we will focus on an efficient deep neural network architecture for computer vision, codenamed Inception, which derives its name from the Network in network paper by Lin et al [12] in conjunction with the famous "we need to go deeper" internet meme [1]. In our case, the word "deep" is used in two different meanings: first of all, in the sense that we introduce a new level of organization in the form of the "Inception module" and also in the more direct sense of increased network depth. In general, one can view the Inception model as a logical culmination of [12] while taking inspiration and guidance from the theoretical work by Arora et al [2]. The benefits of the architecture are experimentally verified on the ILSVRC 2014 classification and detection challenges, where it significantly outperforms the current state of the art.

2. Related Work

Starting with LeNet-5 [10], convolutional neural networks (CNN) have typically had a standard structure – stacked convolutional layers (optionally followed by con-

Inception Blocks

Repeated Local Structure



Case Study (2014): GoogLeNet

This CVPR2015 paper is the Open Access version, provided by the Computer Vision Foundation.
The authoritative version of this paper is available in IEEE Xplore.

Going Deeper with Convolutions

Christian Szegedy[1], Wei Liu[2], Yangqing Jia[1], Pierre Sermanet[1], Scott Reed[3],
Dragomir Anguelov[1], Dumitru Erhan[1], Vincent Vanhoucke[1], Andrew Rabinovich[4]
[1]Google Inc. [2]University of North Carolina, Chapel Hill
[3]University of Michigan, Ann Arbor [4]Magic Leap Inc.
[1]{szegedy,jiayq,sermanet,dragomir,dumitru,vanhoucke}@google.com
[2]wliu@cs.unc.edu, [3]reedscott@umich.edu, [4]arabinovich@magicleap.com

Abstract

We propose a deep convolutional neural network architecture codenamed Inception that achieves the new state of the art for classification and detection in the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14). The main hallmark of this architecture is the improved utilization of the computing resources inside the network. By a carefully crafted design, we increased the depth and width of the network while keeping the computational budget constant. To optimize quality, the architectural decisions were based on the Hebbian principle and the intuition of multi-scale processing. One particular incarnation used in our submission for ILSVRC14 is called GoogLeNet, a 22 layers deep network, the quality of which is assessed in the context of classification and detection.

1. Introduction

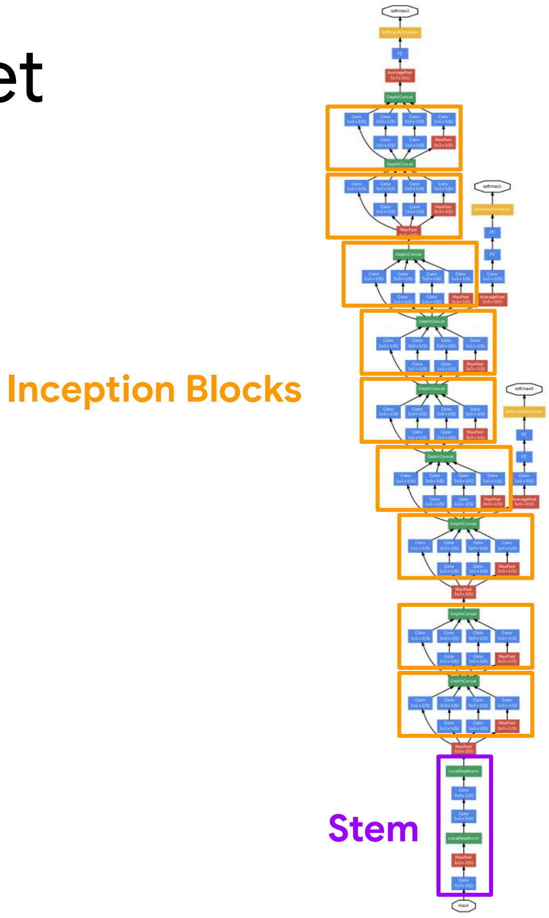
In the last three years, our object classification and detection capabilities have dramatically improved due to advances in deep learning and convolutional networks [10]. One encouraging news is that most of this progress is not just the result of more powerful hardware, larger datasets and bigger models, but mainly a consequence of new ideas, algorithms and improved network architectures. No new data sources were used, for example, by the top entries in the ILSVRC 2014 competition besides the classification dataset of the same competition for detection purposes. Our GoogLeNet submission to ILSVRC 2014 actually uses 12 times fewer parameters than the winning architecture of Krizhevsky et al [9] from two years ago, while being significantly more accurate. On the object detection front, the biggest gains have not come from naive application of bigger and bigger deep networks, but from the synergy of deep architectures and classical computer vision, like the R-CNN algorithm by Girshick et al [6].

Another notable factor is that with the ongoing traction of mobile and embedded computing, the efficiency of our algorithms – especially their power and memory use – gains importance. It is noteworthy that the considerations leading to the design of the deep architecture presented in this paper included this factor rather than having a sheer fixation on accuracy numbers. For most of the experiments, the models were designed to keep a computational budget of 1.5 billion multiply-adds at inference time, so that the they do not end up to be a purely academic curiosity, but could be put to real world use, even on large datasets, at a reasonable cost.

In this paper, we will focus on an efficient deep neural network architecture for computer vision, codenamed Inception, which derives its name from the Network in network paper by Lin et al [12] in conjunction with the famous "we need to go deeper" internet meme [1]. In our case, the word "deep" is used in two different meanings: first of all, in the sense that we introduce a new level of organization in the form of the "Inception module" and also in the more direct sense of increased network depth. In general, one can view the Inception model as a logical culmination of [12] while taking inspiration and guidance from the theoretical work by Arora et al [2]. The benefits of the architecture are experimentally verified on the ILSVRC 2014 classification and detection challenges, where it significantly outperforms the current state of the art.

2. Related Work

Starting with LeNet-5 [10], convolutional neural networks (CNN) have typically had a standard structure – stacked convolutional layers (optionally followed by con-



Case Study (2014): GoogLeNet

Going Deeper with Convolutions

Christian Szegedy[1], Wei Liu[2], Yangqing Jia[1], Pierre Sermanet[1], Scott Reed[3],
Dragomir Anguelov[1], Dumitru Erhan[1], Vincent Vanhoucke[1], Andrew Rabinovich[4]
[1]Google Inc. [2]University of North Carolina, Chapel Hill
[3]University of Michigan, Ann Arbor [4]Magic Leap Inc.
[1]{szegedy,jiayq,sermanet,dragomir,dumitru,vanhoucke}@google.com
[2]wliu@cs.unc.edu, [3]reedscott@umich.edu, [4]arabinovich@magicleap.com

Abstract

We propose a deep convolutional neural network architecture codenamed Inception that achieves the new state of the art for classification and detection in the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14). The main hallmark of this architecture is the improved utilization of the computing resources inside the network. By a carefully crafted design, we increased the depth and width of the network while keeping the computational budget constant. To optimize quality, the architectural decisions were based on the Hebbian principle and the intuition of multi-scale processing. One particular incarnation used in our submission for ILSVRC14 is called GoogLeNet, a 22 layers deep network, the quality of which is assessed in the context of classification and detection.

1. Introduction

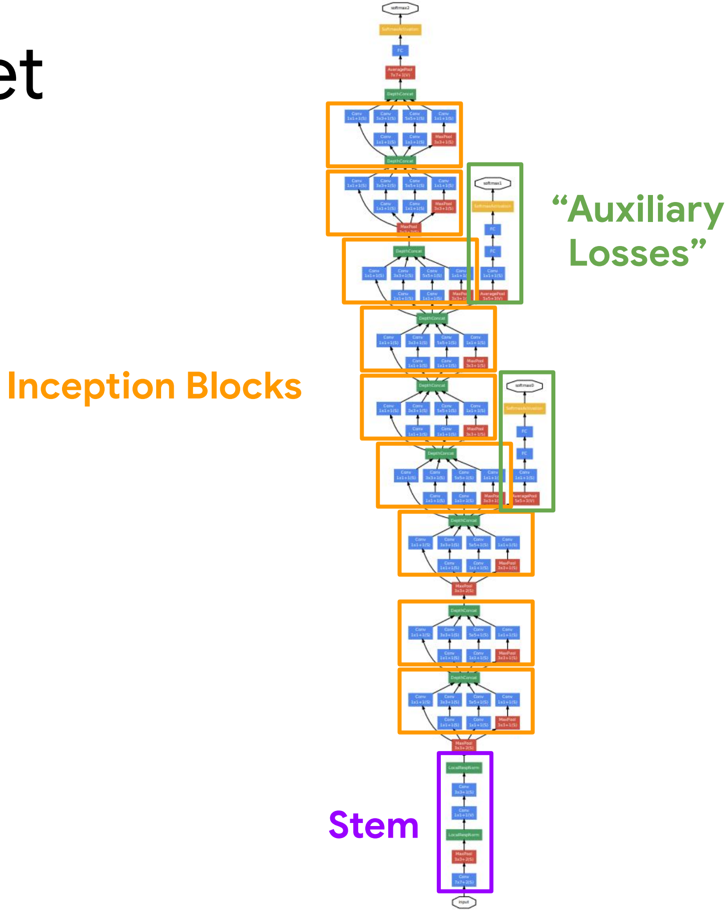
In the last three years, our object classification and detection capabilities have dramatically improved due to advances in deep learning and convolutional networks [10]. One encouraging news is that most of this progress is not just the result of more powerful hardware, larger datasets and bigger models, but mainly a consequence of new ideas, algorithms and improved network architectures. No new data sources were used, for example, by the top entries in the ILSVRC 2014 competition besides the classification dataset of the same competition for detection purposes. Our GoogLeNet submission to ILSVRC 2014 actually uses 12 times fewer parameters than the winning architecture of Krizhevsky et al [9] from two years ago, while being significantly more accurate. On the object detection front, the biggest gains have not come from naive application of bigger and bigger deep networks, but from the synergy of deep architectures and classical computer vision, like the R-CNN algorithm by Girshick et al [6].

Another notable factor is that with the ongoing traction of mobile and embedded computing, the efficiency of our algorithms – especially their power and memory use – gains importance. It is noteworthy that the considerations leading to the design of the deep architecture presented in this paper included this factor rather than having a sheer fixation on accuracy numbers. For most of the experiments, the models were designed to keep a computational budget of 1.5 billion multiply-adds at inference time, so that the they do not end up to be a purely academic curiosity, but could be put to real world use, even on large datasets, at a reasonable cost.

In this paper, we will focus on an efficient deep neural network architecture for computer vision, codenamed Inception, which derives its name from the Network in network paper by Lin et al [12] in conjunction with the famous "we need to go deeper" internet meme [1]. In our case, the word "deep" is used in two different meanings: first of all, in the sense that we introduce a new level of organization in the form of the "Inception module" and also in the more direct sense of increased network depth. In general, one can view the Inception model as a logical culmination of [12] while taking inspiration and guidance from the theoretical work by Arora et al [2]. The benefits of the architecture are experimentally verified on the ILSVRC 2014 classification and detection challenges, where it significantly outperforms the current state of the art.

2. Related Work

Starting with LeNet-5 [10], convolutional neural networks (CNN) have typically had a standard structure – stacked convolutional layers (optionally followed by con-



Case Study (2014): GoogLeNet

Going Deeper with Convolutions

Christian Szegedy[1], Wei Liu[2], Yangqing Jia[1], Pierre Sermanet[1], Scott Reed[3],
Dragomir Anguelov[1], Dumitru Erhan[1], Vincent Vanhoucke[1], Andrew Rabinovich[4]
[1]Google Inc. [2]University of North Carolina, Chapel Hill
[3]University of Michigan, Ann Arbor [4]Magic Leap Inc.
[1]{szegedy,jiayq,sermanet,dragomir,dumitru,vanhoucke}@google.com
[2]wliu@cs.unc.edu, [3]reedscott@umich.edu, [4]arabinovich@magicleap.com

Abstract

We propose a deep convolutional neural network architecture codenamed Inception that achieves the new state of the art for classification and detection in the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14). The main hallmark of this architecture is the improved utilization of the computing resources inside the network. By a carefully crafted design, we increased the depth and width of the network while keeping the computational budget constant. To optimize quality, the architectural decisions were based on the Hebbian principle and the intuition of multi-scale processing. One particular incarnation used in our submission for ILSVRC14 is called GoogLeNet, a 22 layers deep network, the quality of which is assessed in the context of classification and detection.

1. Introduction

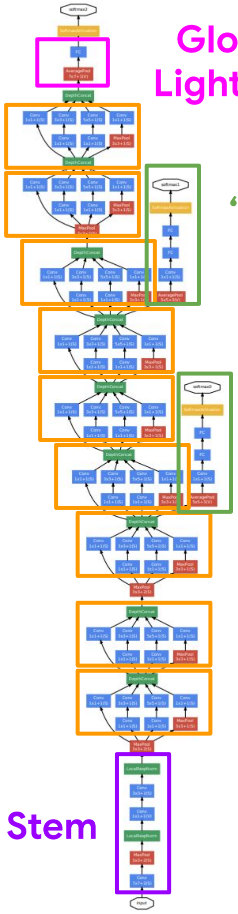
In the last three years, our object classification and detection capabilities have dramatically improved due to advances in deep learning and convolutional networks [10]. One encouraging news is that most of this progress is not just the result of more powerful hardware, larger datasets and bigger models, but mainly a consequence of new ideas, algorithms and improved network architectures. No new data sources were used, for example, by the top entries in the ILSVRC 2014 competition besides the classification dataset of the same competition for detection purposes. Our GoogLeNet submission to ILSVRC 2014 actually uses 12 times fewer parameters than the winning architecture of Krizhevsky et al [9] from two years ago, while being significantly more accurate. On the object detection front, the biggest gains have not come from naive application of bigger and bigger deep networks, but from the synergy of deep architectures and classical computer vision, like the R-CNN algorithm by Girshick et al [6].

Another notable factor is that with the ongoing traction of mobile and embedded computing, the efficiency of our algorithms – especially their power and memory use – gains importance. It is noteworthy that the considerations leading to the design of the deep architecture presented in this paper included this factor rather than having a sheer fixation on accuracy numbers. For most of the experiments, the models were designed to keep a computational budget of 1.5 billion multiply-adds at inference time, so that the they do not end up to be a purely academic curiosity, but could be put to real world use, even on large datasets, at a reasonable cost.

In this paper, we will focus on an efficient deep neural network architecture for computer vision, codenamed Inception, which derives its name from the Network in network paper by Lin et al [12] in conjunction with the famous "we need to go deeper" internet meme [1]. In our case, the word "deep" is used in two different meanings: first of all, in the sense that we introduce a new level of organization in the form of the "Inception module" and also in the more direct sense of increased network depth. In general, one can view the Inception model as a logical culmination of [12] while taking inspiration and guidance from the theoretical work by Arora et al [2]. The benefits of the architecture are experimentally verified on the ILSVRC 2014 classification and detection challenges, where it significantly outperforms the current state of the art.

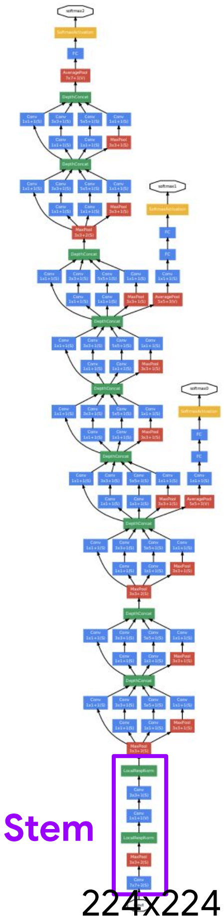
2. Related Work

Starting with LeNet-5 [10], convolutional neural networks (CNN) have typically had a standard structure – stacked convolutional layers (optionally followed by con-



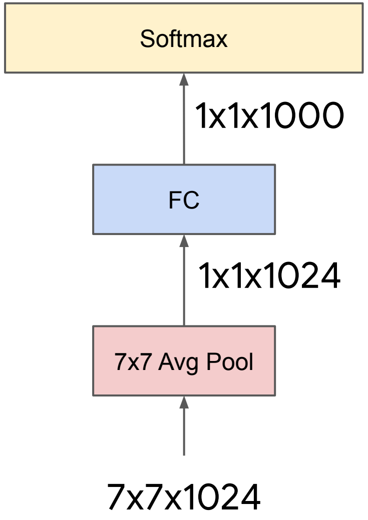
GoogLeNet **Stem**

3x3 Conv	192	2	28x28x192
3x3 Conv	192	1	56x56x192
3x3 Pool		2	56x56x64
7x7 Conv	64	2	112x112x64
Operation	**# filters**	**stride**	**Output shape**

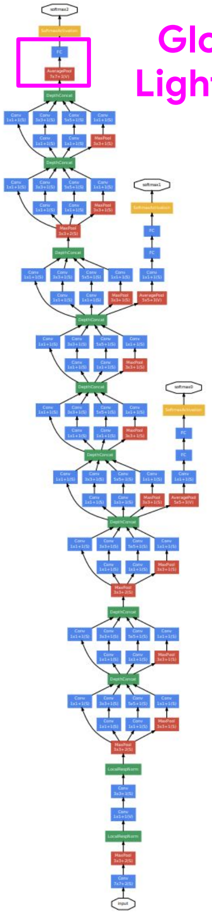


Aggressively reduce resolution in early layers (224x224 to 28x28 in first 4 layers) --- we will see later networks also do this

Global Pool + Lightweight FC

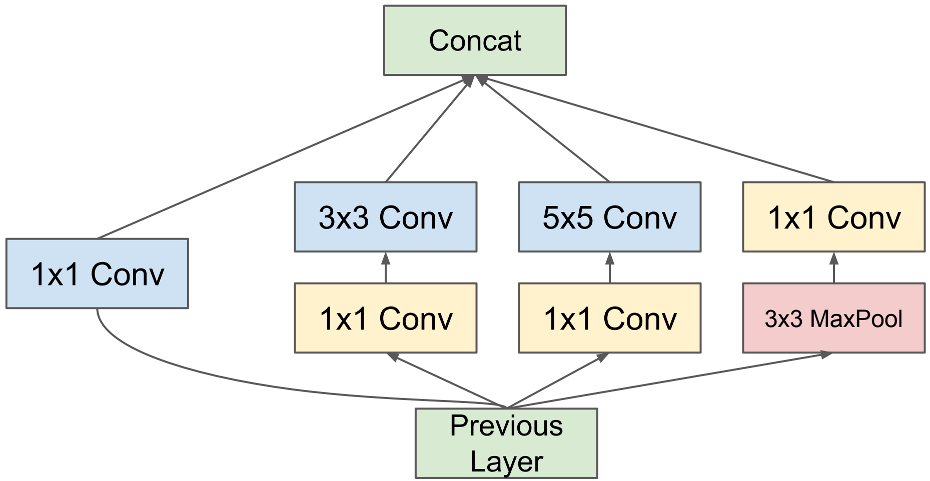


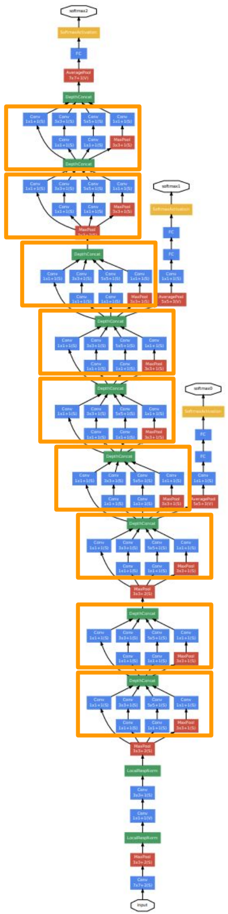
1024x1000 FC
vs
VGG's largest 25088x4096 FC
(*~100x smaller!*)



Global pool +
Lightweight FC

Inception Blocks



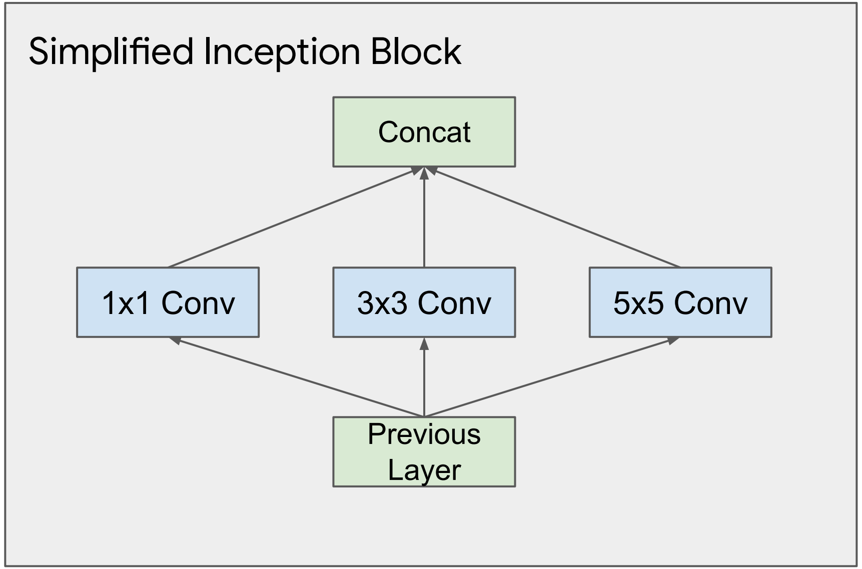


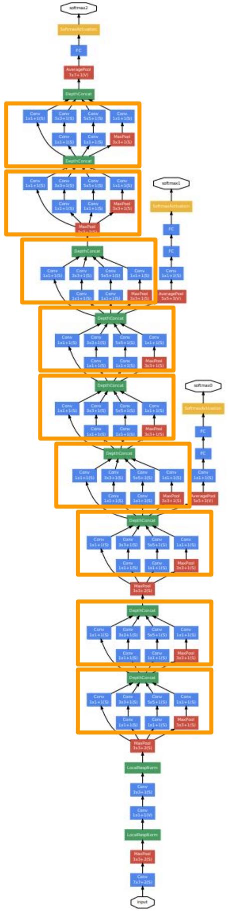
Two tricks:

- Parallel convolutions paths
- Bottleneck layers

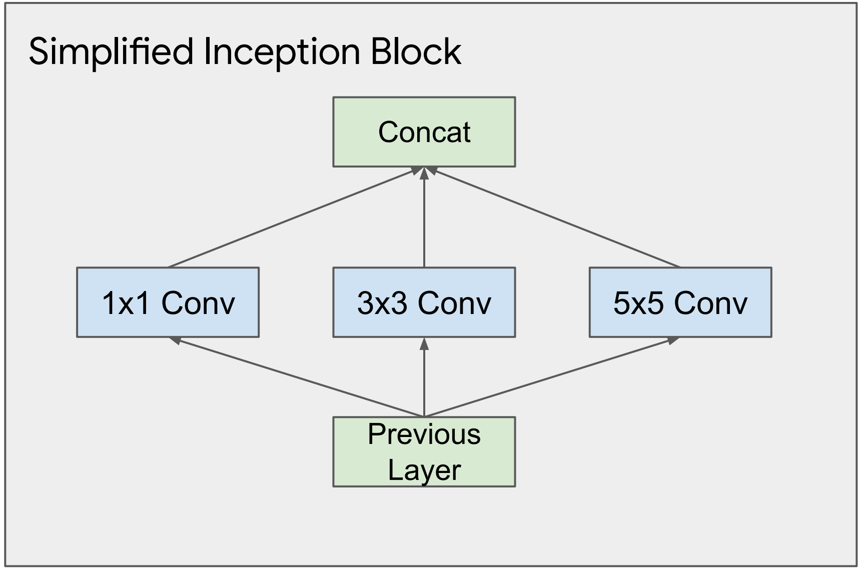
To understand these tricks, let's look at some simplifications

Inception Blocks

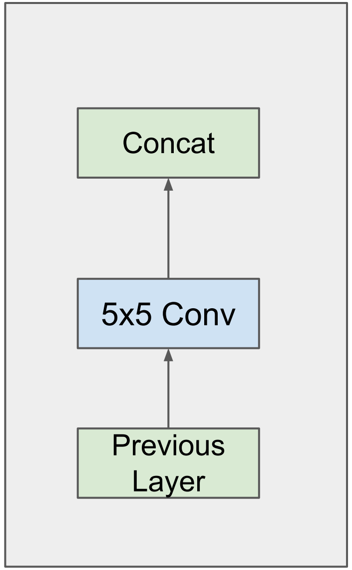


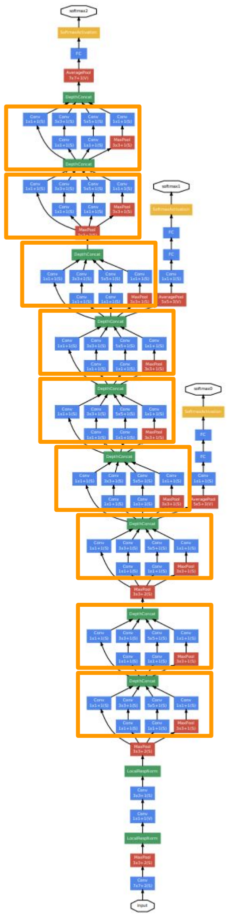


Inception Blocks



vs

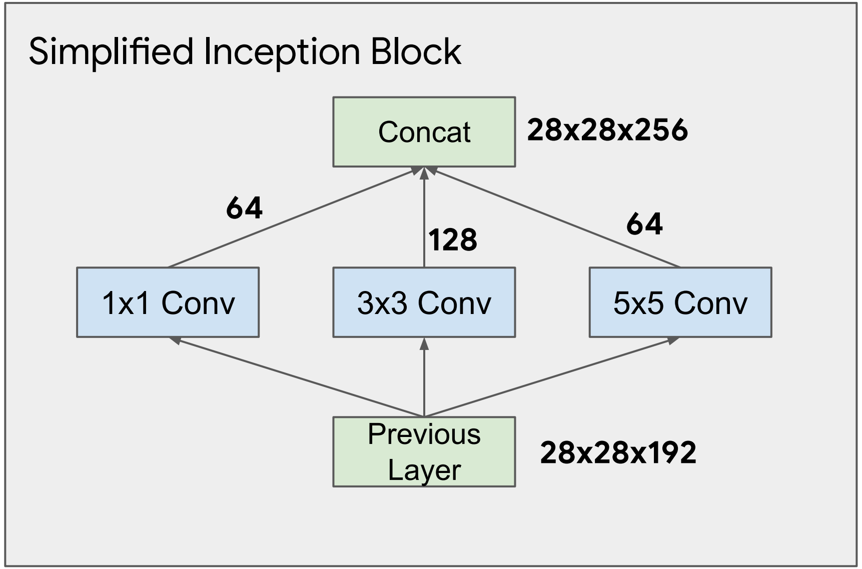




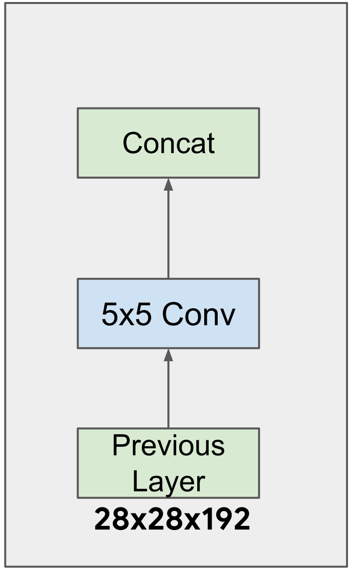
Same receptive field as 5x5: think of replacing 5x5 conv with a "mini-network" with same receptive field

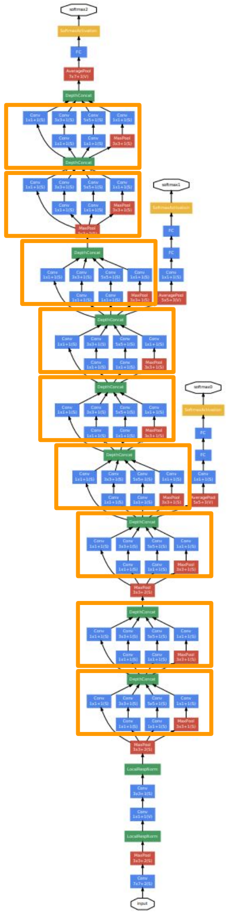
- But in this "mini-network", not all channels of output need to depend on full extent of receptive field

Inception Blocks



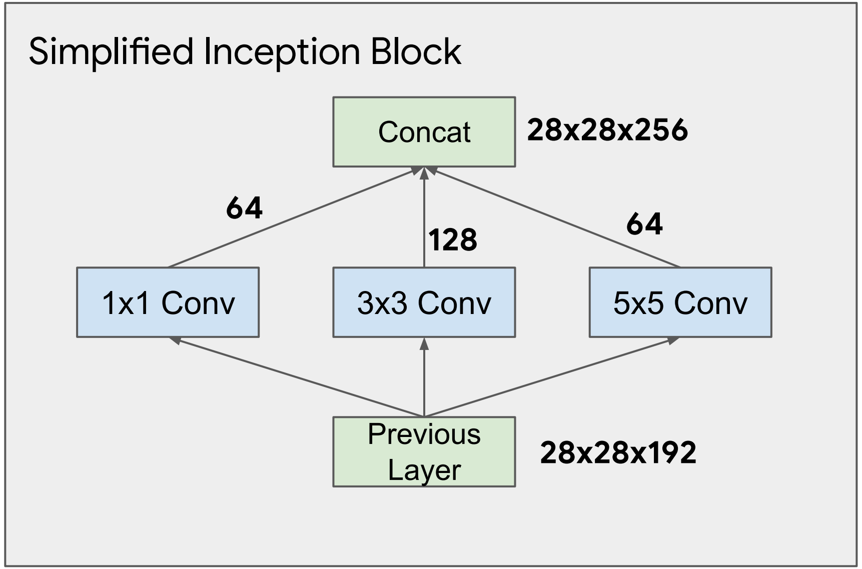
vs



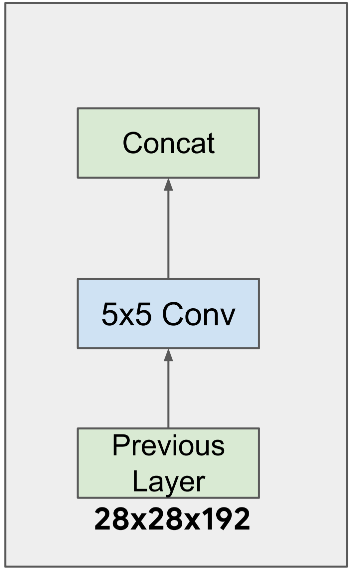


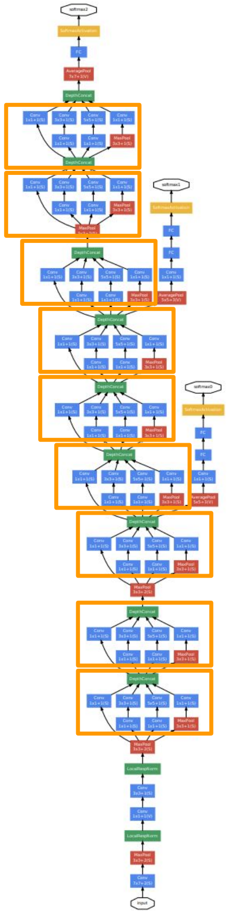
This mini-network (our Inception Block) ends up being more efficient --- let's verify this by counting parameters/ops

Inception Blocks



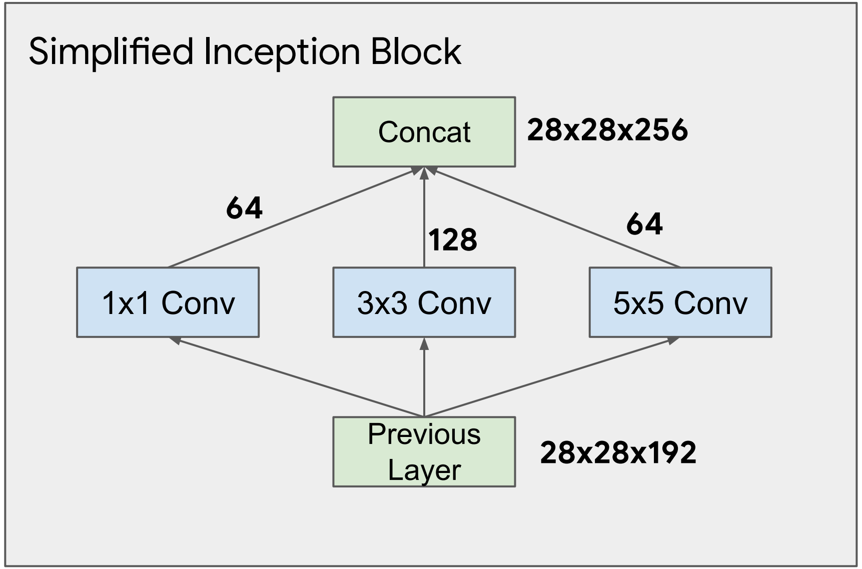
vs



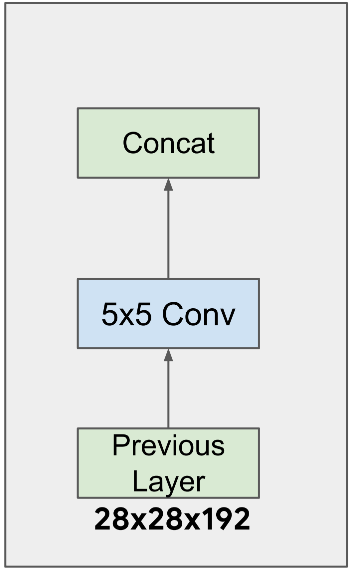


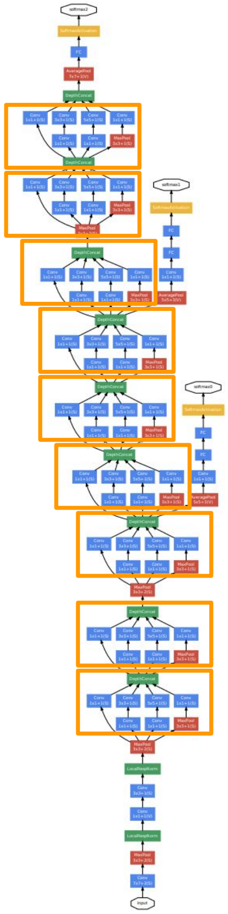
	1x1	3x3	5x5	Total
Params	192x64	9x192x128	25x192x64	
FLOPS				

Inception Blocks



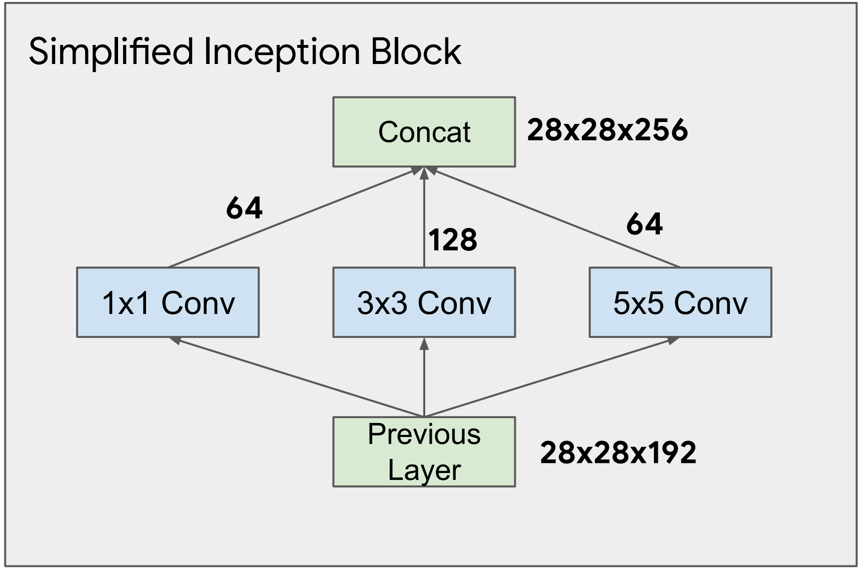
vs



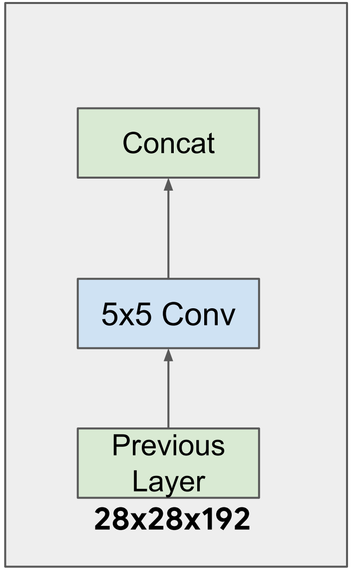


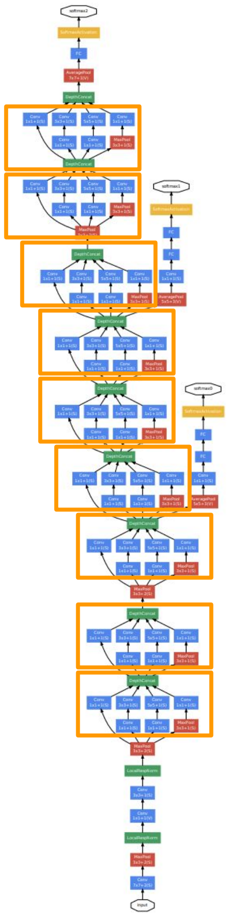
	1x1	3x3	5x5	Total
Params	192x64	9x192x128	25x192x64	
FLOPS	28x28x192x64	9x28x28x192x128	25x28x28x192x64	

Inception Blocks



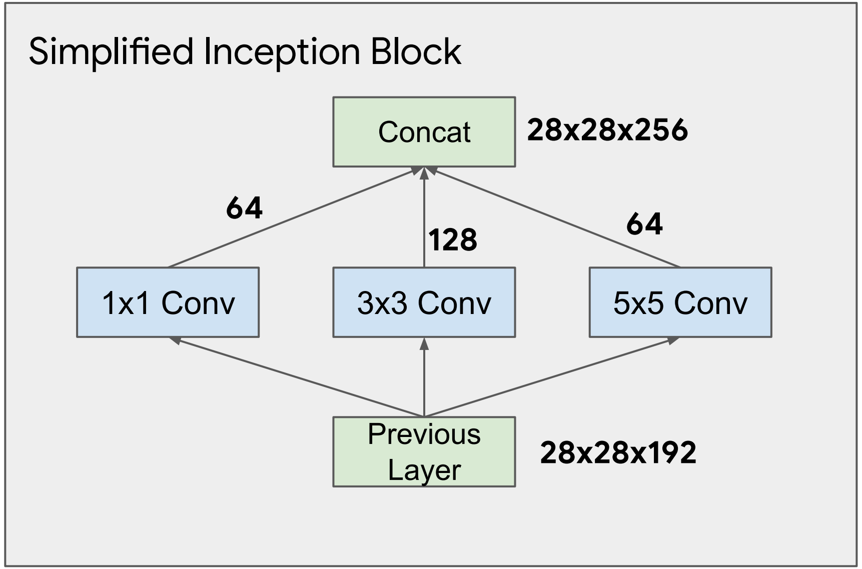
vs



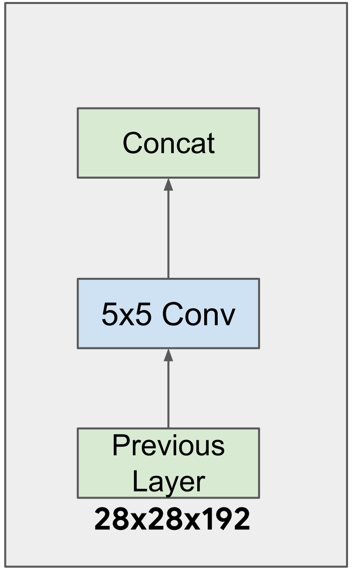


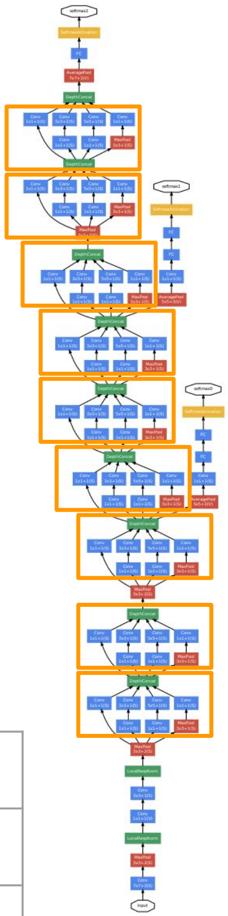
	1x1	3x3	5x5	Total
Params	192x64	9x192x128	25x192x64	540K
FLOPS	28x28x192x64	9x28x28x192x128	25x28x28x192x64	423M

Inception Blocks



vs



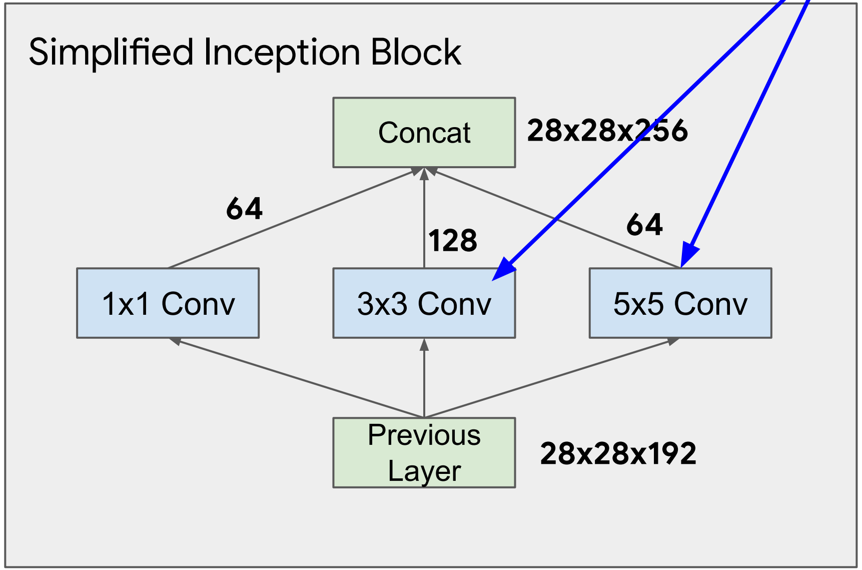


	1x1	3x3	5x5	Total
Params	192x64	9x192x128	25x192x64	540K
FLOPS	28x28x192x64	9x28x28x192x128	25x28x28x192x64	423M

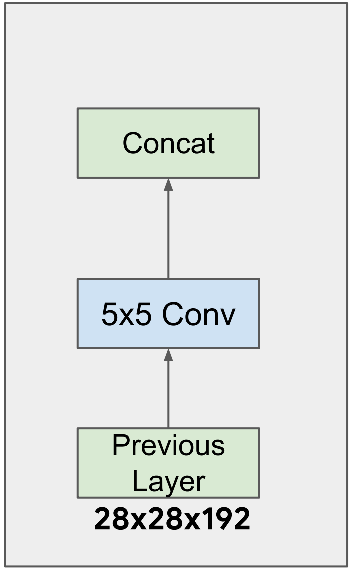
	5x5	Total
Params	25*192x256	1.2M
FLOPS	25*28x28x192x256	963M

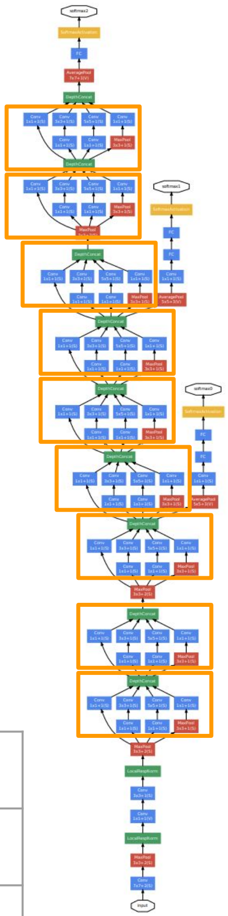
Inception Blocks

Expensive branches: ~9x, ~12x FLOPS of 1x1 branch



vs

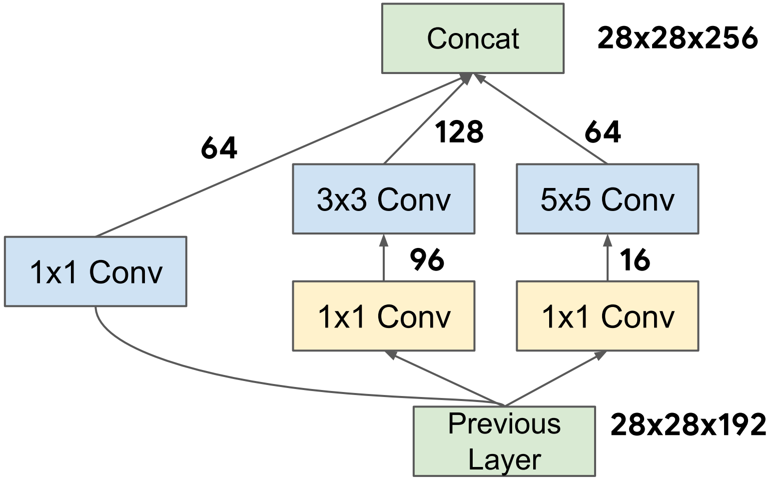




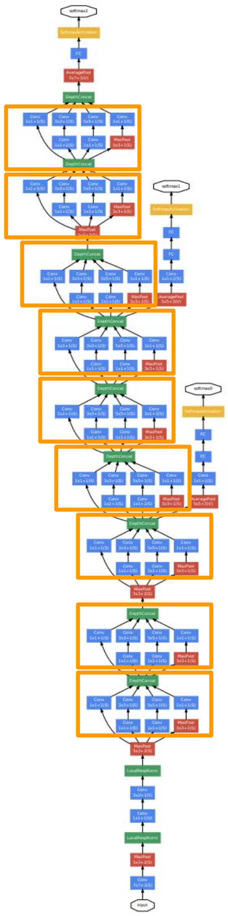
	1x1	3x3	5x5	Total
Params	192x64	9x192x128	25x192x64	540K
FLOPS	28x28x192x64	9x28x28x192x128	25x28x28x192x64	423M

	5x5	Total
Params	25*192x256	1.2M
FLOPS	25*28x28x192x256	963M

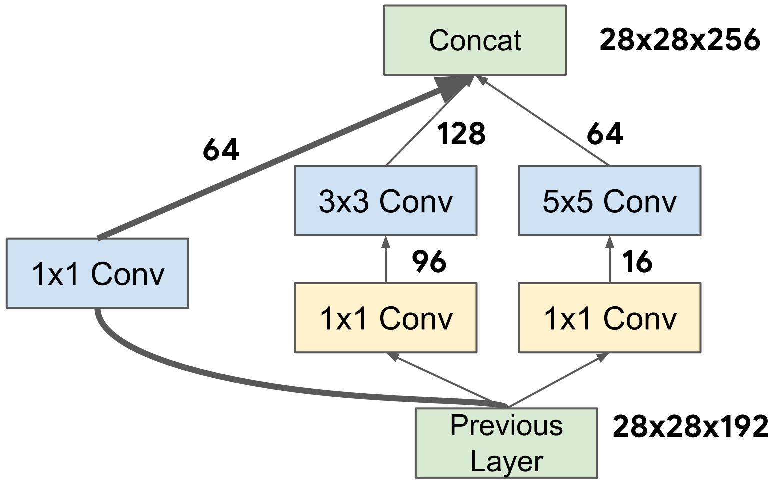
Inception Blocks - “Bottleneck Trick”



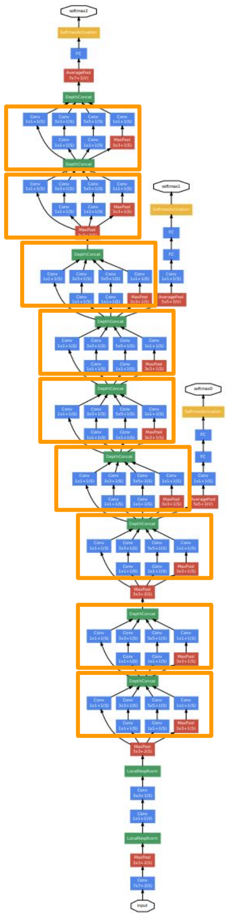
Idea: Reduce dimensions prior to expensive convolutions (to 96 and 16 dimensions, resp)



Inception Blocks - “Bottleneck Trick”

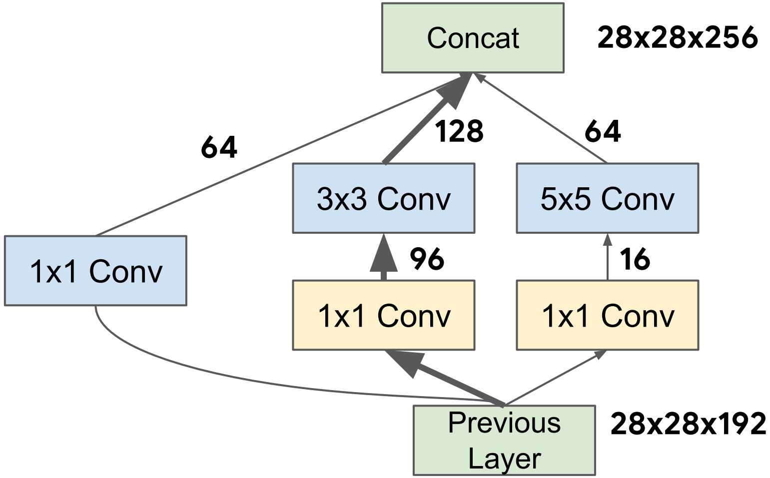


Idea: Reduce dimensions prior to expensive convolutions (to 96 and 16 dimensions, resp)

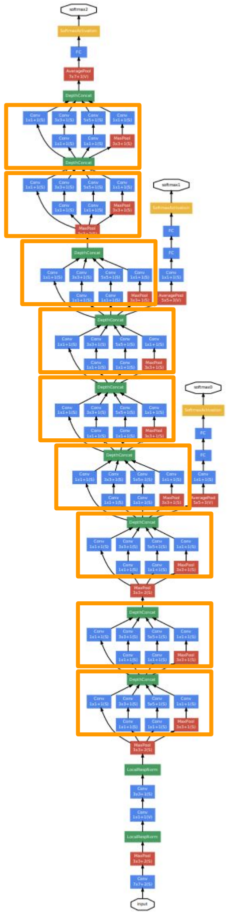


	1x1	3x3	5x5	Total
Params	192x64			
FLOPS	28x28x192x64			

Inception Blocks - “Bottleneck Trick”

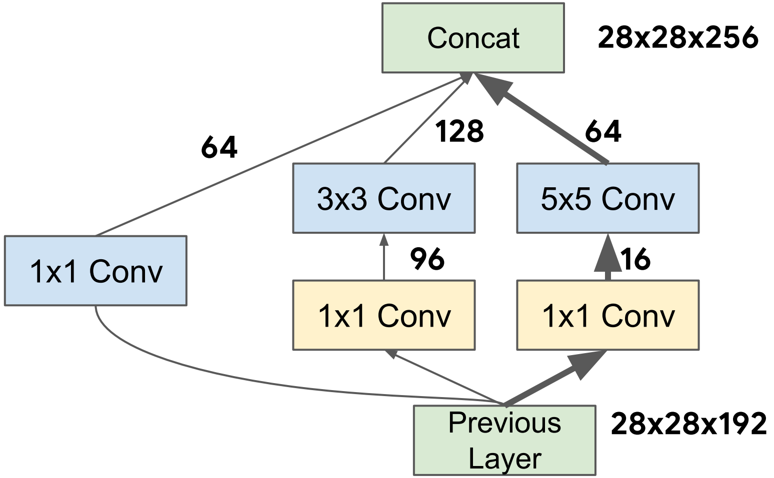


Idea: Reduce dimensions prior to expensive convolutions (to 96 and 16 dimensions, resp)

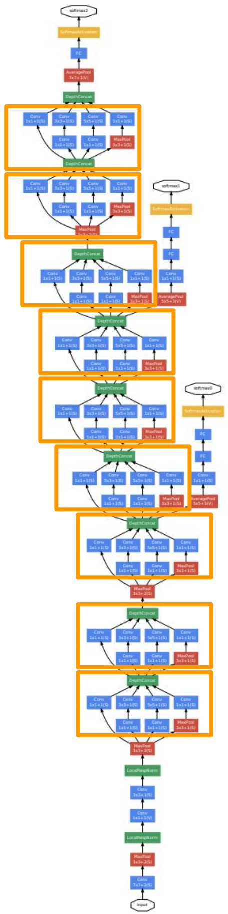


	1x1	3x3	5x5	Total
Params	192x64	192x96+ 9x96x128		
FLOPS	28x28x192x64	28x28x192x96 + 9x28x28x96x128		

Inception Blocks - “Bottleneck Trick”

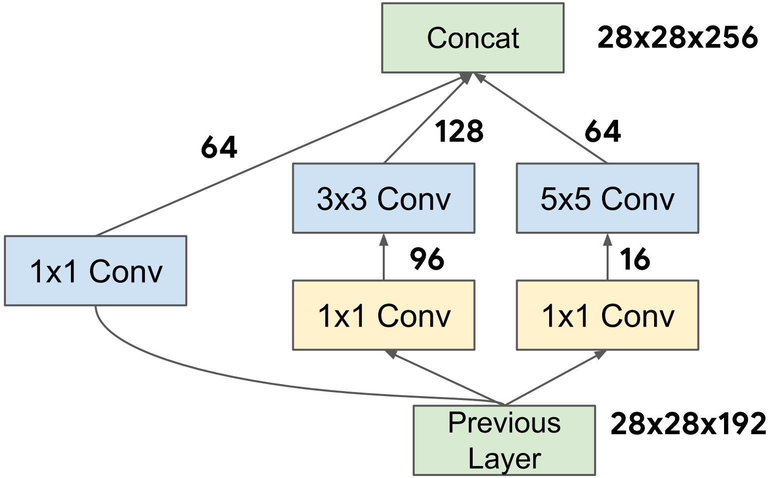


Idea: Reduce dimensions prior to expensive convolutions (to 96 and 16 dimensions, resp)

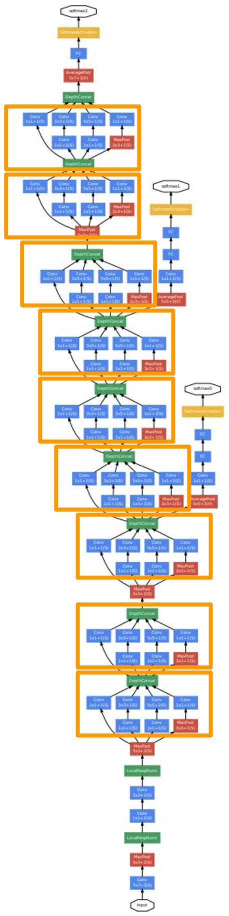


	1x1	3x3	5x5	Total
Params	192x64	192x96+ 9x96x128	192x16 + 25x16x64	
FLOPS	28x28x192x64	28x28x192x96 + 9x28x28x96x128	28x28x192x16 + 25x28x28x16x64	

Inception Blocks - “Bottleneck Trick”



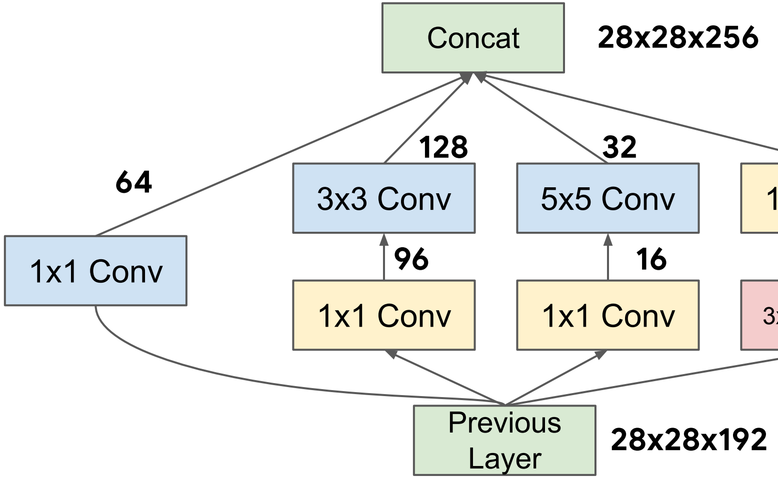
Idea: Reduce dimensions prior to expensive convolutions (to 96 and 16 dimensions, resp)

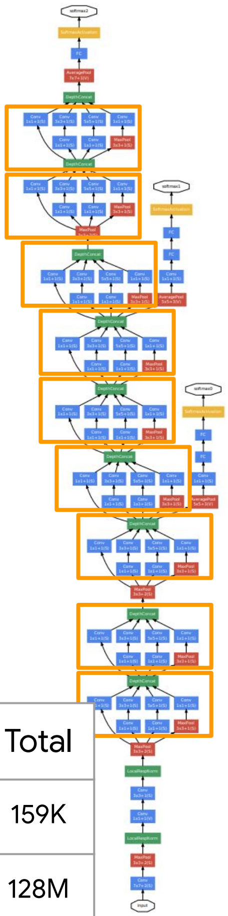


	1x1	3x3	5x5	Total
Params	192x64	192x96+ 9x96x128	192x16 + 25x16x64	170K
FLOPS	28x28x192x64	28x28x192x96 + 9x28x28x96x128	28x28x192x16 + 25x28x28x16x64	133M

Inception Blocks

Add pooling layer "*since pooling operations have been essential for the success of current convolutional networks*"
Note: (we will see pooling operators play a reduced role in later networks)



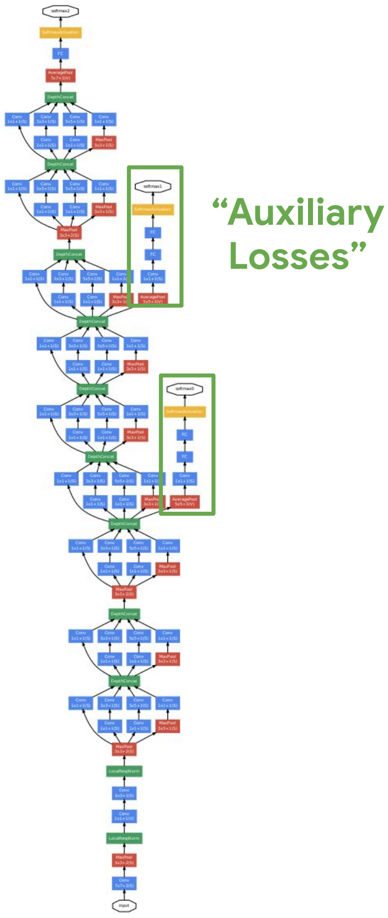


	1x1	3x3	5x5	Pool	Total
Params	192x64	192x96+ 9x96x128	192x16 + 25x16x32	192x32	159K
FLOPS	28x28x192x64	28x28x192x96 + 9x28x28x96x128	28x28x192x16 + 25x28x28x16x32	9*28*28*192+ 28x28x192*32	128M

Auxiliary Losses

- Vanishing gradients a big problem in deeper nets
- Idea:
  - Training time: Add auxiliary classification layers at training time to provide a stronger gradient signal to early layers
  - Test time: discard additional layers

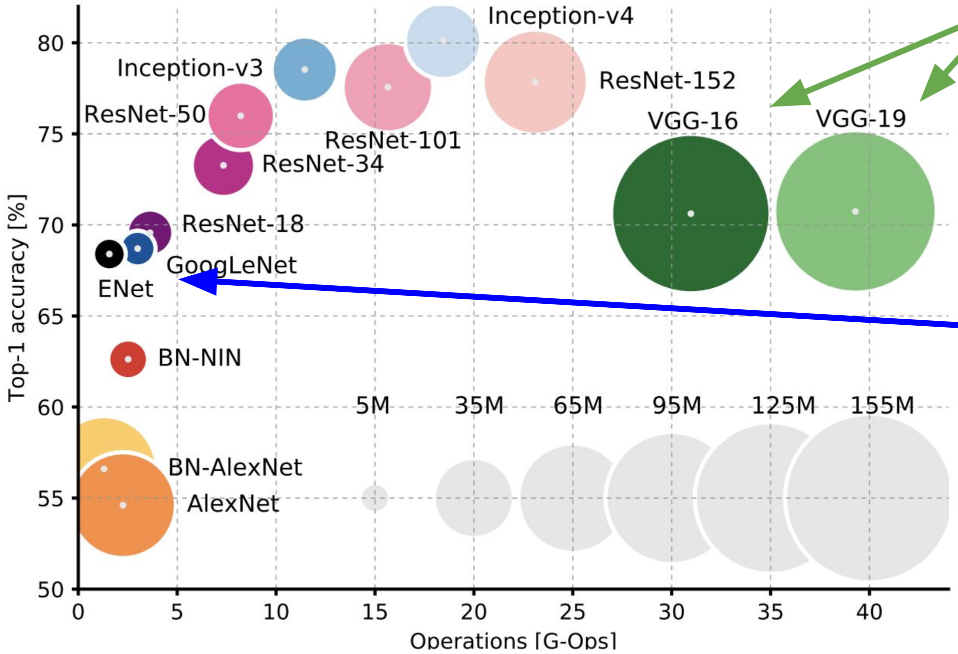
- Later inventions provide better solutions to vanishing gradient:
  - Batch norm
  - Residual connections
- Some papers still use these auxiliary losses



Speed/Accuracy balance

VGG is *yuuuge*; slightly better on Imagenet than GoogLeNet

More accurate

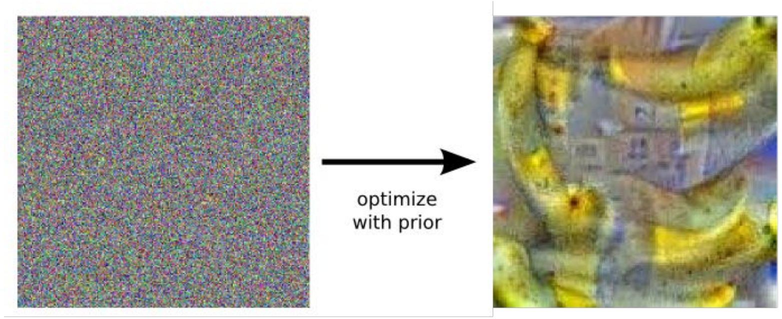


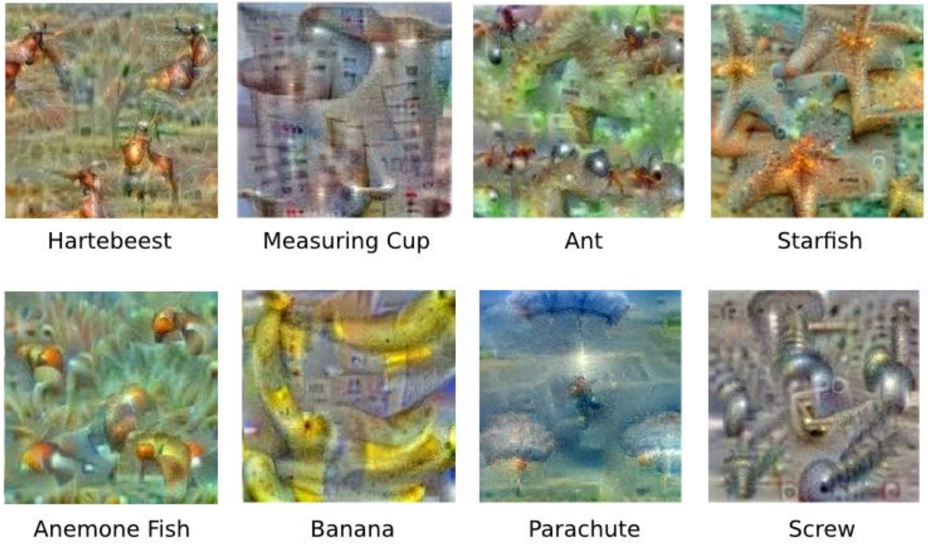
GoogLeNet very lightweight

Slower

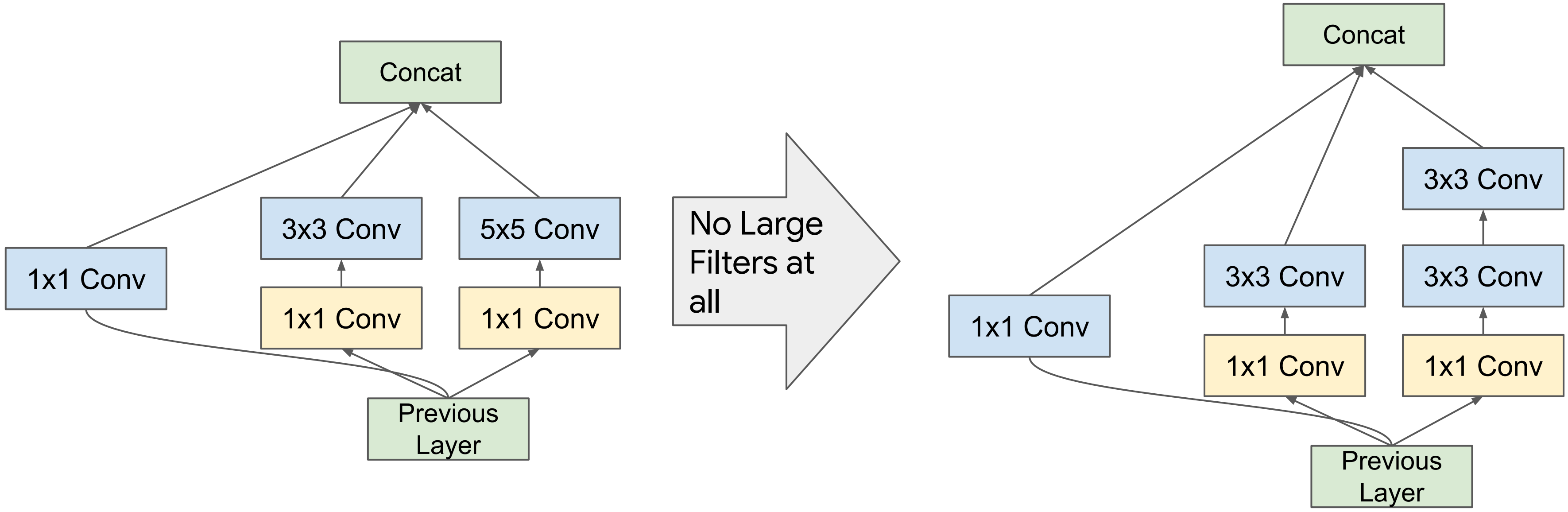
An Analysis of Deep Neural Network Models for Practical Applications by Canziani et al

Neural Network Generated Art with Inception



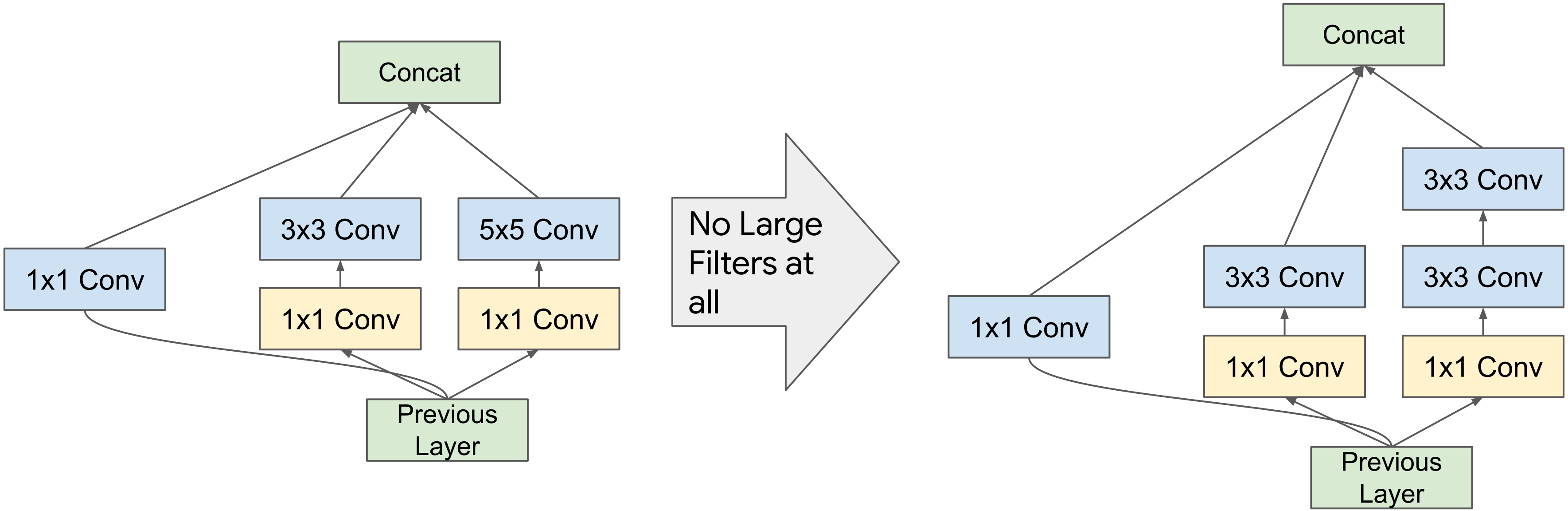


Variations on a Theme: Let’s play the “VGG” Trick



Rethinking the Inception Architecture for Computer Vision by Szegedy et al

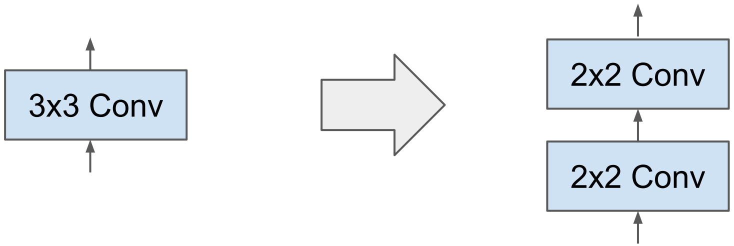
Variations on a Theme: Let's play the "VGG" Trick



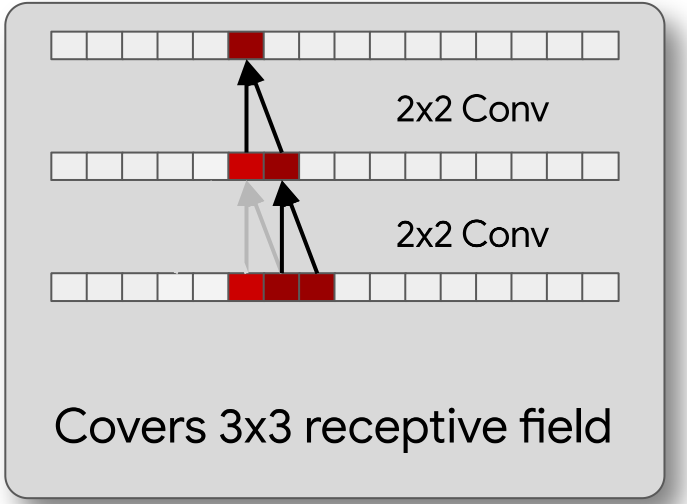
Can we go smaller than 3x3?

Rethinking the Inception Architecture for Computer Vision by Szegedy et al

Spatial Factorization into (non-square) asymmetric convolutions



A little smaller

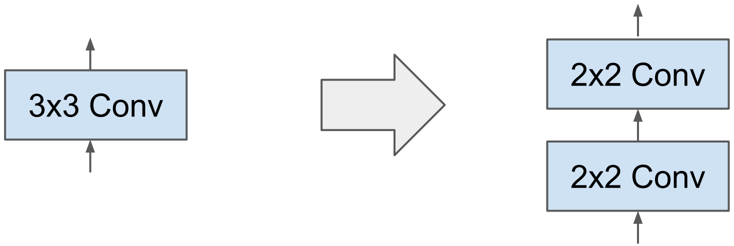


Params: 9*C^2
FLOPS: 9*C^2*H*W

Params: 2*4*C^2 = 8C^2
FLOPS: 2*4*C^2*H*W = 8*C^2*H*W

Rethinking the Inception Architecture for Computer Vision by Szegedy et al

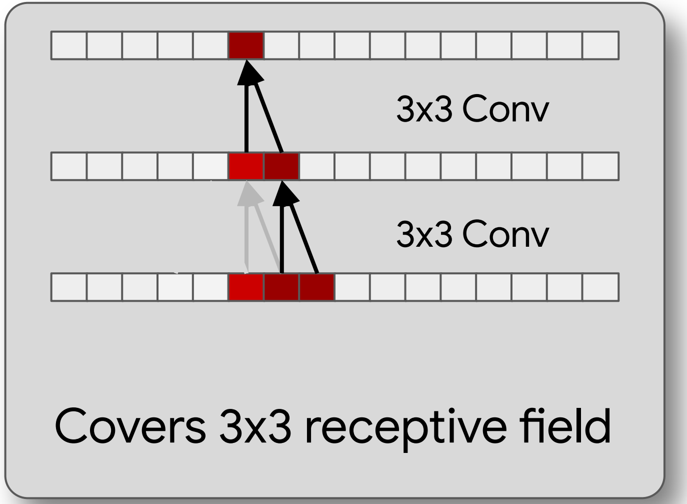
Spatial Factorization into (non-square) asymmetric convolutions



A little smaller

Params: 9*C^2
FLOPS: 9*C^2*H*W

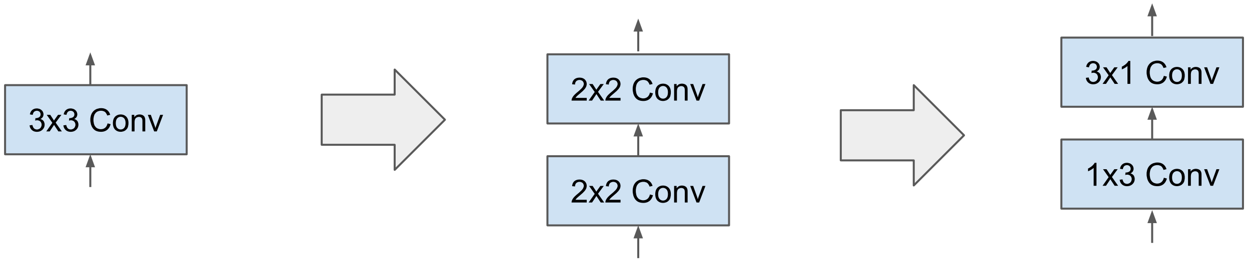
Params: 2*4*C^2 = 8C^2
FLOPS: 2*4*C^2*H*W = 8*C^2*H*W



But... we can do even better :)

[Rethinking the Inception Architecture for Computer Vision] by Szegedy et al

Spatial Factorization into (non-square) asymmetric convolutions



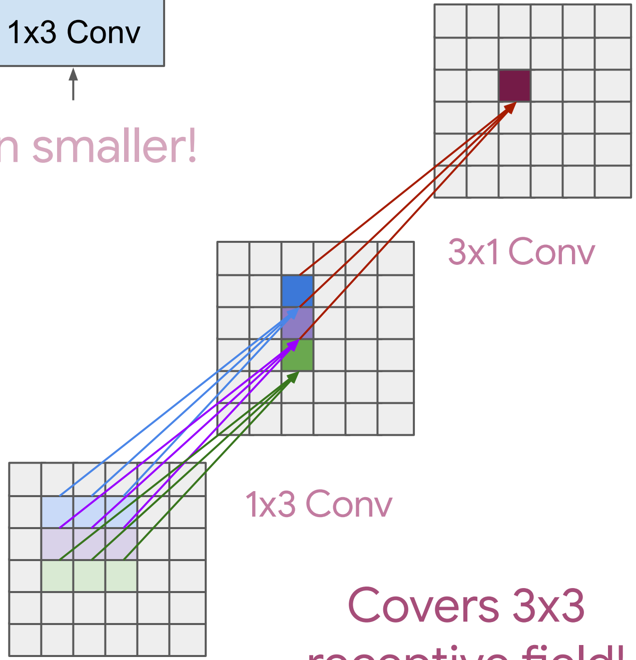
A little smaller

Even smaller!

Params: 9*C^2
FLOPS: 9*C^2*H*W

Params: 8C^2
FLOPS: 8*C^2*H*W

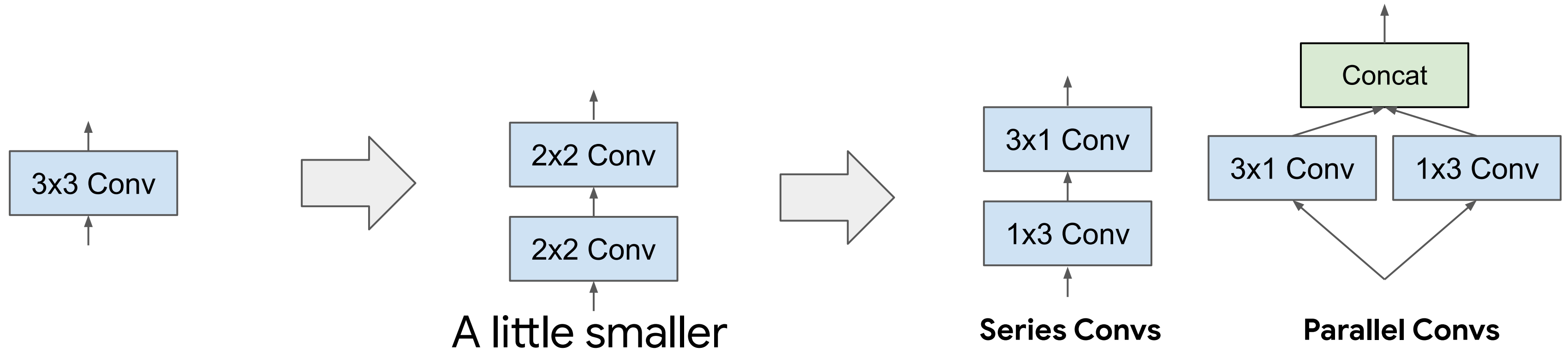
Params: 2*3*C^2 = 6C^2
FLOPS: 2*3*C^2*H*W = 6*C^2*H*W



Covers 3x3 receptive field!

Rethinking the Inception Architecture for Computer Vision by Szegedy et al

Spatial Factorization into (non-square) asymmetric convolutions



Even smaller!

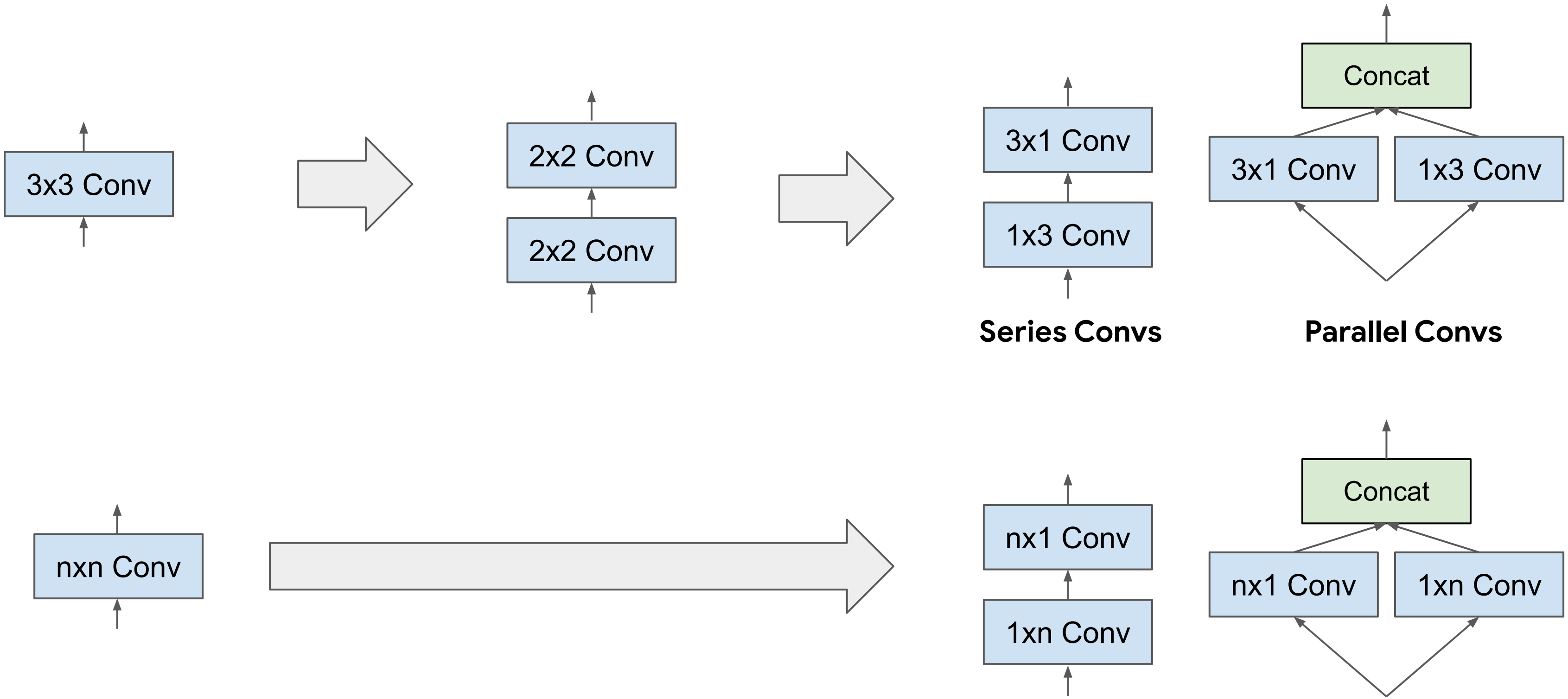
Params: 9*C^2
FLOPS: 9*C^2*H*W

Params: 8C^2
FLOPS: 8*C^2*H*W

Params: 6C^2
FLOPS: 6*C^2*H*W

Rethinking the Inception Architecture for Computer Vision by Szegedy et al

Spatial Factorization into (non-square) asymmetric convolutions

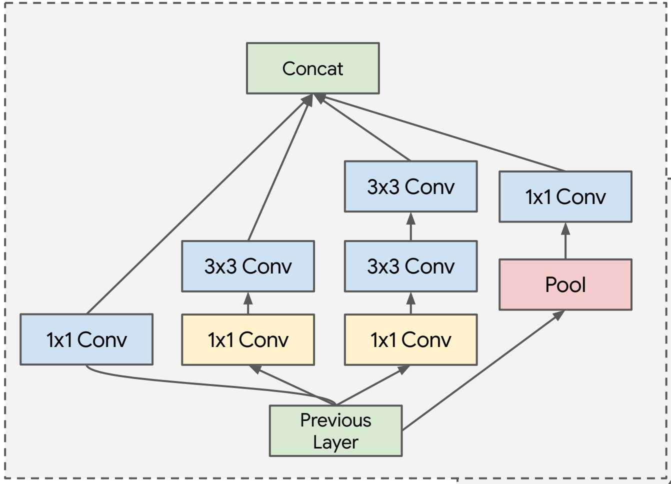


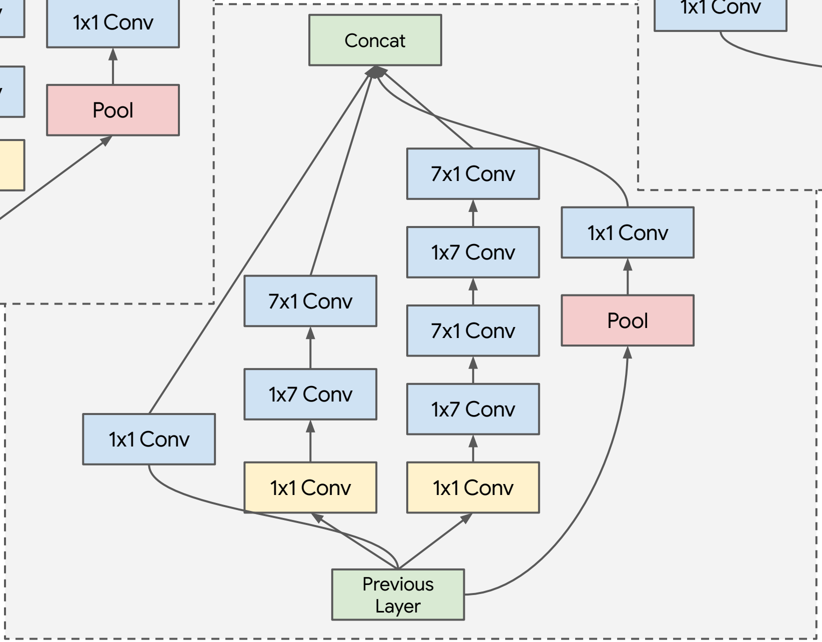
Params: N^2 * C^2
FLOPS: N^2 * C^2 * H * W

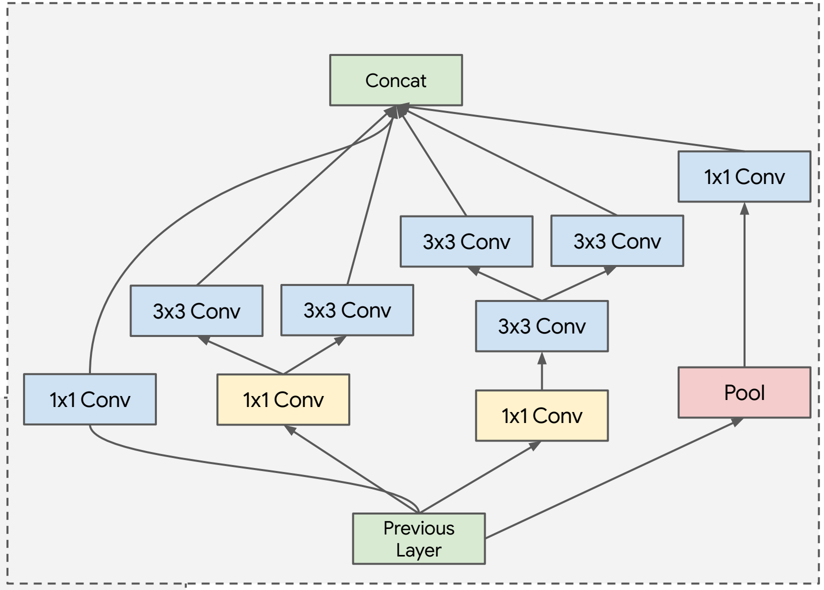
Params: 2 * N * C^2
FLOPS: 2 * N * C^2 * H * W

Rethinking the Inception Architecture for Computer Vision by Szegedy et al

Inception v2 Block Types

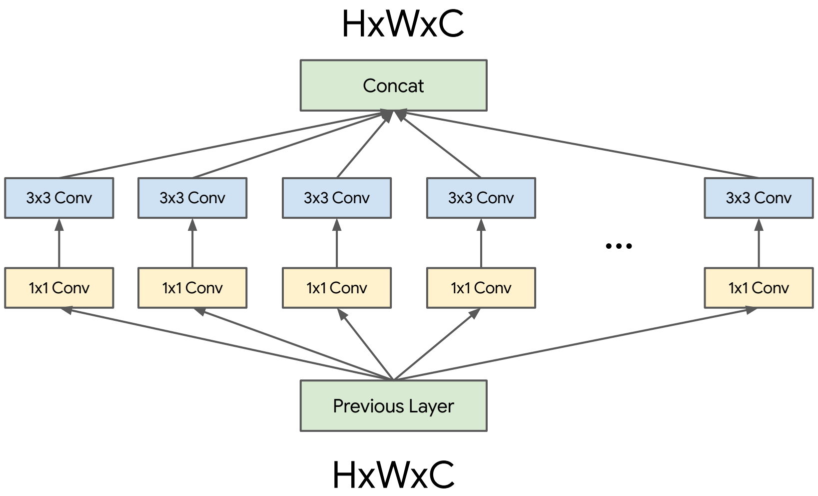






[Rethinking the Inception Architecture for Computer Vision] by Szegedy et al

Another variation: Taking bottleneck trick to extreme limit



- C parallel convolution paths
- Each 1x1 conv yields 1-d output

Parameters

C * C + C * 3 * 3

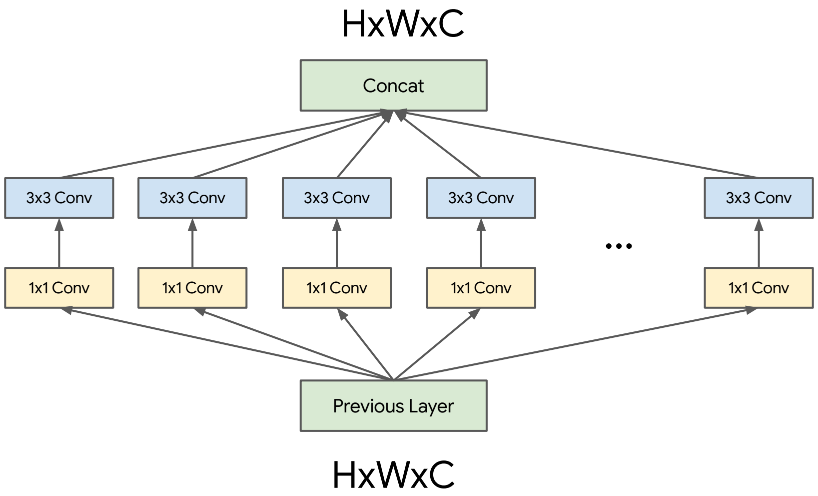
FLOPS

C * C + C * H * W * 3 * 3

Compare with "full" 3x3 Conv:

- Parameters: 3 * 3 * C * C
- FLOPS: 3 * 3 * H * W * C * C

Another variation: Taking bottleneck trick to extreme limit



- C parallel convolution paths
- Each 1x1 conv yields 1-d output

Parameters

C * C + C * 3 * 3

FLOPS

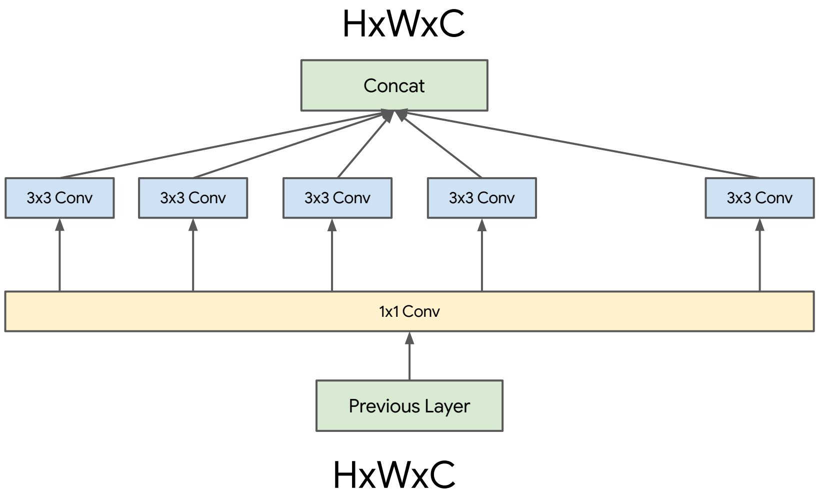
C * C + C * H * W * 3 * 3

Compare with "full" 3x3 Conv:

- Parameters: 3 * 3 * C * C
- FLOPS: 3 * 3 * H * W * C * C

Also known as a "separable convolution" or "depthwise separable" convolution

Separable Convolutions

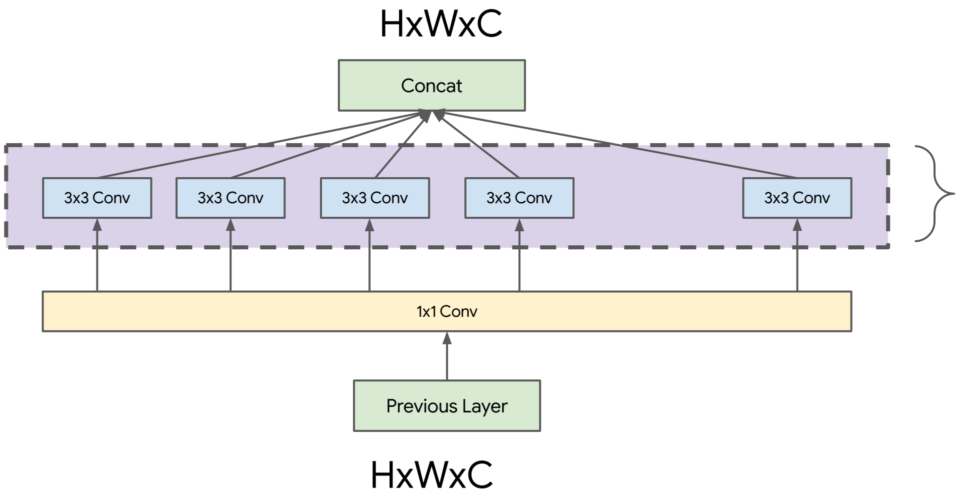


Each 3x3 conv operates independently on a single channel

Equivalent:

- *First apply 1x1 Conv (C->C)*
- *Then apply 3x3 Convs (1->1) along each channel*
- *Concatenate results*

Separable Convolutions

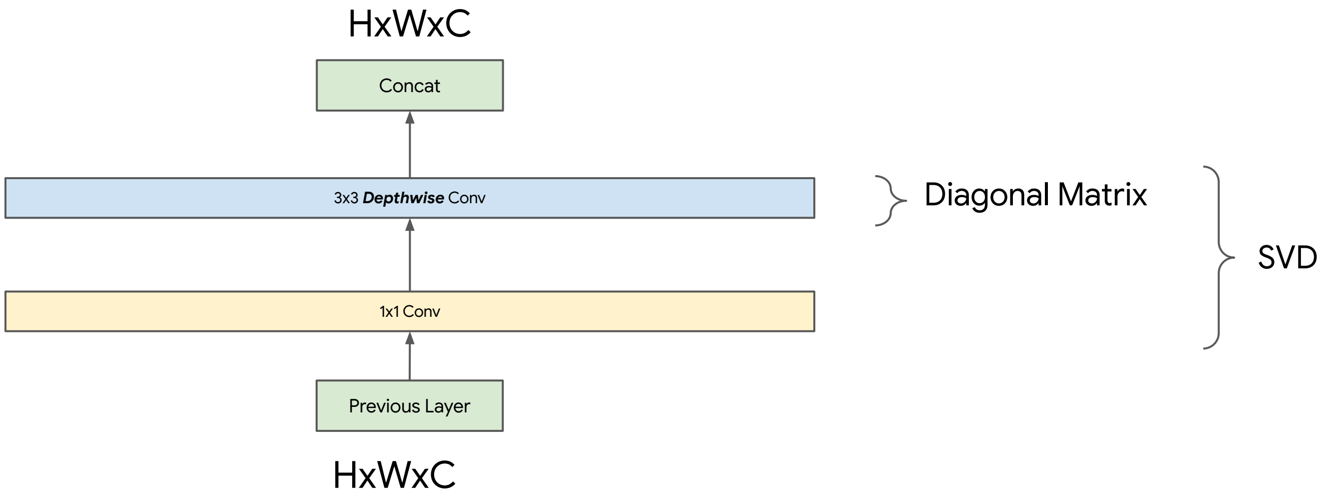


This grouping of independent single-channel convolutions sometimes called a ***depthwise convolution***

Equivalent:

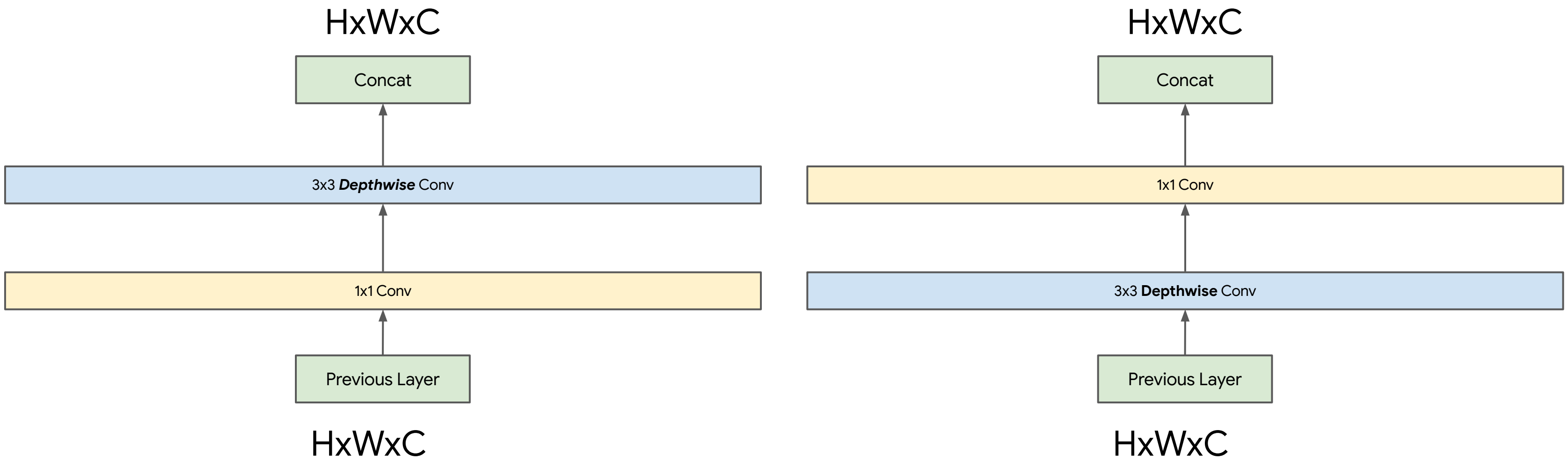
- *First apply 1x1 Conv (C->C)*
- *Then apply 3x3 Convs (1->1) along each channel*
- *Concatenate results*

Separable Convolutions



Separable Convs factor channel dependence from spatial dependence!

Separable Convolutions



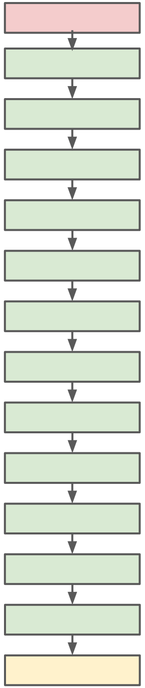
Note: conventionally, Separable Convs are Depthwise Conv followed by 1x1 Conv:

- Not quite equivalent, but same computational properties, difference goes away if you stack many separable convs together

Case Study (2017): MobileNet v1 (Howard et al)

“Full first convolution”

Separable convolutions w/VGG like structure



Type	Output Depth	Output Resolution
Convolution	32	112
Separable Convolution	64	112
Separable Convolution	128	56
Separable Convolution	128	56
Separable Convolution	256	28
Separable Convolution	256	28
Separable Convolution	512	14
Separable Convolution	512	14
Separable Convolution	512	14
Separable Convolution	512	14
Separable Convolution	512	14
Separable Convolution	512	14
Separable Convolution	1024	7
Separable Convolution	1024	7
Avg Pool + FC	1000	1

Early reduction in resolution (like GoogLeNet

No heavy FC

- 95% of computation is 1x1 convolutions efficiently implemented with GEMMs.

Slide credit: Andrew Howard

MobileNet Performance

100% MobileNet 224 Resolution

Model	Imagenet Accuracy	Million MACs	Million Parameters
MobileNet	70.6	568	4.2
Inception V1 TF (GoogleNet)	69.8	1550	6.8
VGG 16	71.5	15300	138

27X Less Computation than VGG16
32X Smaller than VGG16
Nearly Same Accuracy as VGG16

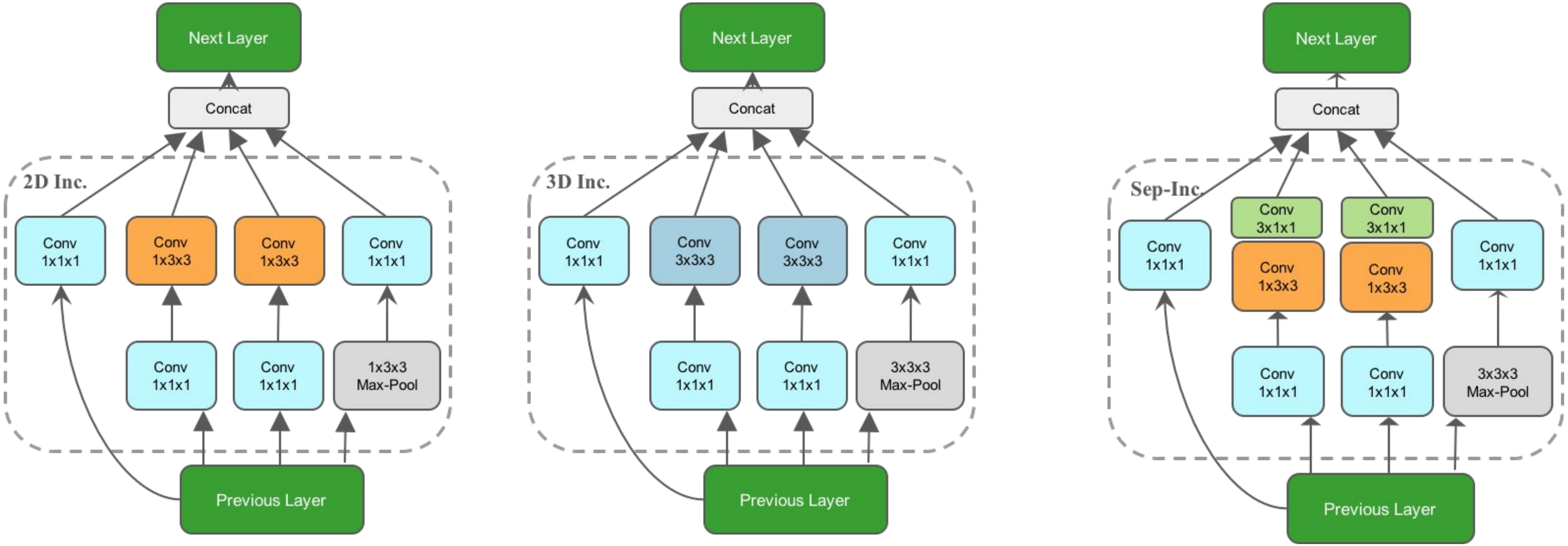
50% MobileNet 160 Resolution

Model	Imagenet Accuracy	Million MACs	Million Parameters
50% MobileNet 160 Resolution	60.2	76	1.32
Squeezenet	57.5	850	1.25
Alexnet	57.2	720	60

9.4X Less Computation than Alexnet
45X Smaller than Alexnet
3% Better than Alexnet

Slide credit: Andrew Howard

Generalization: Temporal Separability



Carreira and Zisserman. Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset. CVPR 2017.

Xie et al. "Rethinking Spatiotemporal Feature Learning For Video Understanding."

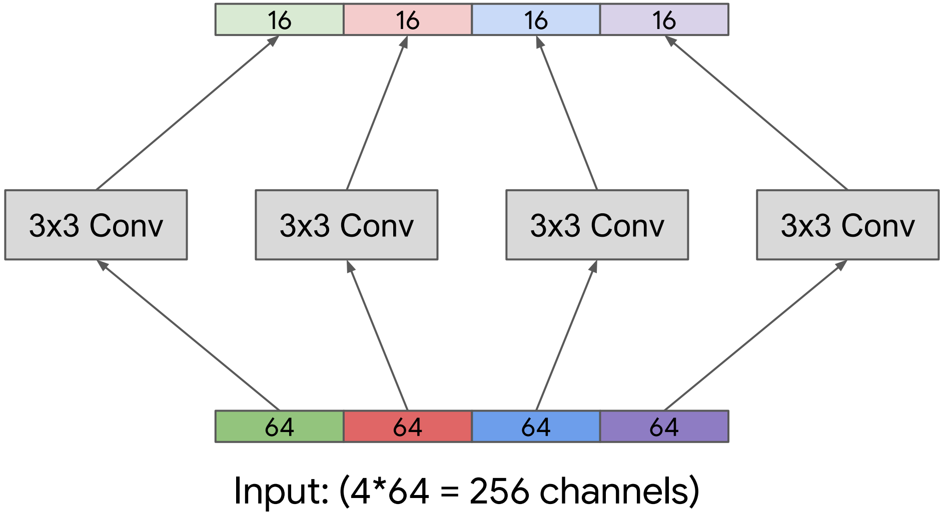
Another Generalization: “Grouped” Convolutions

Not to be confused w/”Group Convolutions”

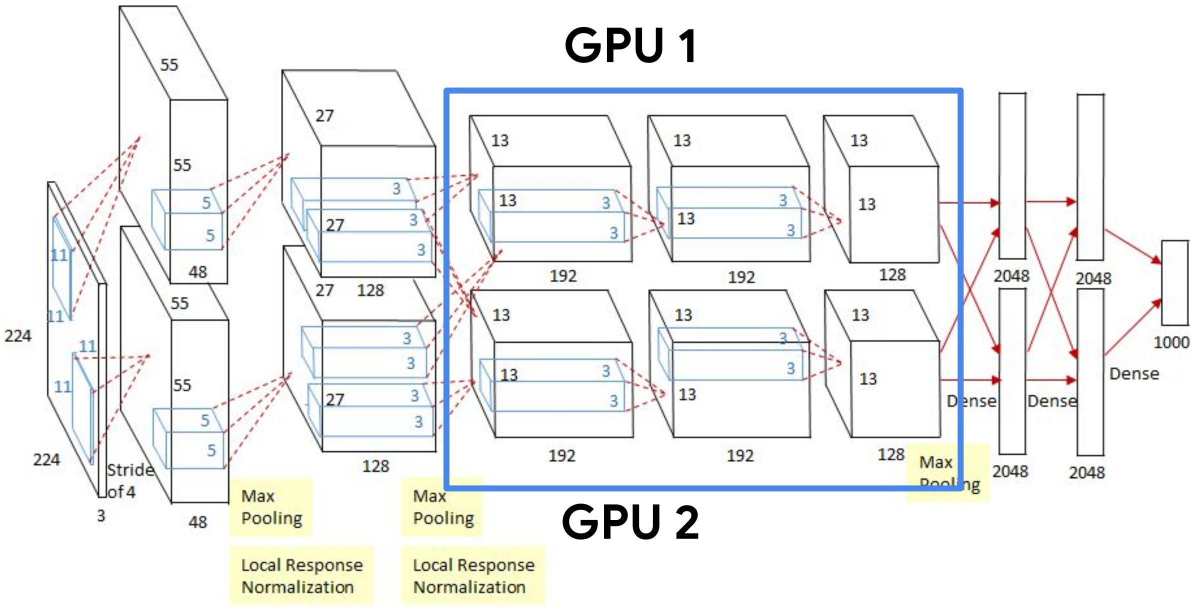
Parallel/Independent convolution pathways:

- Each Conv operates independently on a “group” of K input channels and produces its own “group” of L output channels

- Grouped Conv (with G groups) Op:
    - Input: GK channels
    - Output: LK channels



Grouped Convs in AlexNet



(Earlier we ignored this detail in the AlexNet paper)

Quick Recap

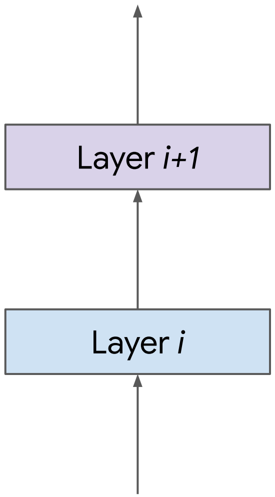
Spent a lot of time focusing on computation via **Factored Convolutions**:

- Inception Blocks
- Bottleneck layers
- Spatial Factorization
- Separable Convolutions (Bottleneck trick to the extreme)
- MobileNet, GoogLeNet, Inception V2
- Group Convolution (as a generalization of Separable Convolutions)

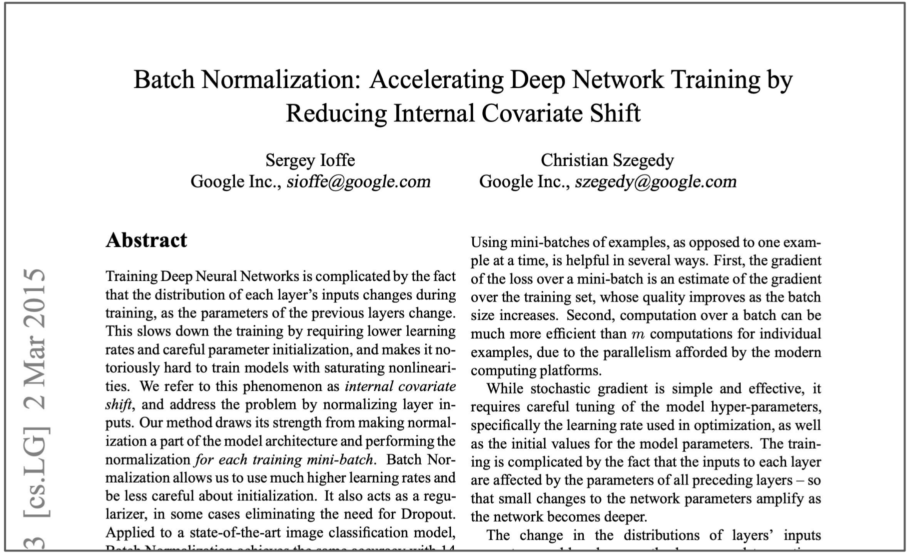
Let's turn to optimization issues. Next up:

- Batch norm
- Residual networks

Motivation: Internal Covariance Shift



During training, Layer i+1 needs to keep adapting to Layer i's shifting input distribution :(



Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe
Google Inc., *sioffe@google.com*

Christian Szegedy
Google Inc., *szegedy@google.com*

Abstract

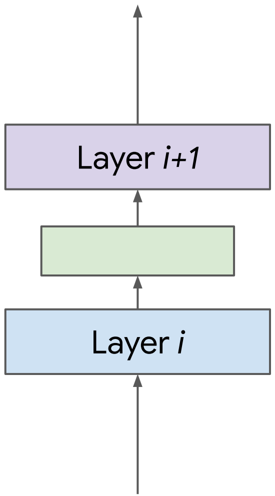
Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as *internal covariate shift*, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization *for each training mini-batch*. Batch Normalization allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout. Applied to a state-of-the-art image classification model,

Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a batch can be much more efficient than m computations for individual examples, due to the parallelism afforded by the modern computing platforms.

While stochastic gradient is simple and effective, it requires careful tuning of the model hyper-parameters, specifically the learning rate used in optimization, as well as the initial values for the model parameters. The training is complicated by the fact that the inputs to each layer are affected by the parameters of all preceding layers – so that small changes to the network parameters amplify as the network becomes deeper.

The change in the distributions of layers' inputs

Motivation: Internal Covariance Shift



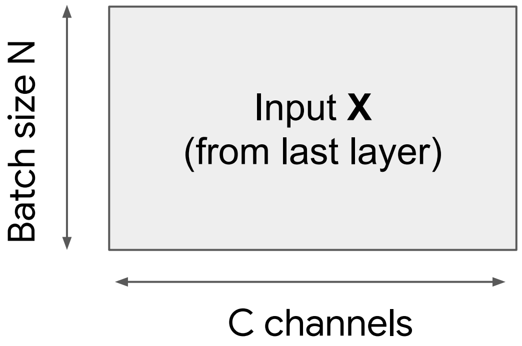
During training, Layer i+1 needs to keep adapting to Layer i's shifting input distribution :(

Idea of batch norm: Add intermediate layer that normalizes Layer i's output distribution to zero mean, unit variance.

Desiderata: Want this new layer to be:

- Differentiable
- Computationally efficient

Batch Normalization (for FC layers)



Shape: [1, C]

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

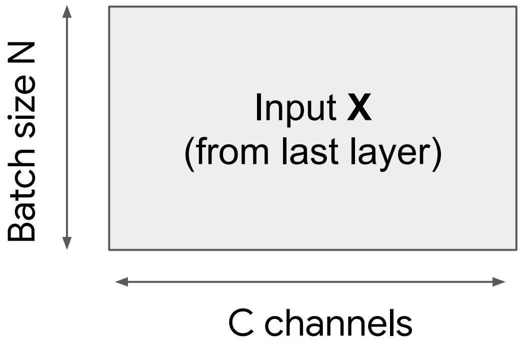
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

Approach:

- Ideally, normalize by entire training dataset --- but if we need to do this every step, too expensive. *Normalize by minibatch stats instead.*
- Normalize features independently.

Batch Normalization (for FC layers) Training



Shape: [1, C]

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Gamma, **beta**, learnable parameters!

Relax hard zero mean,
unit variance constraint;

Allows BN to recover identity function (if that were the optimal thing to do)

Approach:

- Ideally, normalize by entire training dataset --- but if we need to do this every step, too expensive. *Normalize by minibatch stats instead.*
- Normalize features independently.

Getting the Batch Norm Statistics Right

- If minibatch size m too small: "batch norm statistics" will be very noisy
  - When training on multiple GPUs, typically estimate per-device BN statistics; but for small batch sizes, often better to sync statistics across devices
- At test time, estimate batch norm statistics by averaging over very large set (using moving averages)

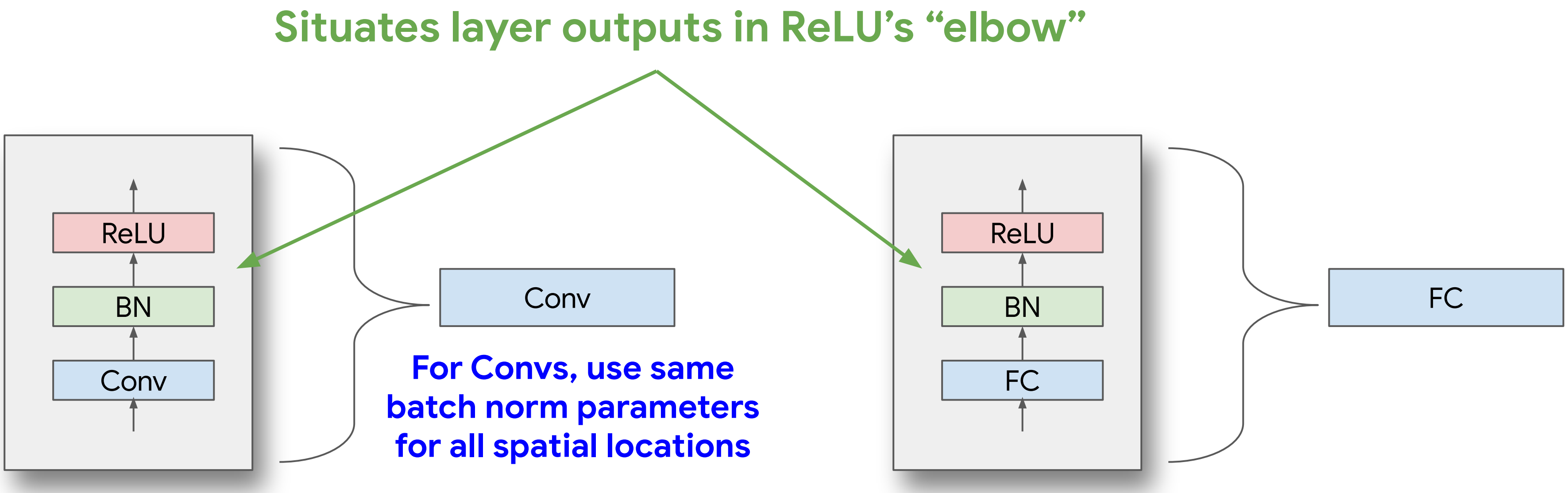
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Typical Batch Norm Usage

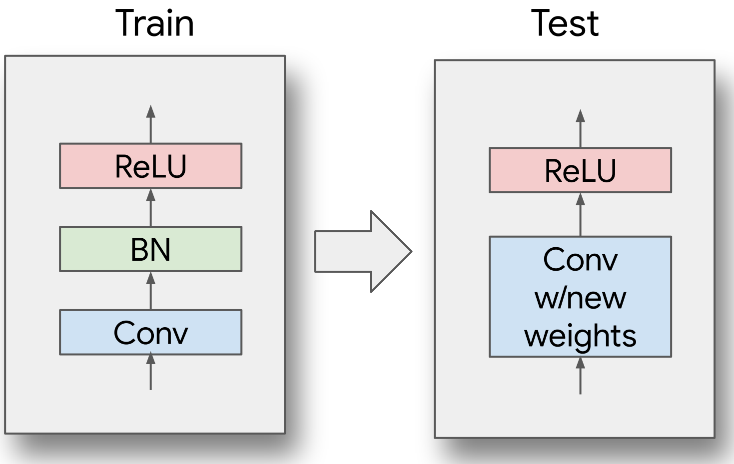


({Conv, FC} -> Batch Norm -> ReLU) is the typical pattern for most modern convnets (except at the last layer)

- Note: can remove bias parameter from previous layer when using BN

We will assume (henceforth) that BN and ReLU are present when we use "Conv"

Batch Norm Folding/Fusing

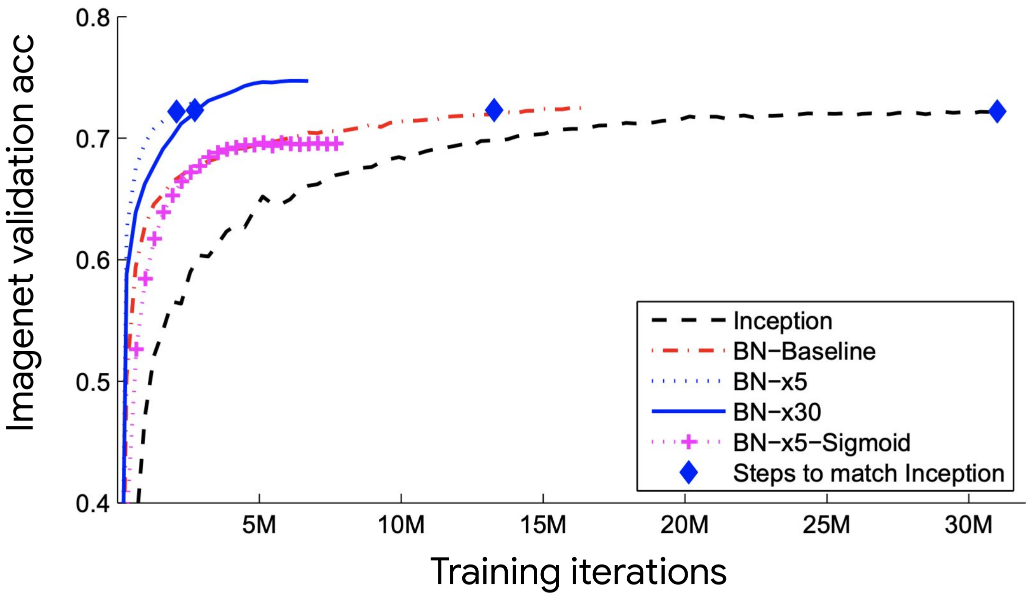


Since Batch Norm is linear w.r.t input, at test time (you can think of it as a 1x1 Conv if you want), the operation can be merged into the previous Conv/FC

So adding BN to a ConvNet does not introduce additional computation at inference time

Example: Inception-BN on ImageNet

- Simpler variant of Inception v2; a whopping ~30 layers (by my count)
- Batch norm before every nonlinearity



Batch Norm Benefits/Gotchas

- Reduces Internal Covariate Shift (maybe, not really?)
- Smooths optimization landscape,
- Helps stabilize, regularize, speed up training
- No added computation at test time
- Reduces need to do dropout
- Hard to debug sometimes - different train/test modes
- Batch norm “wants” a large batch size
- Output for a given example now has a strange dependency on everything in minibatch

Quick Recap

- Batch Normalization motivation: “internal covariate shift”
- Batch Normalization update equations
- Folded Batch Normalization parameters
- Many successor to Batch Norm: e.g., GroupNorm, Batch Renorm, Filter Response Normalization... but Batch Norm is still king :)

So far, we skipped around a bit - but now we return back to end of 2015...

Residual Networks (2015)

Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

Abstract

Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. We provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth. On the ImageNet dataset we evaluate residual nets with a depth of up to 152 layers—8× deeper than VGG nets [41] but still having lower complexity. An ensemble of these residual nets achieves 3.57% error on the ImageNet test set. *This result won the 1st place on the ILSVRC 2015 classification task. We also present analysis on CIFAR-10 with 100 and 1000 layers.*

The depth of representations is of central importance for many visual recognition tasks. Solely due to our extremely deep representations, we obtain a 28% relative improvement on the COCO object detection dataset. Deep residual nets are foundations of our submissions to ILSVRC & COCO 2015 competitions[1]*, where we also won the 1st places on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation.*

1. Introduction

Deep convolutional neural networks [22, 21] have led to a series of breakthroughs for image classification [21, 50, 40]. Deep networks naturally integrate low/mid/high-level features [50] and classifiers in an end-to-end multi-layer fashion, and the "levels" of features can be enriched by the number of stacked layers (depth). Recent evidence [41, 44] reveals that network depth is of crucial importance, and the leading results [41, 44, 13, 16] on the challenging ImageNet dataset [36] all exploit "very deep" [41] models, with a depth of sixteen [41] to thirty [16]. Many other non-trivial visual recognition tasks [8, 12, 7, 32, 27] have also

[1]http://image-net.org/challenges/LSVRC/2015/ and http://mscoco.org/dataset/#detections-challenge2015.

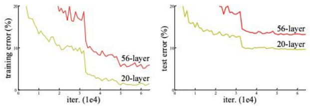


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

greatly benefited from very deep models.

Driven by the significance of depth, a question arises: *Is learning better networks as easy as stacking more layers?* An obstacle to answering this question was the notorious problem of vanishing/exploding gradients [1, 9], which hamper convergence from the beginning. This problem, however, has been largely addressed by normalized initialization [23, 9, 37, 13] and intermediate normalization layers [16], which enable networks with tens of layers to start converging for stochastic gradient descent (SGD) with back-propagation [22].

When deeper networks are able to start converging, a *degradation* problem has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. Unexpectedly, such degradation is *not caused by overfitting*, and adding more layers to a suitably deep model leads to *higher training error*, as reported in [11, 42] and thoroughly verified by our experiments. Fig. 1 shows a typical example.

The degradation (of training accuracy) indicates that not all systems are similarly easy to optimize. Let us consider a shallower architecture and its deeper counterpart that adds more layers onto it. There exists a solution *by construction* to the deeper model: the added layers are *identity* mapping, and the other layers are copied from the learned shallower model. The existence of this constructed solution indicates that a deeper model should produce no higher training error than its shallower counterpart. But experiments show that our current solvers on hand are unable to find solutions that

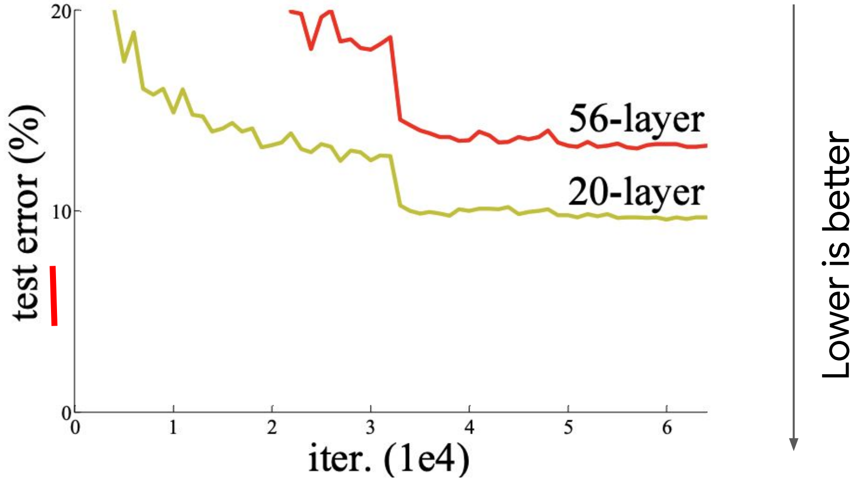
From ~20 layers to >100 layers!

ResNets @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
  - ImageNet Classification: *"Ultra-deep"* 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

What would happen if we could just add more layers?

(if compute weren't an issue)



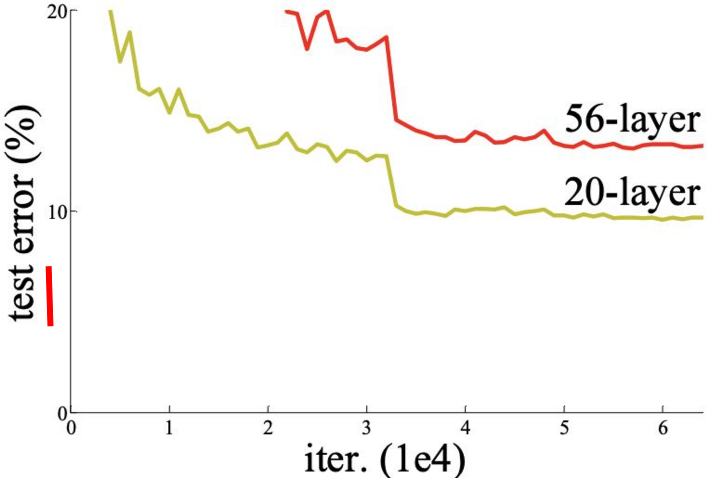
CIFAR dataset (32x32 inputs)

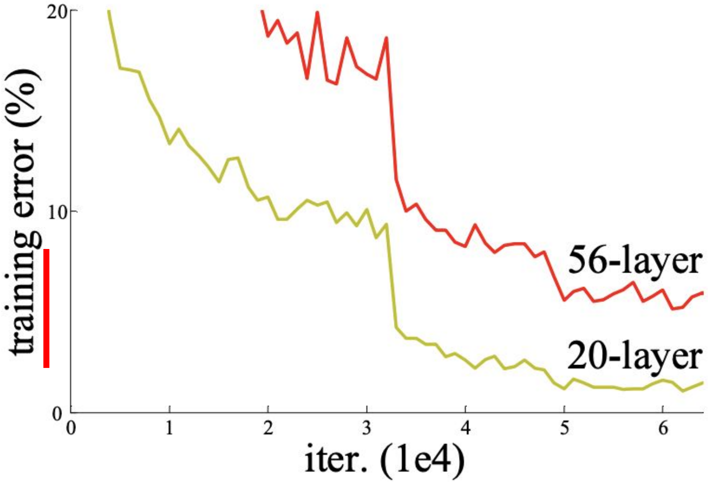
Observation: Deep 56 layer net underperforms shallower 20 layer net.

Hypothesis: Overfitting?? Let's check train error

[Deep Residual Learning for Image Recognition](#) by He et al

What would happen if we could just add more layers?
(if compute weren't an issue)





Observation: Deep 56 layer net still underperforms shallower 20 layer net in training error!!

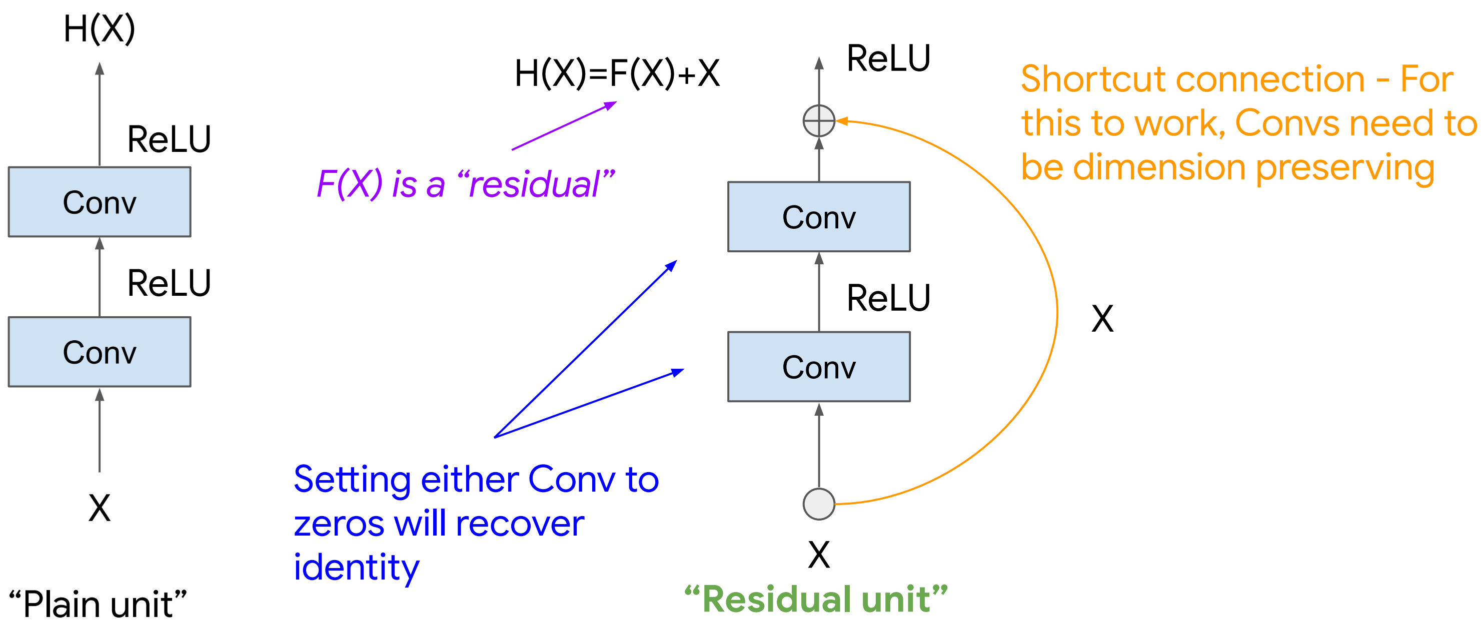
[Deep Residual Learning for Image Recognition](#) by He et al

Next Hypothesis: Optimization issue? Is SGD is harder for deeper models (e.g. due to vanishing gradients?)

Idea: Let’s make it “easy” for optimizer to learn identity transforms in extra layers

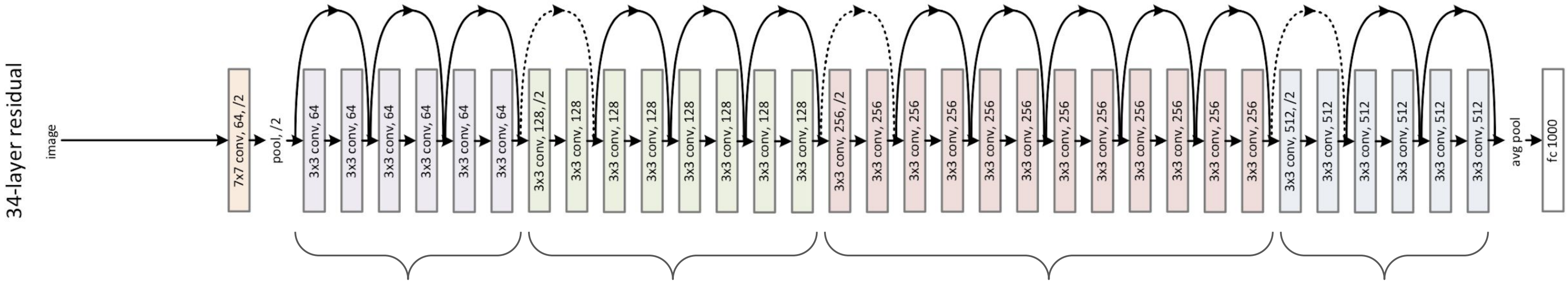
- Why would this help?
    - If so, then we can always set additional layers of a deep network to be identity and mimic performance of a shallow model
    - In this case, performance of deep network should always be equal or better to shallow network on training loss

Identity mapping with shortcuts



Residual Networks

- Use Conv w/stride 2 instead of Pool
- Like VGG - extremely simple structure
- Like inception, aggressively reduce resolution in early layers, Pool at top with no heavy FC



Block 1, c=64, 3 residual units

Block 2, c=128, 4 residual units

Block 3, c=256, 6 residual units

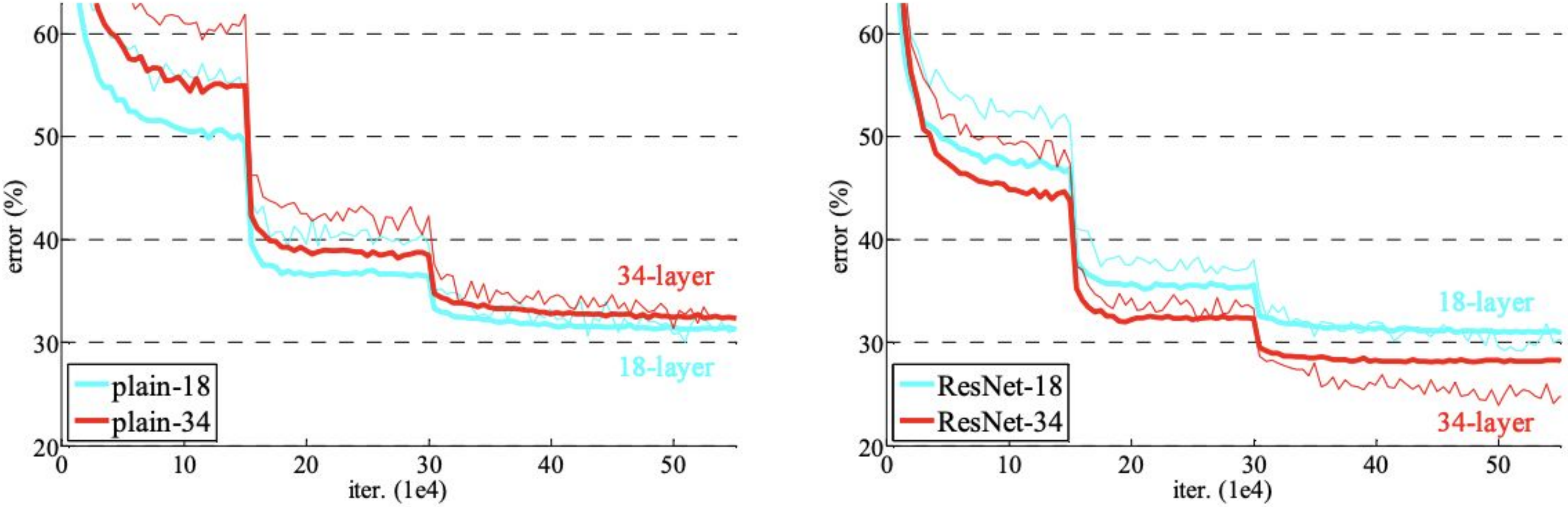
Block 4, c=512, 3 residual units

Special case residual units when we change resolution (use 1x1 Conv(X) instead of X in shortcut w/o ReLU)

(3+4+6+3 residual units) * (2 convs per residual unit) + First conv + Last FC

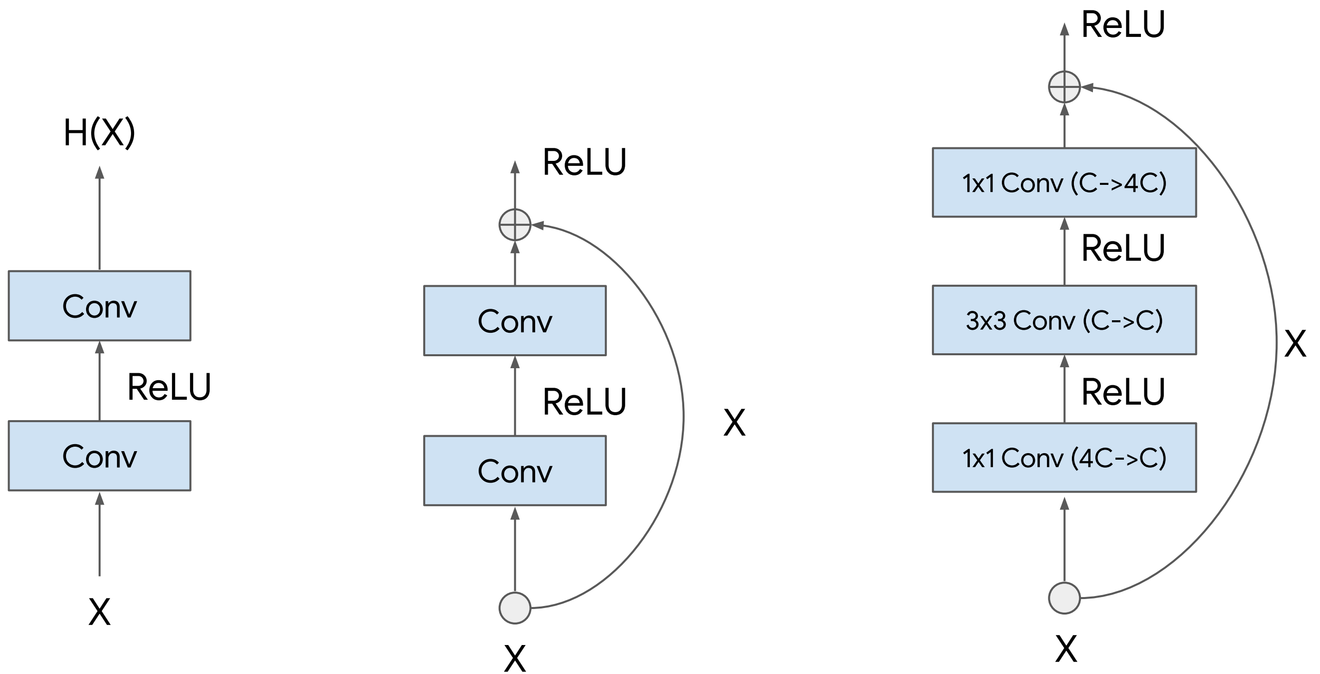
= ***34 layers***

Residual Networks solve the optimization problem



Residual Connections allow deeper network to outperform shallower network!

Bottleneck Units



"Plain unit" "Basic Residual unit" **"Bottleneck Residual unit"**

Deeper for less compute

Resnet 18/34/50/101/152

Input size: 224x224

Basic Residual Units: 18-layer, 34-layer — Bottleneck Residual Units: 50-layer, 101-layer, 152-layer

	layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
	conv1	112×112	7×7, 64, stride 2				
Block 1	conv2_x	56×56	3×3 max pool, stride 2				
			$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
Block 2	conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
Block 3	conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
Block 4	conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
		1×1	average pool, 1000-d fc, softmax				
	FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

FLOPS for comparison: MobileNet (v1): $2.5\text{x}10^8$ VGG: $19.6\text{x}10^9$

Often see ablations done with a smaller Resnet, then experiments that "pull out all the stops" with a heavier variant

Total World Dominance (on ImageNet and COCO)

We will cover COCO later

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

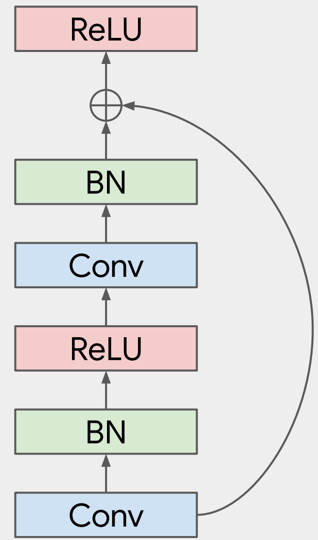
Human top-5 error ~5%

Single Model Results

After 5 years, Resnet still ubiquitously used!

Resnet v2 w/Pre-activation residual units

Original Residual unit

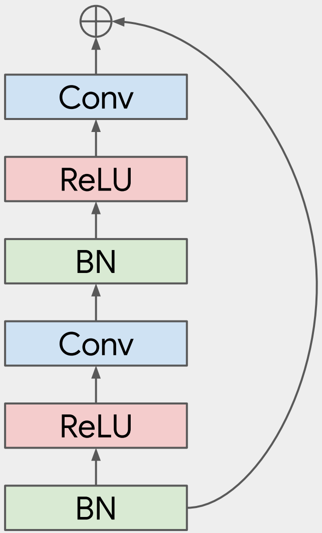


Can't recover identity function from stacked original Residual Units because of ReLU

Allows for entire network to recover identity function (if we ignore downsampling layers)

Better for backprop; allows deeper models to be trained (e.g. 1001-layer Resnet on CIFAR)

"Pre-activation Residual Unit"



Remember: before, we were implicitly assuming Batch Norm as part of the Conv

Identity Mappings in Deep Residual Networks by He et al

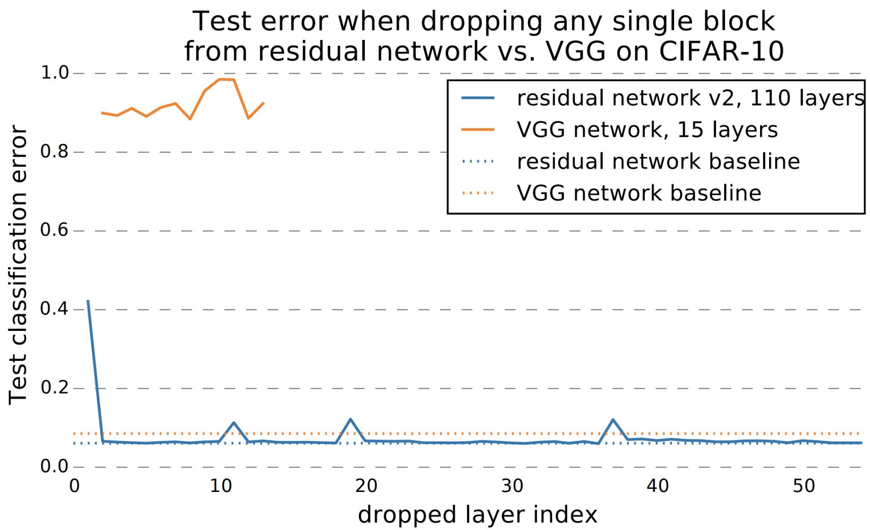
What are all those layers doing!!?!

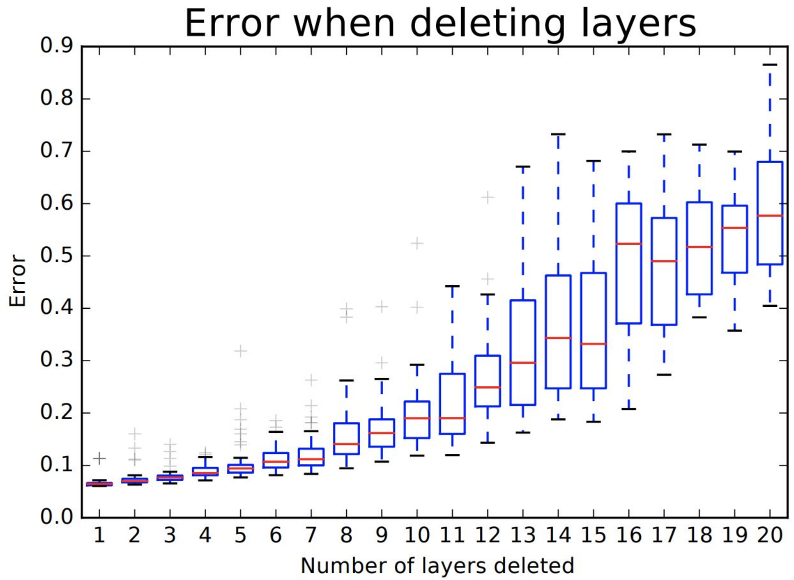
Answer: being *very* redundant! Let's discuss a few ways to think about these layers.

"While depth of representation has been posited as a primary reason for their success, there are indications that these architectures defy a popular view of deep learning as a hierarchical computation of increasingly abstract features at each layer."

Highway and Residual Networks Learn Unrolled Iterative Estimation, Greff et al

Dropping blocks from ResNet

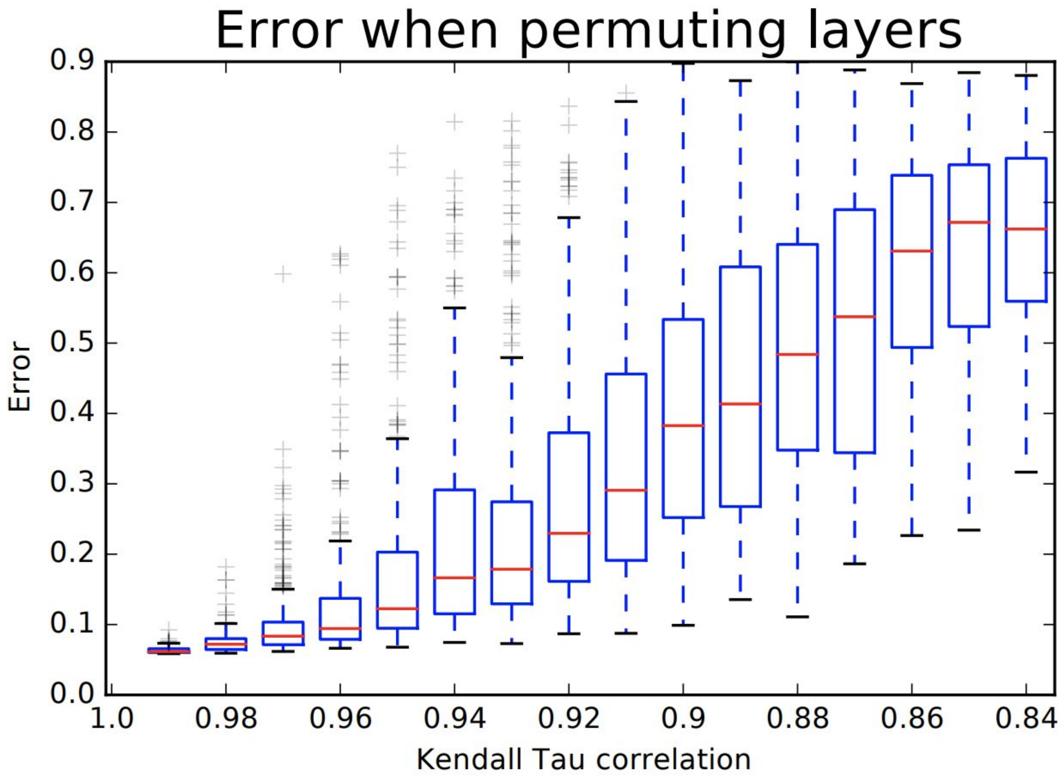




Weird but true fact: you can delete blocks from Resnet (even after training) and expect performance to be roughly the same (!)

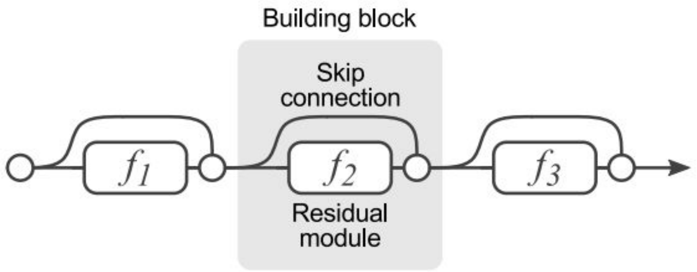
Residual Networks Behave Like Ensembles of Relatively Shallow Networks by Veit et al

Permuting Blocks from Resnet

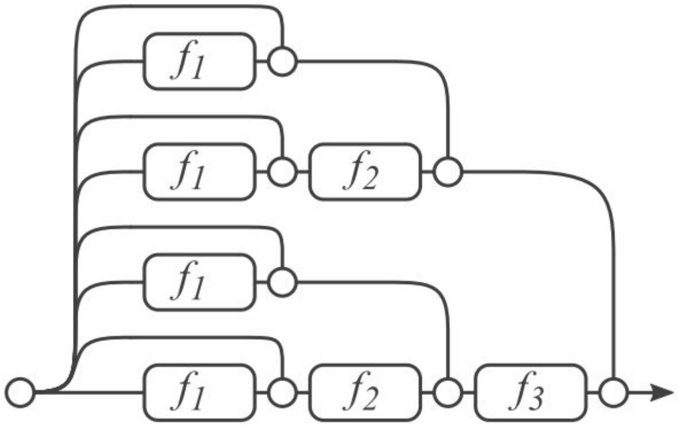


[Residual Networks Behave Like Ensembles of Relatively Shallow Networks]() by Veit et al

Multipath Ensembling Interpretation



Simple Resnet with 3 residual units



Equivalent "Unraveled View"

Resnet behaves like an ensemble over an exponential collection of networks consisting of paths through this unraveled view --- (though note that it is not actually an ensemble.)

[Residual Networks Behave Like Ensembles of Relatively Shallow Networks](#) by Veit et al

Iterative Estimation interpretation of Resnets

- Residual Connections Encourage Iterative Inference by Jastrzebski et al

- Highway and Residual Networks Learn Unrolled Iterative Estimation by Greff et al

RESIDUAL CONNECTIONS ENCOURAGE ITERATIVE INFERENCE

Stanisław Jastrzębski[1,2,*], Devansh Arpit[2,*], Nicolas Ballas[3], Vikas Verma[5], Tong Che[2] & Yoshua Bengio[2,6]

[1] Jagiellonian University, Cracow, Poland
[2] MILA, Université de Montréal, Canada
[3] Facebook, Montreal, Canada
[4] University of Bonn, Bonn, Germany
[5] Aalto University, Finland
[6] CIFAR Senior Fellow
[*] Equal Contribution

ABSTRACT

Residual networks (Resnets) have become a prominent architecture in deep learning. However, a comprehensive understanding of Resnets is still a topic of ongoing research. A recent view argues that Resnets perform iterative refinement of features. We attempt to further expose properties of this aspect. To this end, we study Resnets both analytically and empirically. We formalize the notion of iterative re-

...ly encourage fea-
...oss as we go from
...s that Resnets are
...ment. In general,
...havior in the first
...features. Finally
...ntation explosion
...ing strategies can

Resnets both analytically and empirically. **We formalize the notion of iterative refinement in Resnets by showing that residual connections naturally encourage features of residual blocks to move along the negative gradient of loss as we go from one block to the next.** In addition, our empirical analysis suggests that Resnets are

Quick Recap

- Residual Connections as a way to “easily” learn identity transformation
- Resnet Architectures with Basic and Bottleneck residual units
- Pre-activation residual units
- Layer redundancy, ensemble-like behavior and other theoretical interpretations of Resnets

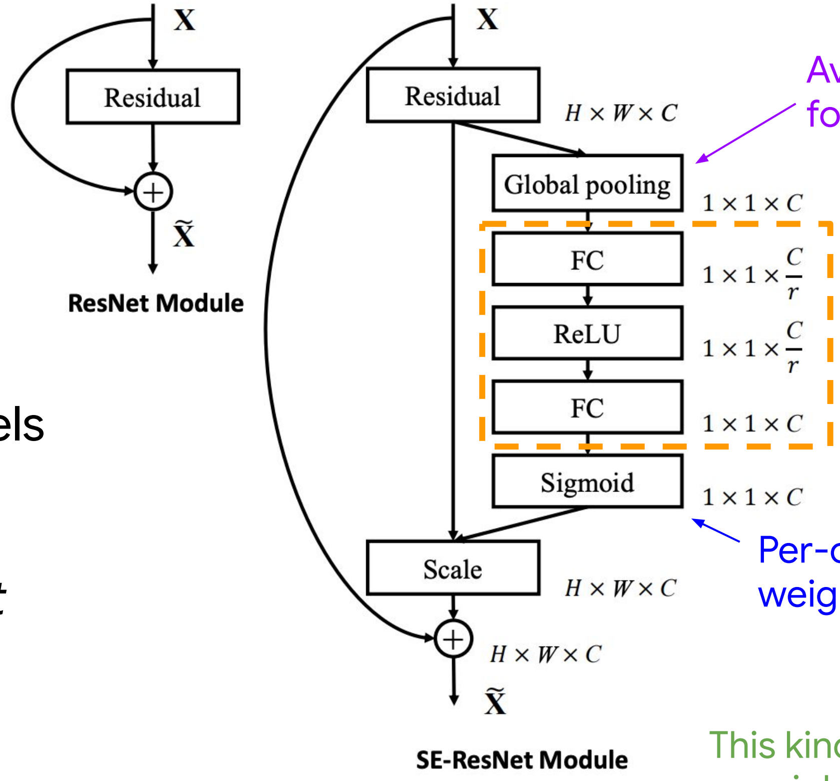
ImageNet since Residual Networks

- 2016: Ensembles of Inception and Resnet based models
- 2017: Squeeze and Excitation networks

Post 2017

- More emphasis on automating architecture design

Squeeze and Excitation



Average pooling
for global context

Apply small net
w/bottleneck trick

Per-channel [0, 1]
weights

This kind of feature reweighting is sometimes called self-gating

Idea: Use global image context to selectively emphasize/suppress channels

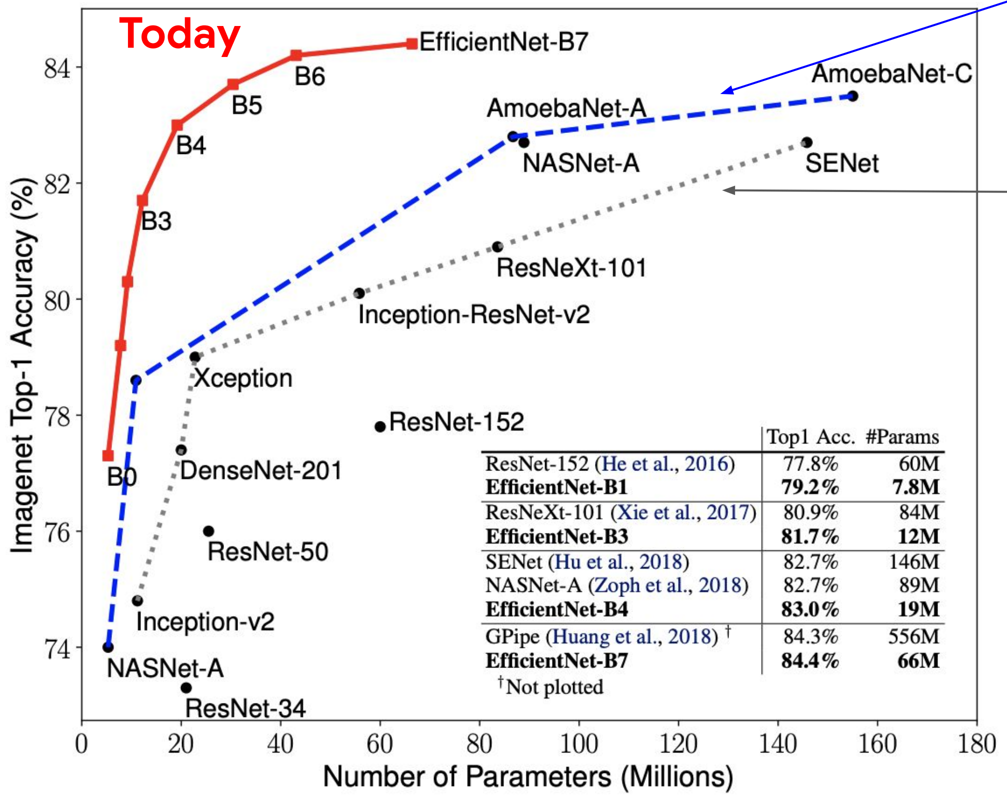
SE modules + Resnet variant won Imagenet 2017

[Squeeze-and-Excitation Networks](#) by Hu et al

Neural Architecture Search (NAS)

NAS circa 2017, 2018

Human-designed

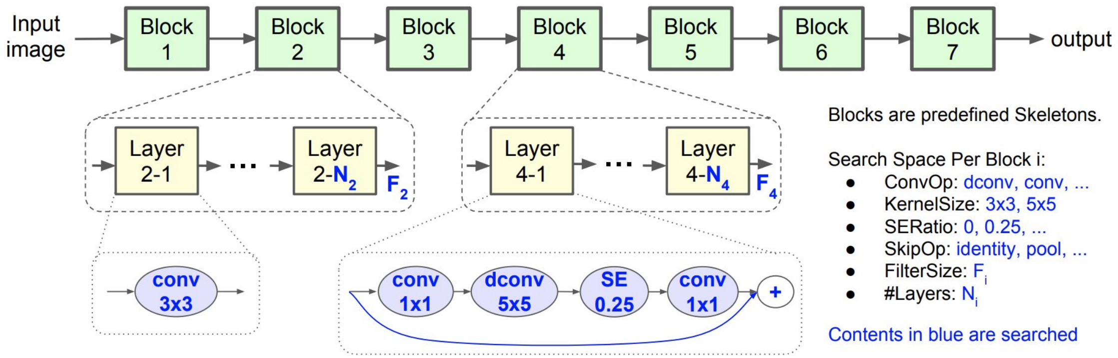


	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.2%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.7%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**83.0%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

†Not plotted

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks by Tan & Le

Neural Architecture Search (NAS)



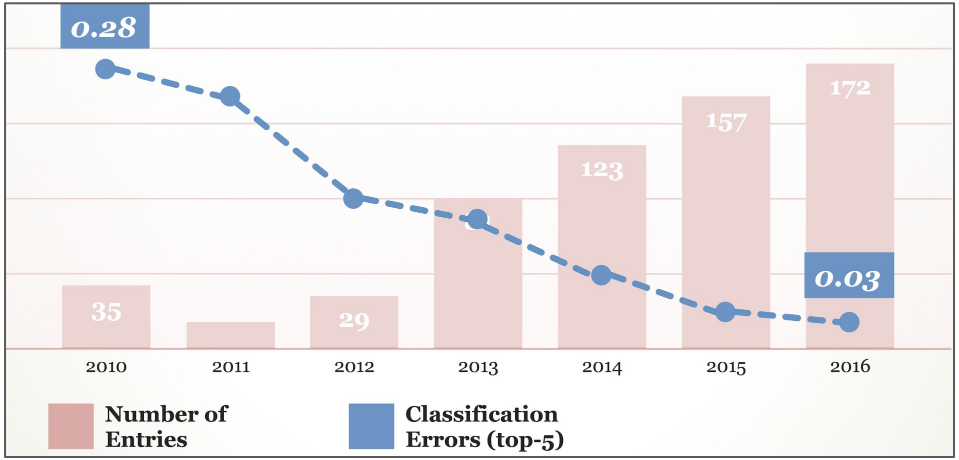
Three ingredients of a NAS system:
- Search space
- Search strategy
- Performance Estimation

- Neural architecture search with reinforcement learning by Zoph et al
- Learning transferable architectures for scalable image recognition by Zoph et al
- Progressive Neural Architecture Search by Liu et al
- MnasNet: Platform-Aware Neural Architecture Search for Mobile by Tan et al
- EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks by Tan et al
- DARTS: Differentiable architecture search by Liu et al
- Neural Architecture Search: A Survey by Elsken et al

Fig from MnasNet: Platform-Aware Neural Architecture Search for Mobile by Tan et al

ImageNet Coda

After 8 years (2017), ImageNet team declared victory, moved competition to Kaggle



Impact:

- 10x reduction of image classification error, beating human level performance
- >15K citations (major underestimate of impact)
- "Made neural nets cool again"
- Inspired many datasets --- "ImageNet of X"

[ImageNet: Where have we been? Where are we going?] by Fei Fei Li and Jia Deng

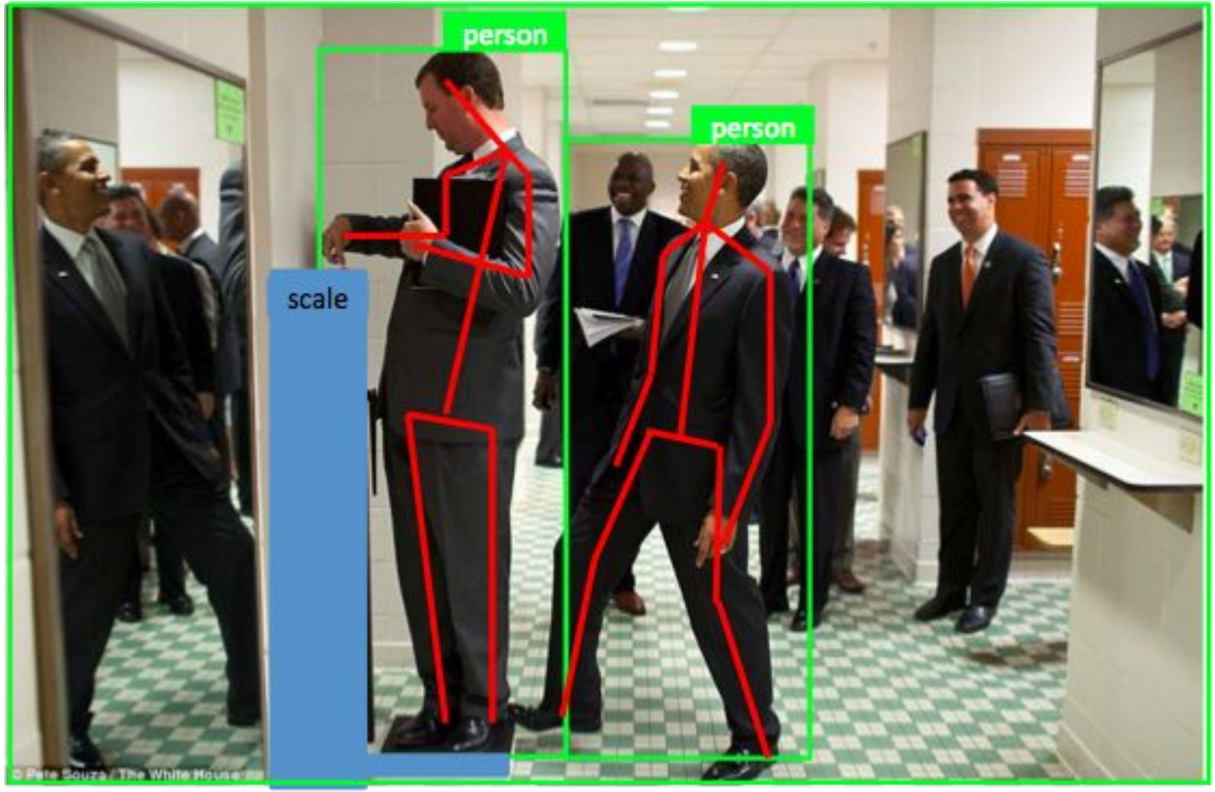
“This is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.”

WINSTON CHURCHILL

(Quote from Fei Fei Li and Jia Deng, quoting Winston Churchill)

So what's next?

Next: Boxes, Segments, Human Pose



Based on a figure from Jia Deng and Kevin Murphy