Economics 2355: CNN Architectures

Melissa Dell

Harvard University

February 2021

Outline

Plain Vanilla Networks

- In the introduction to deep learning, we saw a fully connected network, in which all neurons in one layer are connected to all neurons in the next layer
- This implies *a lot* of parameters as the size of the network grows
- It also fails to capture the structures inherent in much of the data we'd like to process

Structure in Image Data

- How do we as humans process images? The spatial structure is key. We interpret pixels in the context of other pixels around them.
- The vanilla network is devoid of spatial structure. That's why it isn't capable of detecting edges, etc.
- However, it is straightforward to build this structure into a network and doing so also substantially reduces the number of parameters we need to estimate

Convolutions

- Recall that we can think of each pixel in our input image as a neuron
- You can think of convolutions as a spatial filter that you slide over the image. You take the dot product between the input pixels and the weights in the filter, sum them up, and pass them through a non-linearity (i.e. ReLU) to compute the associated neuron in the next layer

Convolution

We do this by using spatial filters (weight matrices) to connect layers of the network

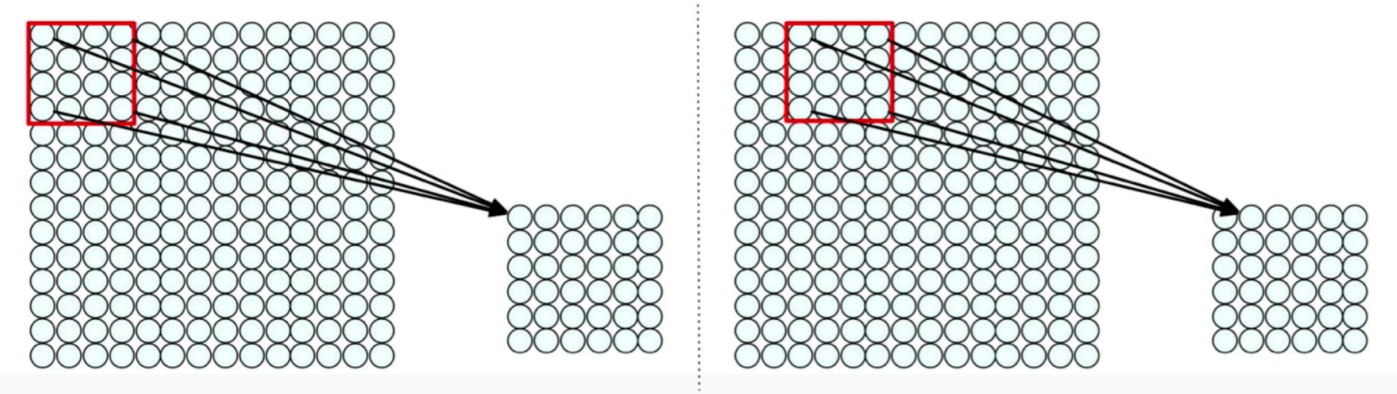


Image credit: Stanford CS 231n Lecture Notes

Convolution

- The filters can have different sizes, i.e. 3 x 3, 5 x 5, and 7 x 7 are common
- The filters can have different strides and different zero padding of image edges to control the number of neurons in the output layer. i.e. with 3 x 3 you zero pad by 1 to maintain the size with a stride of 1, with 5 x 5 you zero pad by 2, etc.
- The filters always extend through the full depth of the input, i.e. if you have a 25 x 25 x 3 input (i.e. a 25 x 25 pixel RGB image), you apply a single 5 x 5 (x 3) filter with stride 1, and zero pad the edges, your output layer will be 25 x 25 x 1.
- In practice, you'd never apply just one set of filters, i.e. a 25 x 25 x 3 input that you apply 92 3 x 3 filters with stride 1 to would yield a 25 x 25 x 92 output layer.

Convolution in Practice

- Different filters (weight matrices) will extract different sorts of features
- We need different filters to pull out different features (colors, edges, etc) as well as to address intra-class variation (i.e. a horse facing right versus left)
- A 3 x 3 filter with weights each equal to 1/9 would take a simple average. A Gaussian filter would put more weight on the middle pixels
- Other filters can pull out edges and a variety of other image features
- With deep learning, you are using training examples to estimate these filters, rather than hand engineering them (which is very brittle)

Pooling

- Pooling is used to downsample images, preserving the depth
- The most common approach is max pooling, which simply takes the max pixel value in a filter as it is moved across the image

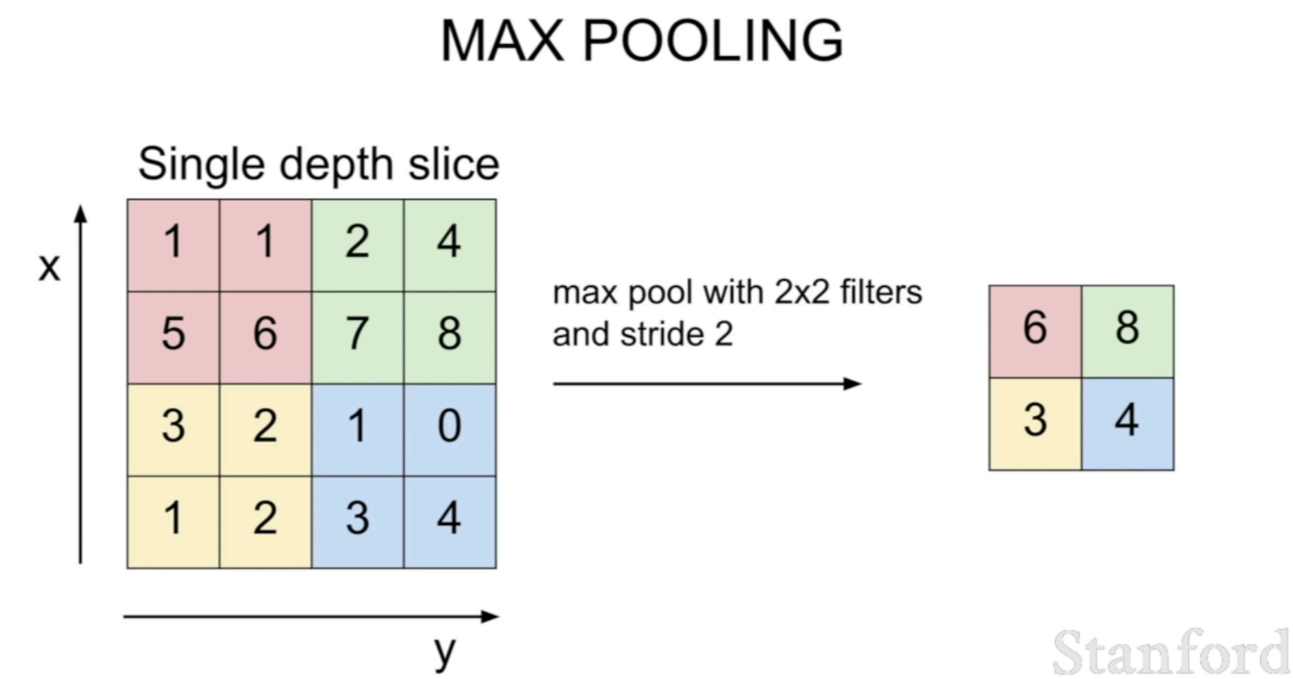


Image credit: Stanford CS 231n

Convolution versus Pooling

- Both convolution and pooling can be used to downsample, but they do so in very different ways
- Convolution involves parameters, estimated through backpropogation. Pooling doesn't have any parameters, you just take i.e. the max pixel value within the filter
- Multiple convolution filters are typically applied to an input layer, to extract different features (potentially while downsampling), whereas max pooling can be thought of as a single filter that extracts the max value
- Convolutions are applied over the depth of the image, i.e. a single 11 x 11 (x 3) filter applied to a 227 x 227 x 3 image at stride 4 will yield an output layer of dimension 55 x 55 x 1. Whereas max pooling using the same size filter and stride would yield a 55 x 55 x 3 output.

Outline

Benchmark Datasets

- CIFAR10: 60K images in 10 classes, with 6K images per class (smaller, older dataset)
- ImageNet: 14 million images, over 1 million with object bounding boxes (main dataset for advancement of this literature in the early and mid-2010s)
- COCO: an object detection, segmentation, and captioning dataset, with around 330K images and 1.5 million object instances; central benchmark for object detection and segmentation

Challenges

- Benchmark datasets often have annual challenges associated with them
- The architectures we will see in this lecture were developed for the ImageNet challenges in 2012-2016. Contest categories include image classification (top 1 and 5), object localization, and object detection
- Some people argue that datasets like ImageNet are even more important to the advancement of deep learning than the model architectures that have been developed
- The ImageNet challenge ended in 2017 because the challenge was essentially solved (higher than human accuracy); the COCO challenge continues, with a focus on detection, segmentation, and captioning

Outline

Linear Classifier

A classifier is a function that combines the inputs x with weights W to product a vector with the class probabilities: $F(W, x)$. A linear classifier is just $F(W, x) = Wx$.

Suppose we have 32 x 32 x 3 images and we want to classify which of 10 classes each of the images belongs to (i.e. car, horse, etc). The dimensions are:

- $F(W, x)$ - the class probabilities - is 10 x 1, a probability for each class
- If we restack x into a column vector, it will have dimension 3072 x 1. Hence, W must be 10 x 3072 to yield the probabilities for the 10 output classes.

This implies that we can think of the 10 x 3072 weight matrix as consisting of a single (32 x 32 x 3) template for each of the ten classes. Each image will be compared pixel by pixel to each of the template filters to assess the class probabilities (no convolutions in this simple model).

Linear Classifier as a Template Matching Approach

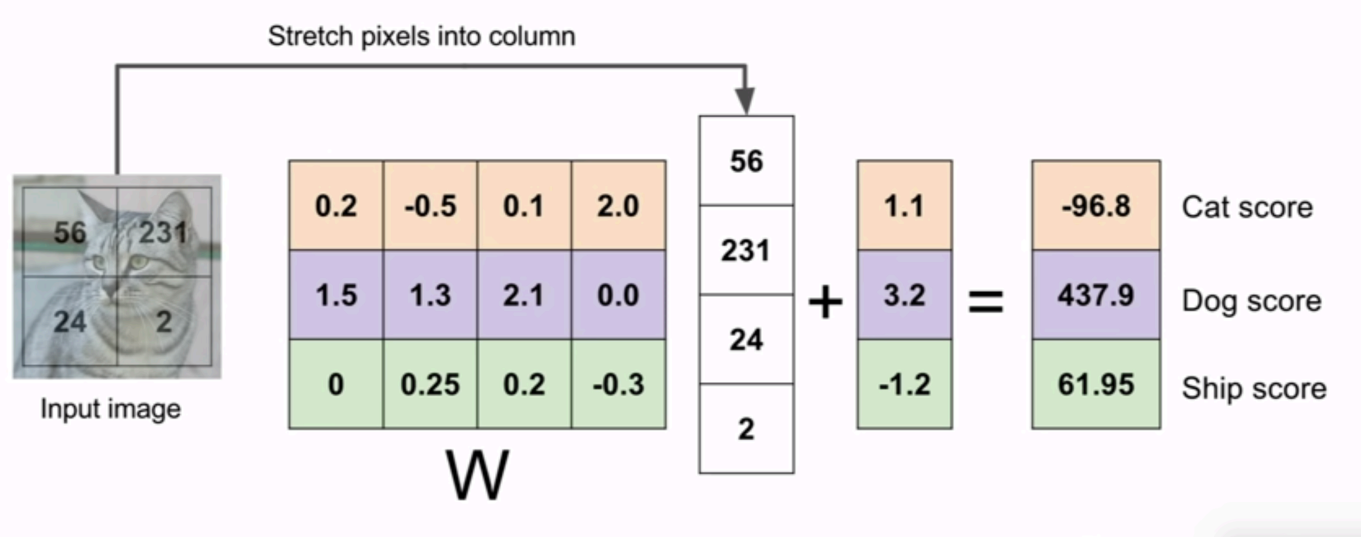


Image credit: Stanford CS 231n

Trained Weights of a Linear Classifier on CIFAR10

$$f(x, W) = Wx + b$$

only allowed to learn one template for category

Image credit: Stanford CS 231n

Outline

CNN Architecture Overview

The major advances in this field were made by a series of papers that won the ImageNet competition, starting in 2012

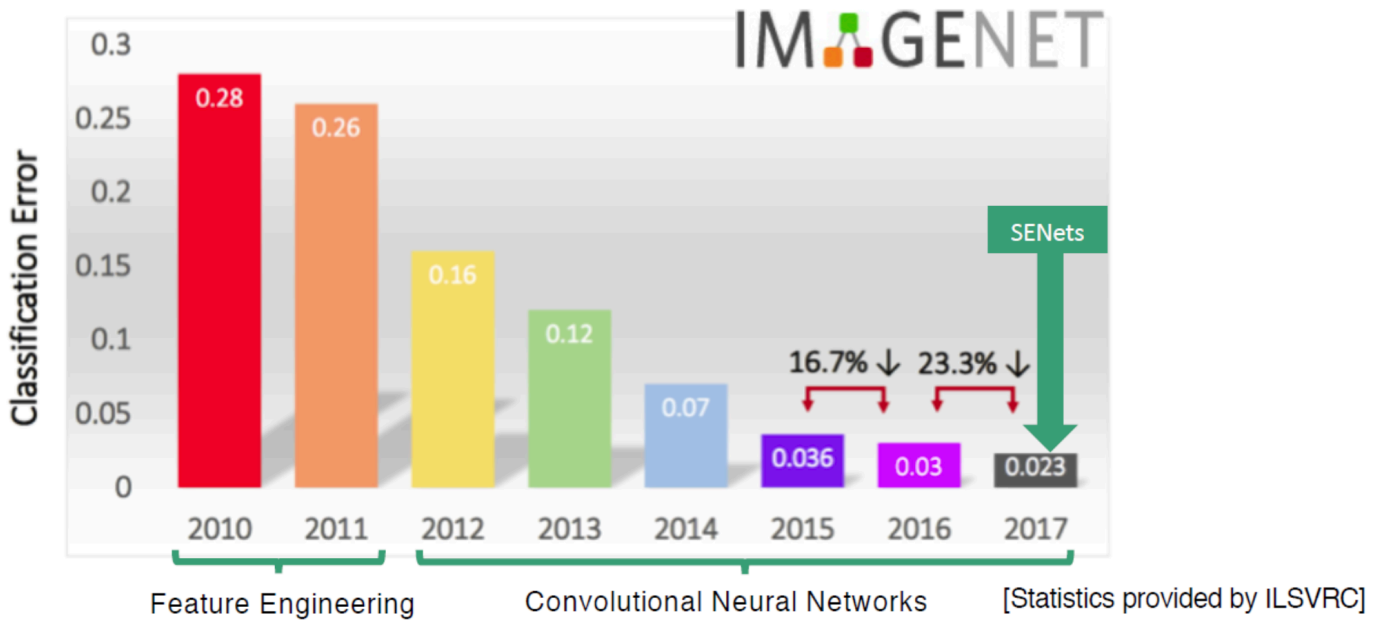
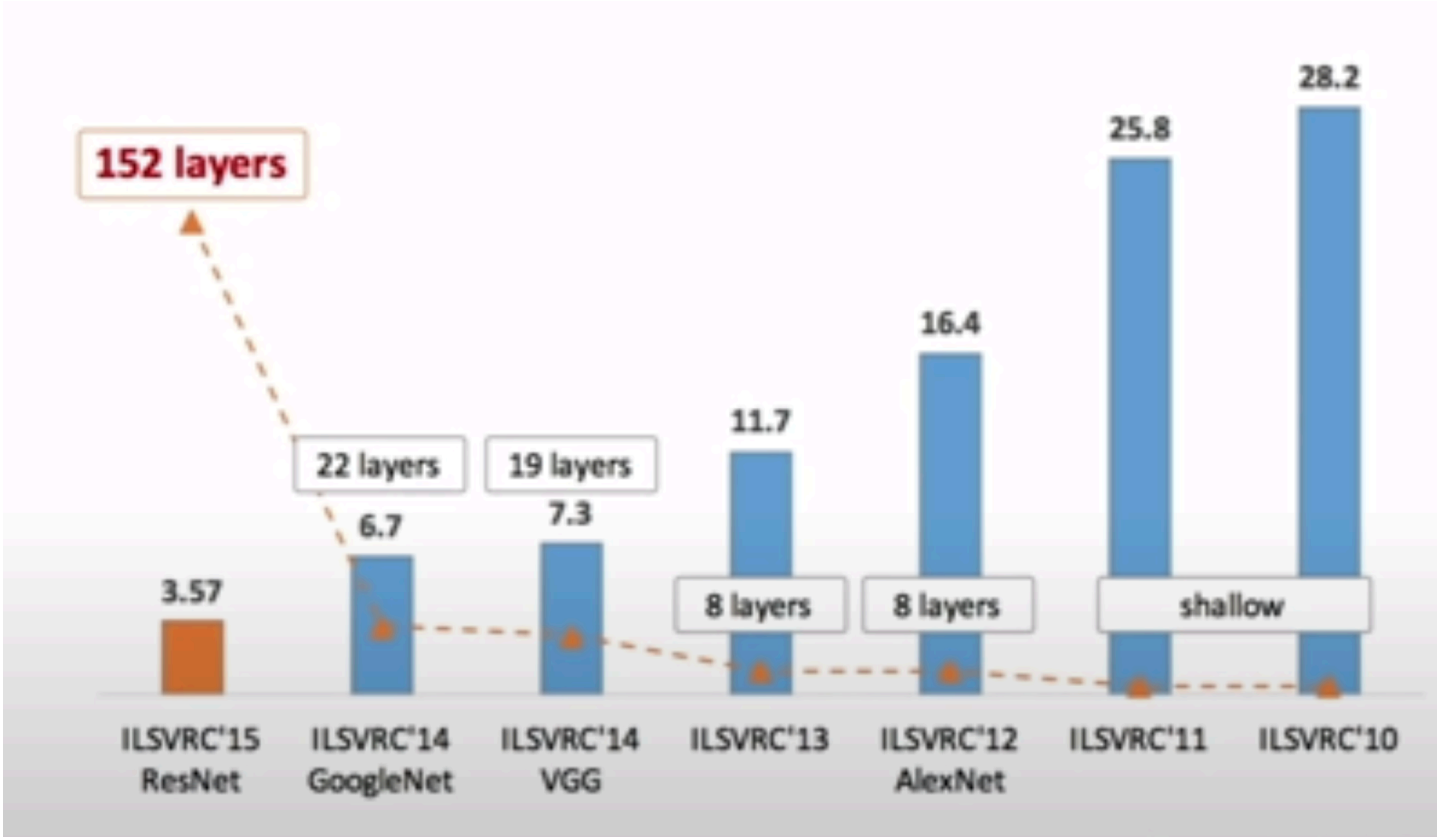


Image from kaggle.com

CNN Comparisons



CNN Comparison

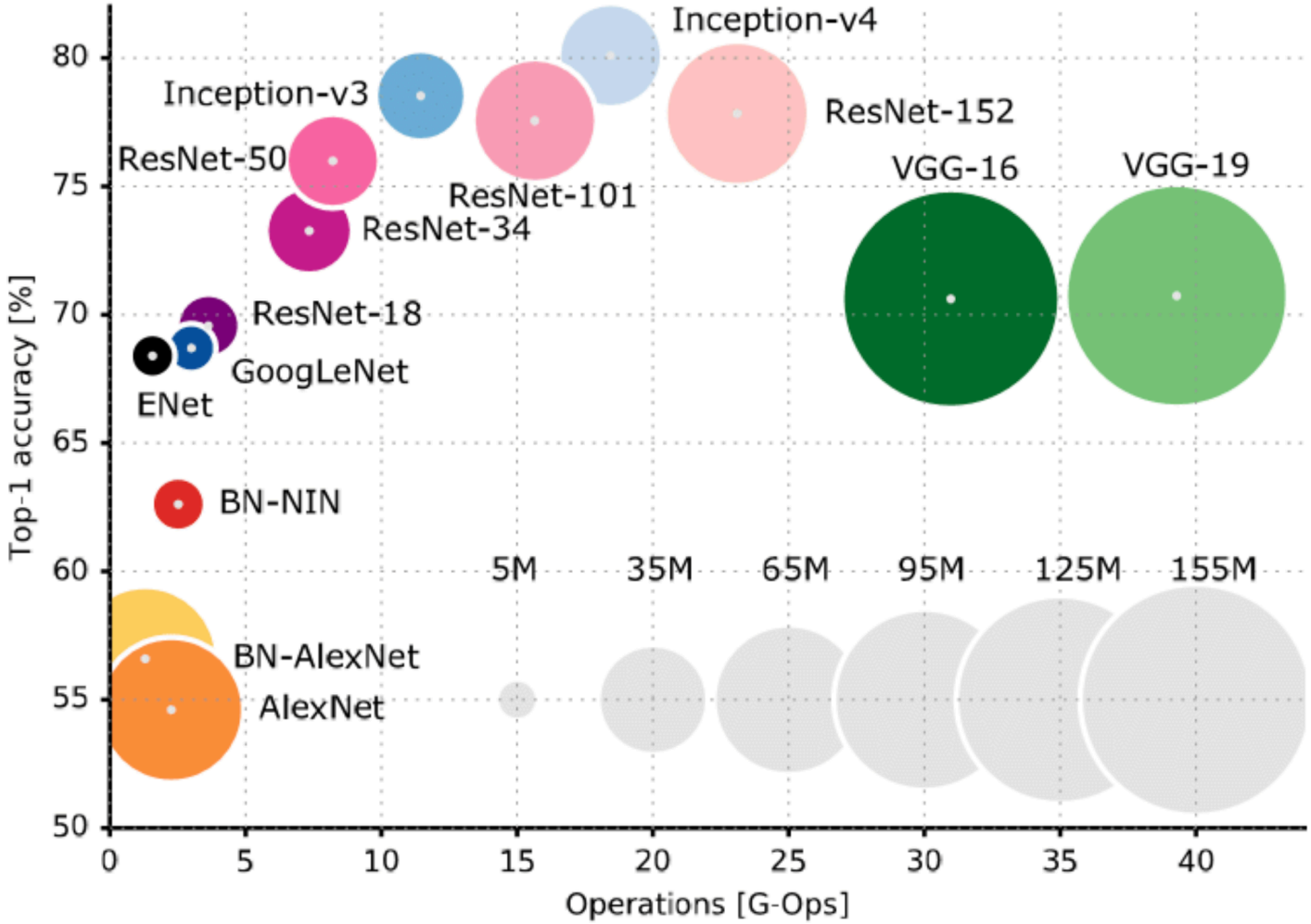
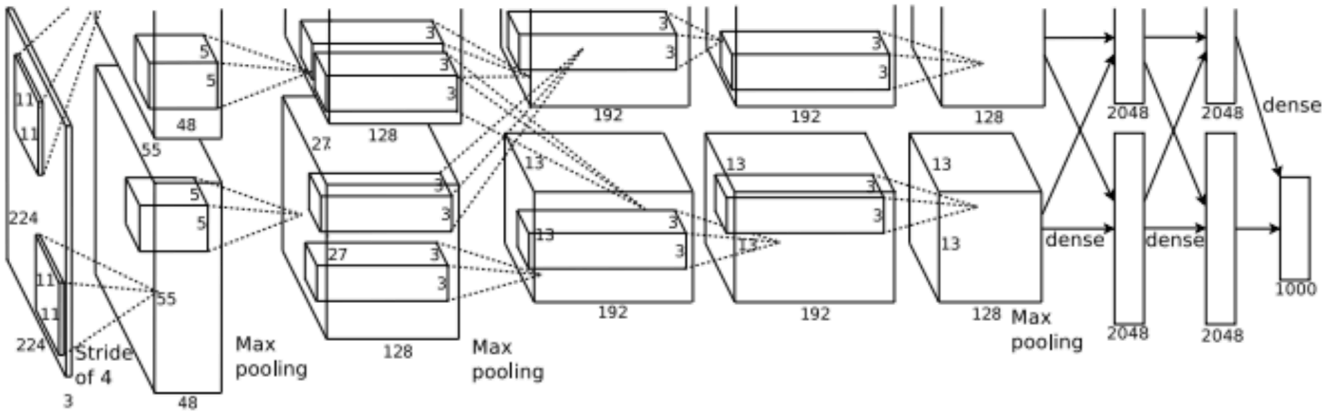


Image from Deep Learning: Decade in Review

Outline

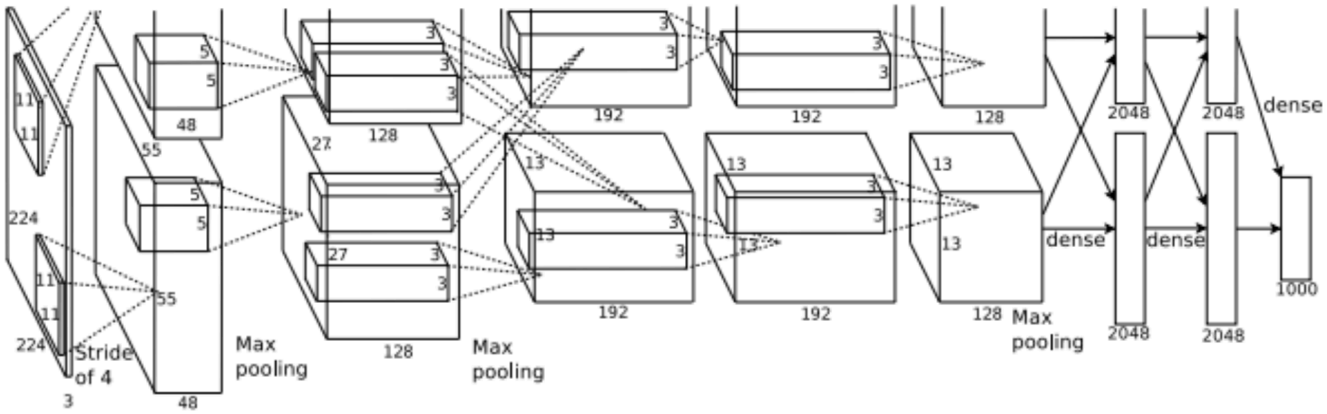
AlexNet

AlexNet won the ImageNet challenge in 2012, and was the first CNN-based paper to do so. It dramatically lowered the error rate (from 26.2% to 15.3%) and is one of the most cited papers in deep learning.



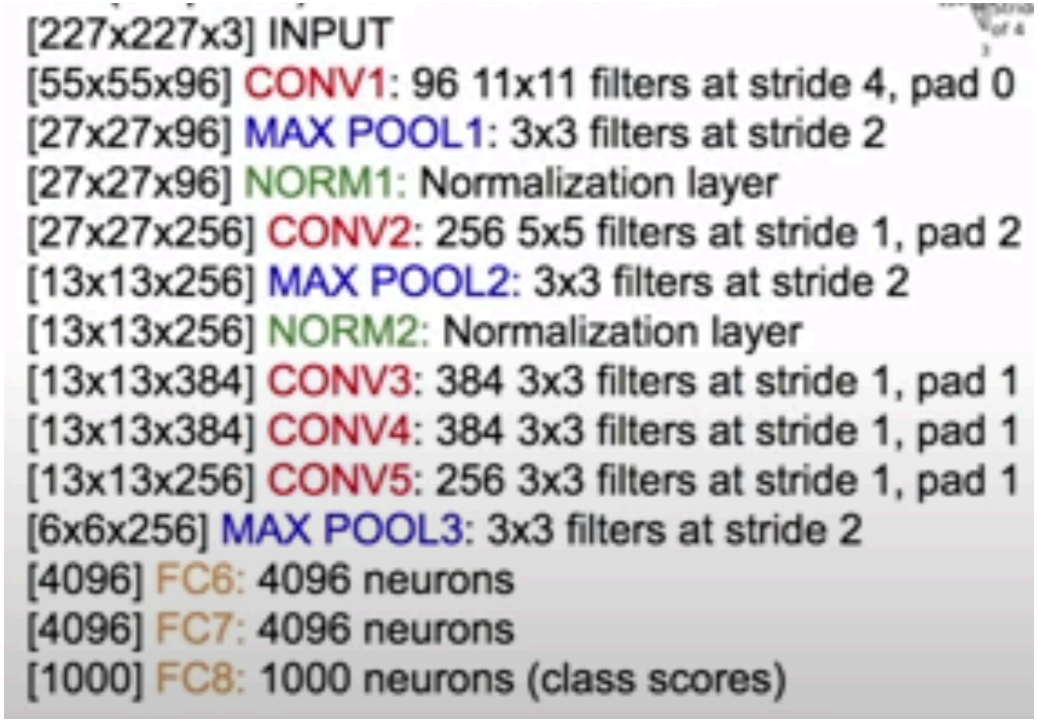
Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton

AlexNet



1. CONV1 MAXPOOL1 NORM1
2. CONV2 MAXPOOL2 NORM2
3. CONV3
4. CONV4
5. CONV5 MAXPOOL3
6. FC6
7. FC7
8. FC8

Details of the Architecture



Details of the Model

- AlexNet was an early adopter of ReLU, which allowed the network to be trained much, much faster
- Also made heavy use of data augmentation and used dropout to avoid overfitting, which helped improve performance
- One of the first papers to leverage the massively parallel processing power of GPUs, which allowed it to train a much deeper neural net
- Built upon ideas in LeNet, developed by Yan LeCun and others in the late 1990s

Outline

VGG

key idea: deeper network, smaller filters: 3 x 3 convs with periodic 2 x 2 max pooling (7.3% top 5 error)

VGG

- Insight: if you use smaller filters in deeper networks, have same effective receptive field
- Suppose you are using 3 x 3 filters throughout, with stride 1. In the first layer, it has a receptive field of 3. In the second layer, the effective receptive field is 5, because each of the neurons it is applied to contains information from the neighboring pixels. A 3 x 3 filter applied to this output is similarly looking at 7 x 7 in the input layer.
- Hence, same receptive field as 7 x 7 conv layer, but more nonlinearity; also fewer parameters

VGG 16 Architecture

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

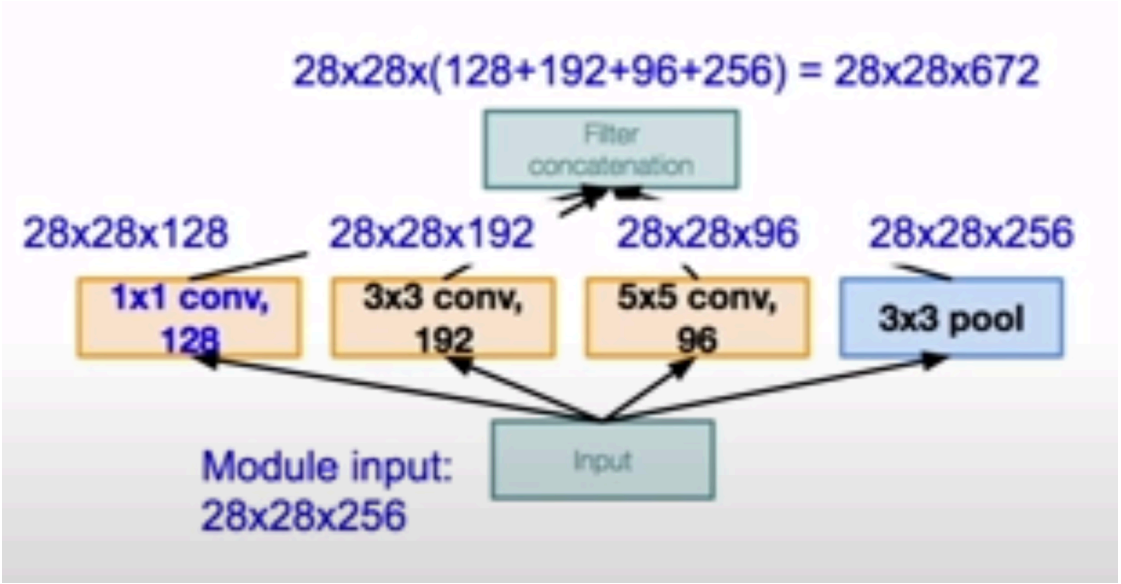
Outline

CoogLeNet (Inception)

- Very similar performance to VGG in 2014 ImageNet Challenge
- Got rid of fully connected layers - saves a lot of parameters
- Key feature is the inception model, a local network topology at each layer

Inception Module

- Idea is to apply several different feature operations in parallel and concatenate together
- If you were to do this naively, the number of parameters would quickly blow up: 854M conv ops alone, plus pooling. Pooling preserves the depth, so at each layer the feature depth can only grow



Making Inception Computationally Feasible

Key insight: bottleneck layers that project features maps to lower dimensions using a 1 x 1 convolution - preserves spatial dimension while reducing depth by taking a dot product at each spatial location (linear combination of input feature maps)

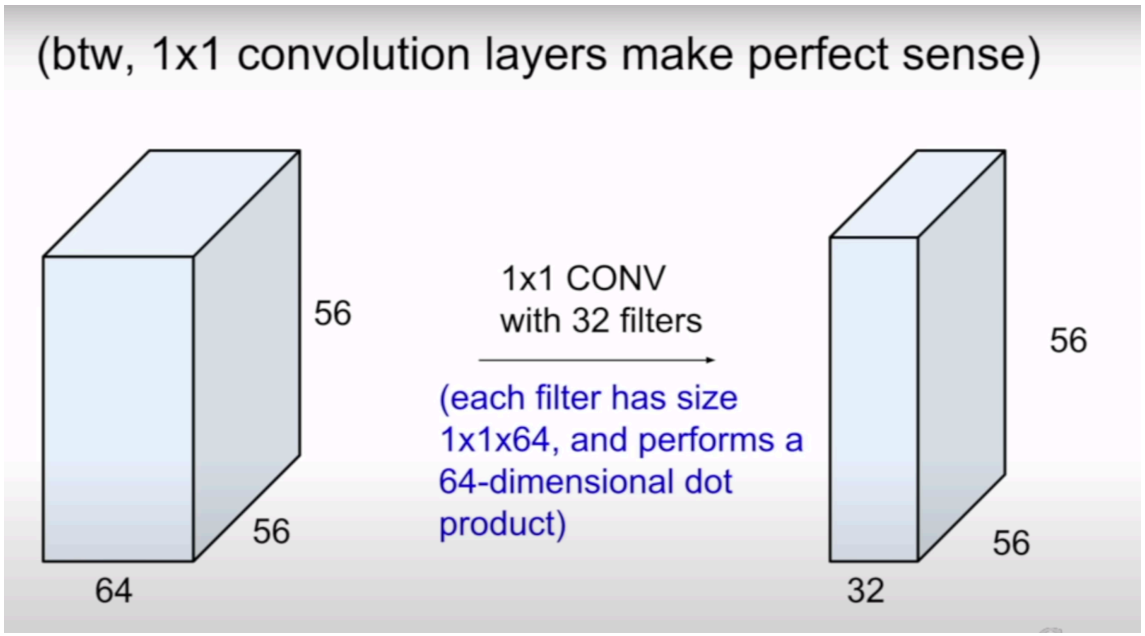
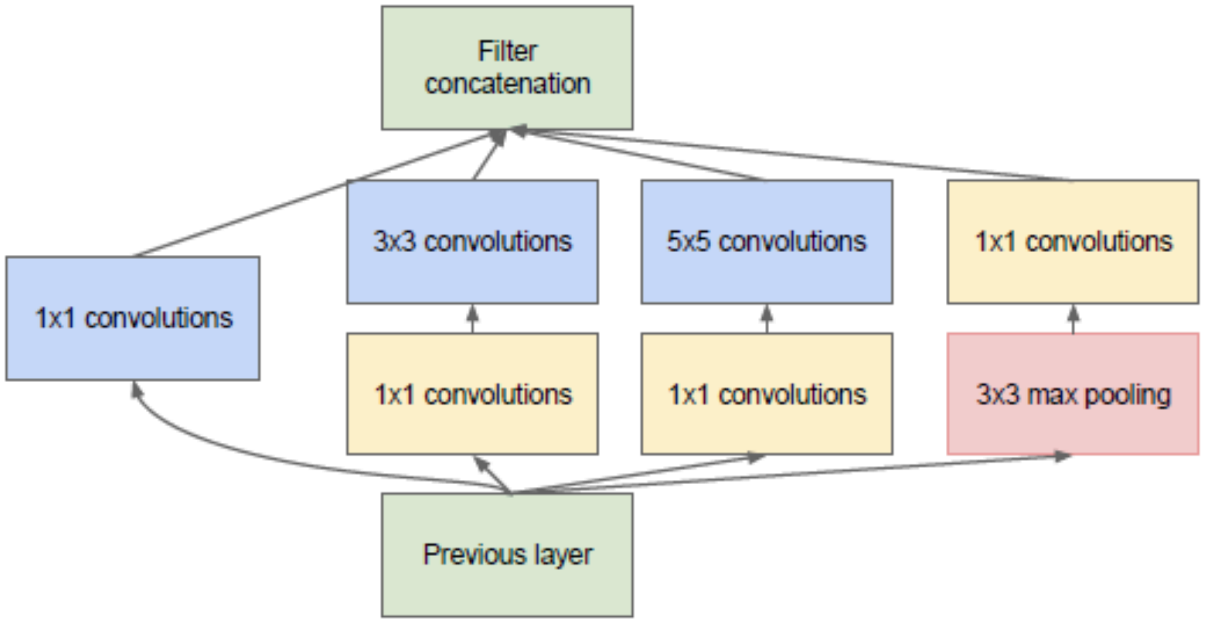


Image credit: Stanford CS 231n

Inception Module with Dimension Reduction

Apply 1 x 1 x 64 convs before 3 x 3 and 5 x 5 conv opps and after pooling

Reduce conv ops to 358 M (from 854 M) and output from 672 to 480 depth



Szegedy et al., 2014

Intuition

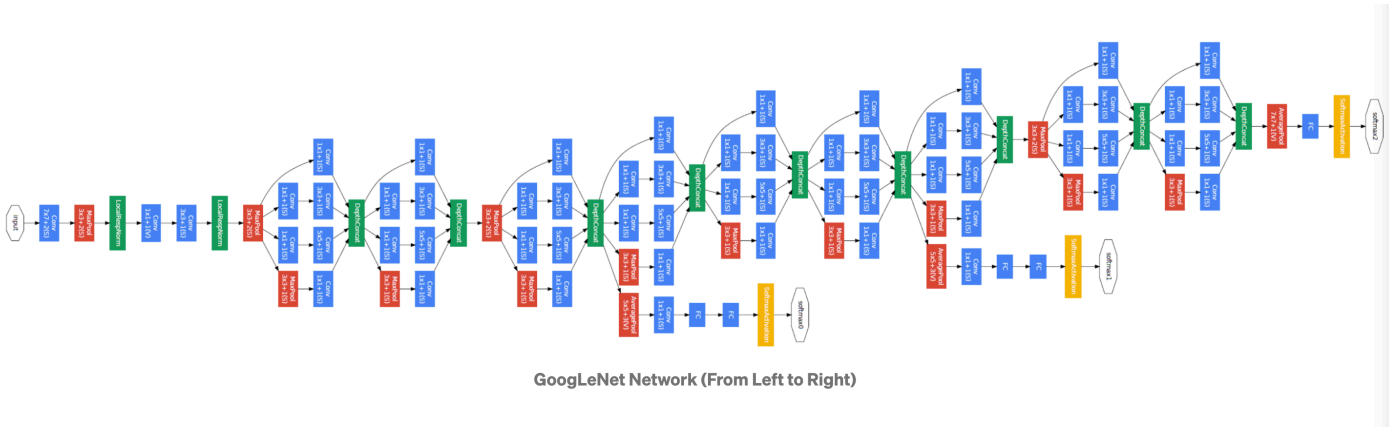
- Do lose some information by taking a linear combination of the feature layers with the 1 x 1 conv, but some redundancy in them
- The additional non-linearity from applying ReLU after the 1 x 1 conv is beneficial

Getting Rid of the Fully Connected Layers

- Suppose at the second to last layer you have 1024 7 x 7 features maps; you want to use this to predict a 1 x 1024 layer of class probabilities. This implies 7 x 7 x 1024 x 1024 =51.3M weights if fully connected
- GoogLeNet applies global average pooling at the end of the network, by averaging each of the 1024 7 x 7 features maps into a 1 x 1 features map, to produce a 1024 x 1 output. This has zero parameters
- Even better, the authors found that a move from FC layers to average pooling improved top-1 accuracy by 0.6%, by reducing overfitting

Full Architecture

22 layers!



GoogLeNet Network (From Left to Right)

Initial modules are some vanilla conv/pool layers

Also two auxiliary (avg pool - 1 x 1 conv - FC- FC - softmax) layers used for training (not testing or inference); intuition is that the middle layers of the network should do reasonably well at classification as well. Having these auxiliary classification outputs gets more gradient training injected at the earlier layers.

Full Architecture

Why this specifically?

- The intuitions we shared - i.e. combining these networks within a network using 1 x 1 convs plus not needing the fully connected layers; need auxiliary layers to deal with vanishing gradient problem
- Combined with the fact that Google (the author) has massive compute and can do very extensive cross-validation, this happened to work best....

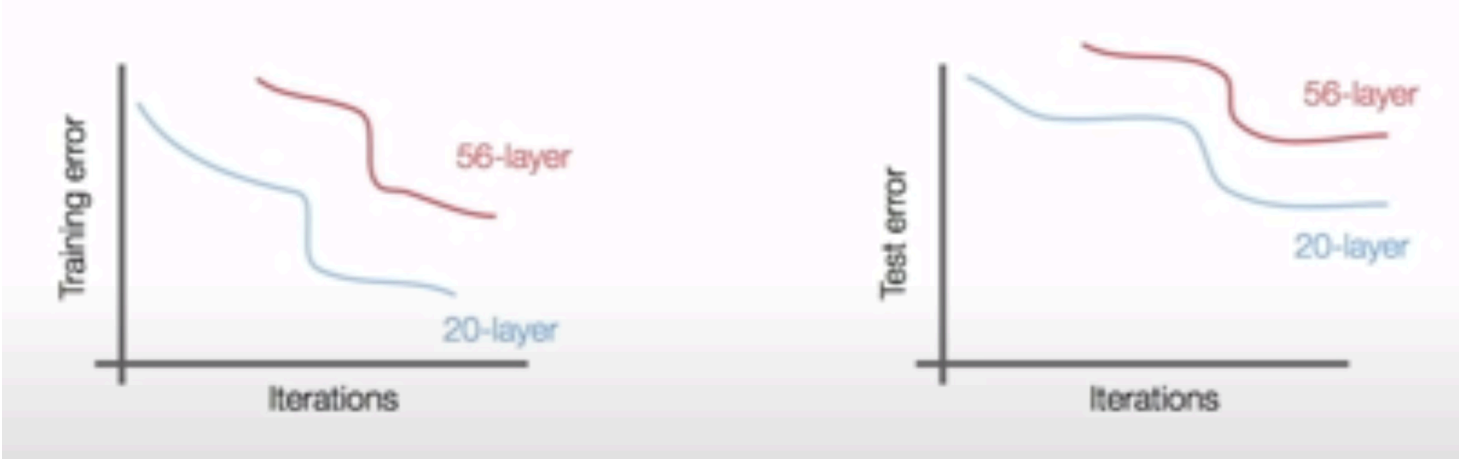
Outline

ResNet

- Up to 152 (!) layers
- 3.57% top 5 error; swept all benchmarks in ImageNet (classification, localization, detection), also won COCO (detection, segmentation)

Getting Deeper and Deeper

In a normal CNN, can we just keep adding more and more layers to improve accuracy? No!



He et al., 2015

Doing worse in both test and train - not caused by overfitting... It's an optimization problem

Optimizing Deep Neural Nets

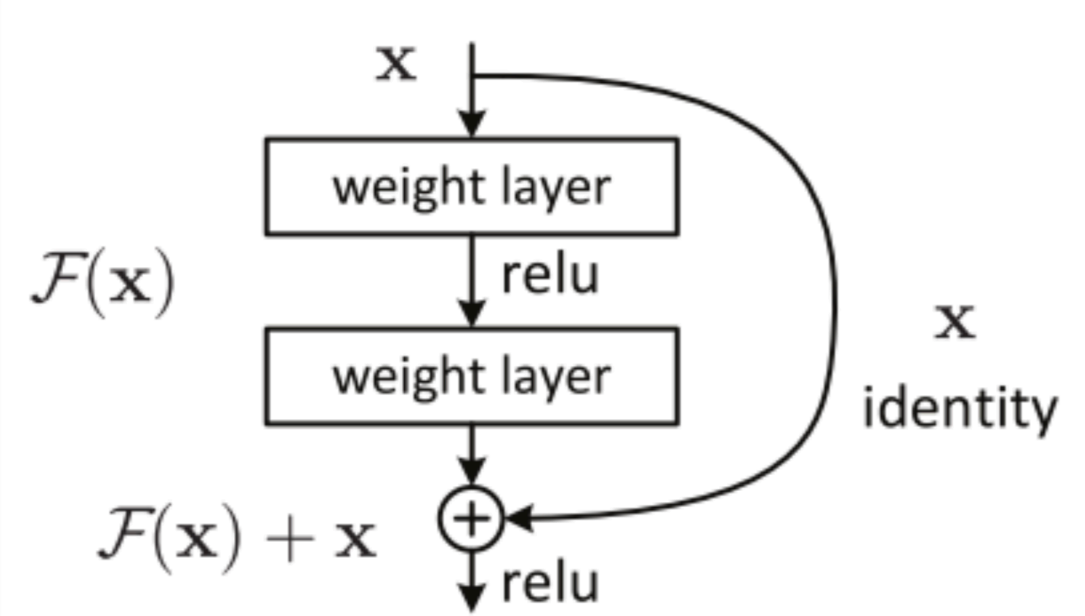
- Absent optimization and overfitting problems, deep neural nets should be able to do just as well as shallower nets
- Could just take the learned layers from the shallower model and set additional layers to the identity mapping

Vanishing Gradients

- As the gradient is back-propagated to earlier layers, repeated multiplication may make the gradient infinitely small.
- Makes learning very slow. In the extreme, if the gradient is zero can't update the parameters at all.
- Auxiliary loss layers in the middle of GoogLeNet add additional supervision and are meant to address this problem, allowing GoogLeNet to get as deep as 22 layers
- Still, didn't scale to even deeper networks

Optimizing Deep Neural Nets

Instead of choosing weights to fit some desired function $H(x)$, with input layer x, choose the weights to fit the residual $F(x) = H(x) - x$. Then can construct output $F(x) + x$. If choose weights zero, $H(x) = x$, so should be able to do at least as well as the shallower net.



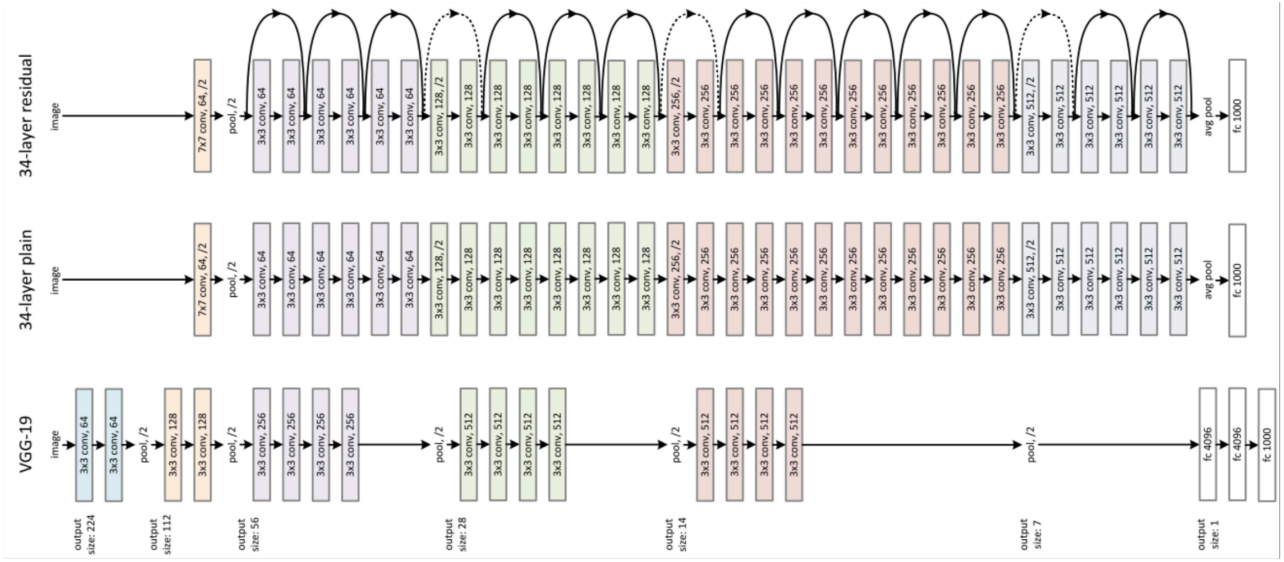
a residual block

$$F(x) = W2 * relu(W1 * x + b1) + b2$$

Optimizing Deep Neural Nets

- Residual mappings make it easier to learn an identity mapping
- Helps address the vanishing gradient problem: what if we were to backpropogate through the identity function? The gradient would be multiplied by 1.
- Intuition: if you have a very deep network, should need just small tweaks at each layer (or perhaps nothing)
- Residual blocks have become the elementary building block for almost all CNNs today

ResNet Architecture

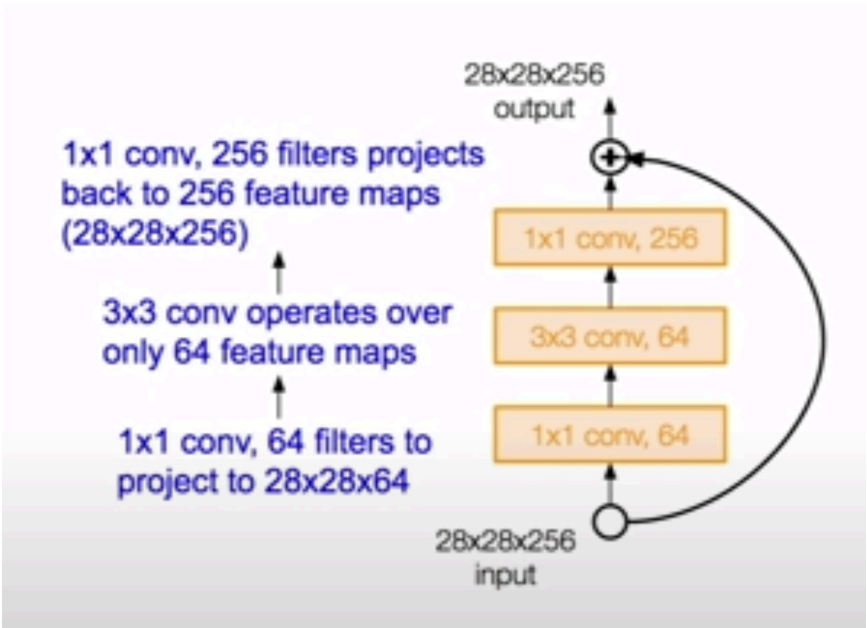


He et al., 2015

Every residual block has 2 layers of 3 x 3 filters; periodically double number of filters and downsample spatially using stride 2; no FC layers at end (global average pooling after last conv layer)

Bottleneck Layers

For ResNet50+, use bottleneck layers to improve efficiency (like GoogLeNet)



Outline

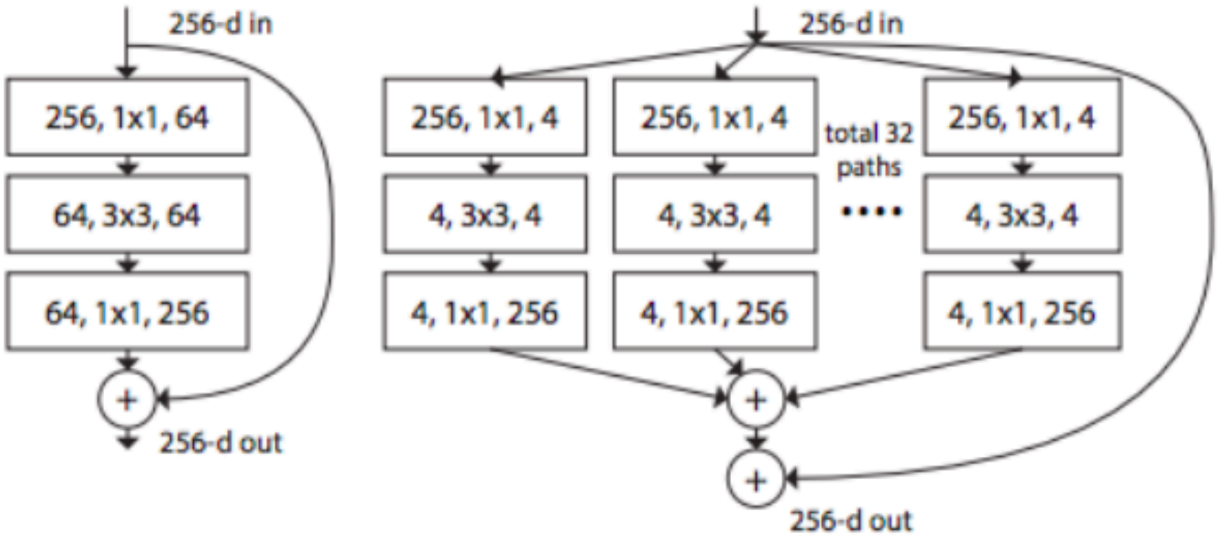
ResNet v. Inception

GoogLeNet has three central properties:

- Bottleneck - reduce dimensions before conv layers
- Shortcuts to go deep - i.e. auxiliary training layers
- Multiple Branches - extracting features of various sizes by using multiple filters in parallel

ResNet has the first two (dramatically improves on going deep) but not the third (i.e. depth but not width). This led to ResNeXt...

ResNeXt Blocks



CNN Comparison

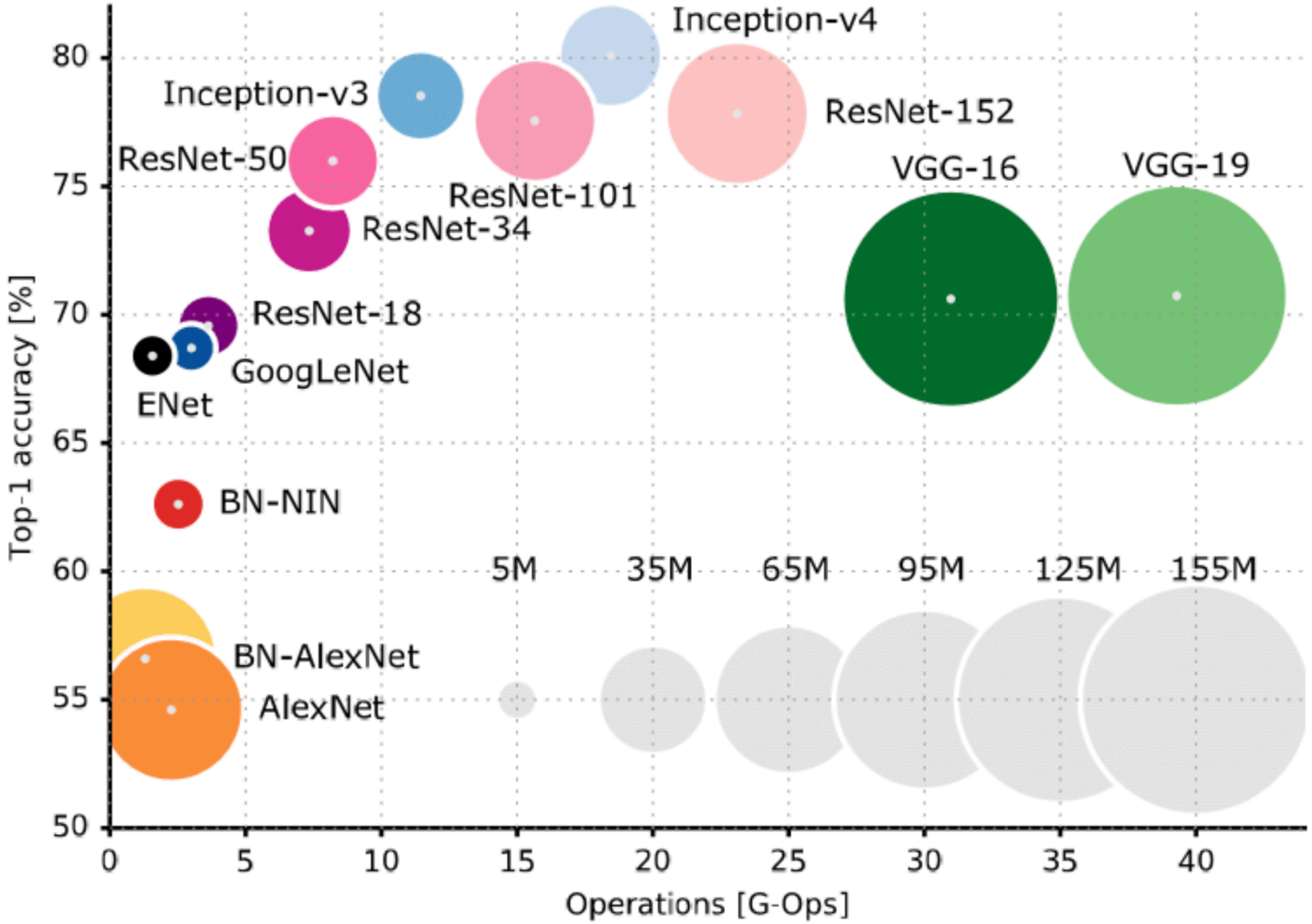


Image from Deep Learning: Decade in Review