Agenda

Concepts revision

Relook at CNN convolution in depth to understand parameters in network

CNN architectures introduction

Functional api

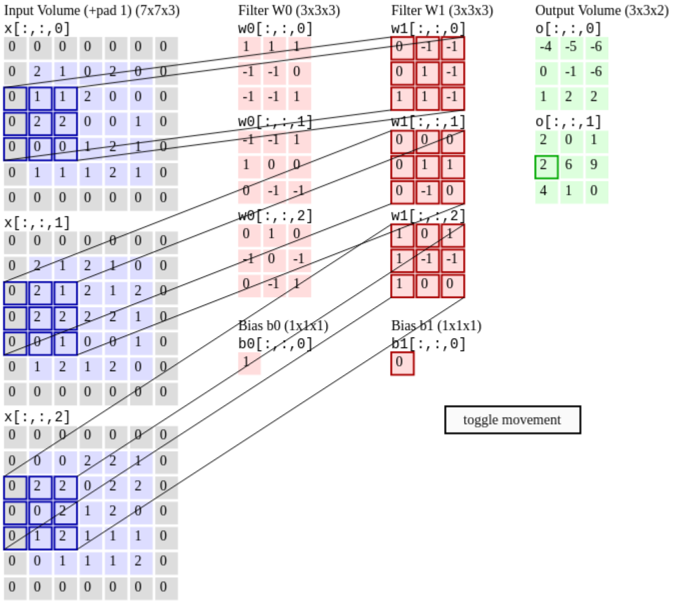
VGG code

Advanced Training Techniques

Further Steps

Convolution

32x32x3 3x3x8 -> 30x30x8



I = 32x32x3

I1 I2 I3

F1 F2 F3

I1xF1 = O1

I2xF2 = O2

I3xF3 = O3

O = O1 + O2 + O3

32x32x3 * (3x3x3)x8 -> 30 x 30 x 8

O = I1xF1 + I2xF2 + I3xF3

28x28x3 * 3x3x3 -> 26x26

28x28x3 * (3x3x3)*16 -> 26x26x16

26x26x16 * (3x3x16) *32 -> 24x24x32

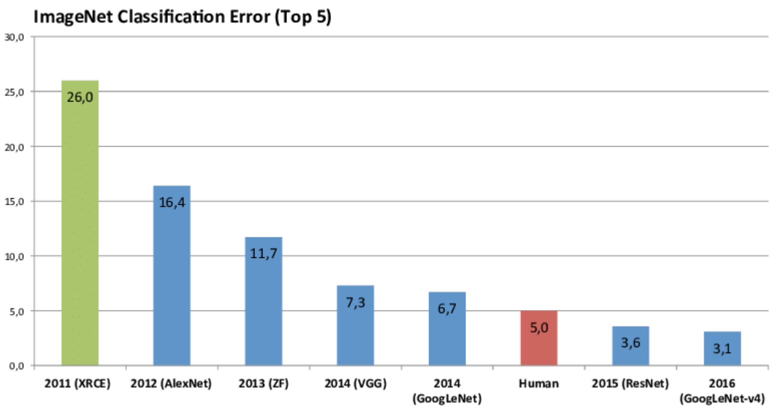
64x64x3 -> 62x62x16

(3x3x3)x16

64x64x3 -> 62x62x16 . How many params ?

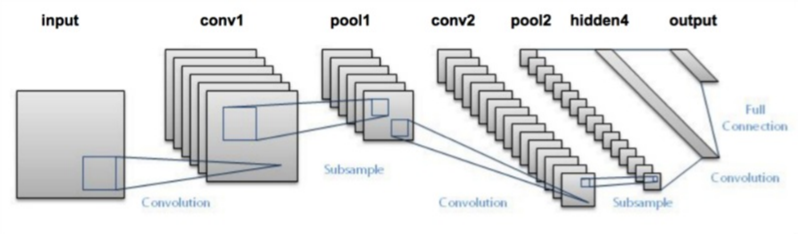
62x62x16 -> 60x60x32. How many params ?

(3x3x16)x32



CNN architectures

Lenet-5(1998)

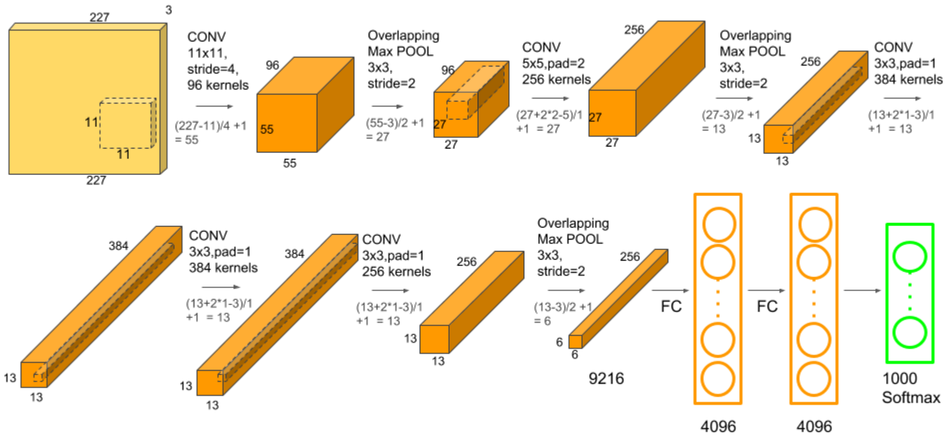


Tanh activation function used

5x5 convolutions used

Limited use due to lack of computational power

Alexnet(2012)



(n-k)/s + 1

Stood 1st in Imagenet Challenge in 2012 reducing the top-5 error from 26% to 15.3%

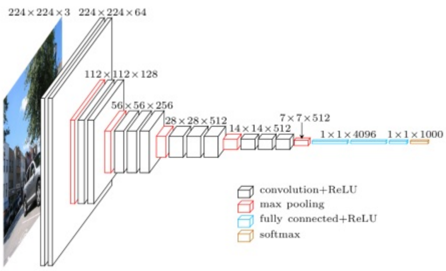
Relu was introduced

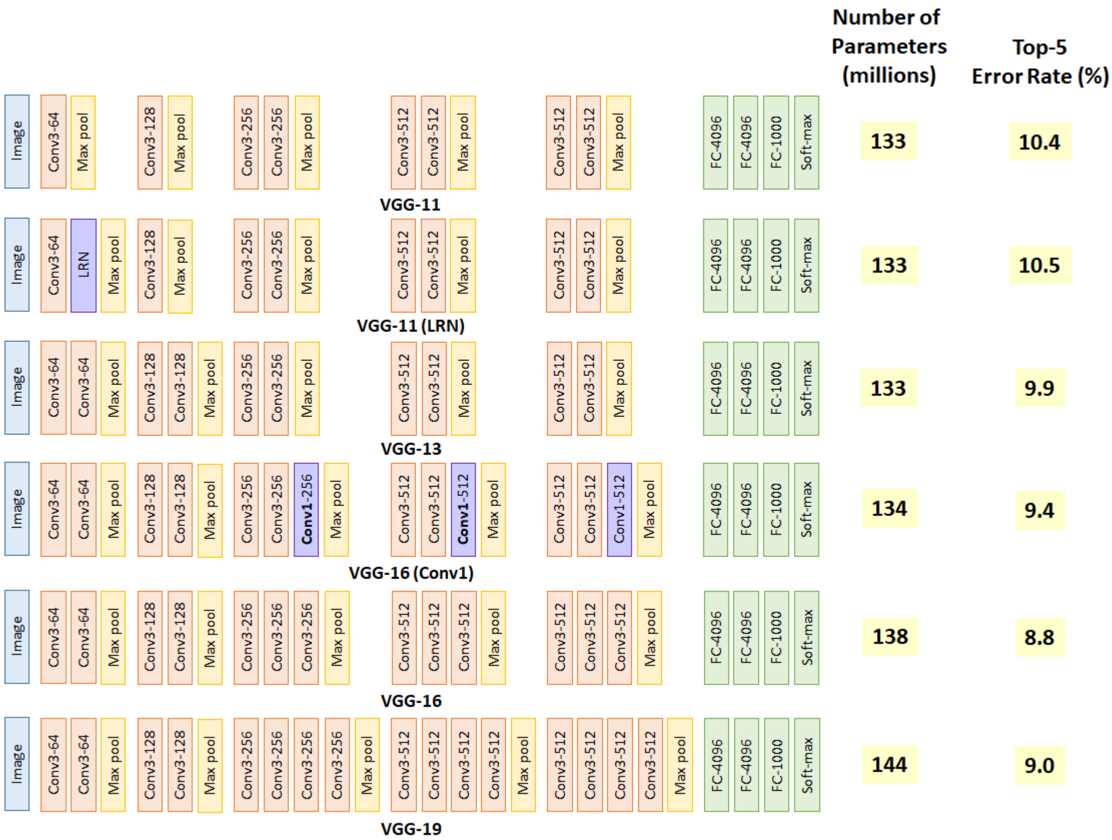
11x11, 5x5, 3x3 convolutions used

Used dropout regularization

Overlapping pooling used i.e 3x3 pooling with stride 2

VggNet(2014)





Only 3x3 convolutions used

Simple architecture followed throughout

Was used as a feature extractor in many use cases due to its effectiveness

In 2014, 16 and 19 layer networks were considered *very* deep. Training VGG16 and VGG19 was challenging (specifically regarding convergence on the deeper networks), so in order to make training easier, first *smaller* versions of VGG with less weight layers are trained first. The smaller networks converged and were then used as *initializations* for the larger, deeper networks — this process is called ***pre-training***.

Pre-training is a very time consuming, tedious task, requiring an *entire network* to be trained **before** it can serve as an initialization for a deeper network. Pre-training is no longer used (in most cases) and instead prefer Xaiver/Glorot initialization or MSRA initialization is used.

Unfortunately, there are two major drawbacks with VGGNet:

1. It is *painfully slow* to train.
2. The network architecture weights themselves are quite large (in terms of disk/bandwidth).

Due to its depth and number of fully-connected nodes, VGG is over 533MB for VGG16 and 574MB for VGG19. This makes deploying VGG a tiresome task.

We still use VGG in many deep learning image classification problems; however, smaller network architectures are often more desirable (such as SqueezeNet, GoogLeNet, etc.).

Functional api

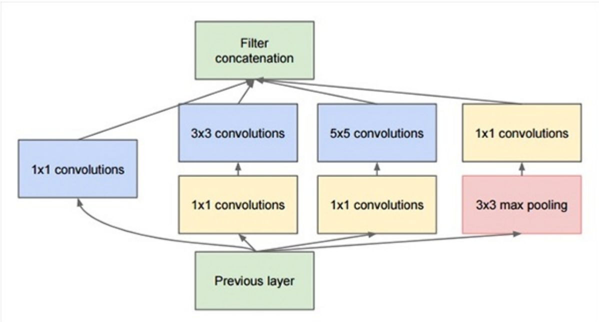
The **sequential** API allows you to create models layer-by-layer for most problems. It is limited in that it does not allow you to create models that share layers or have multiple inputs or outputs.

Alternatively, the **functional** API allows you to create models that have a lot more flexibility as you can easily define models where layers connect to more than just the previous and next layers. In fact, you can connect layers to (literally) any other layer.

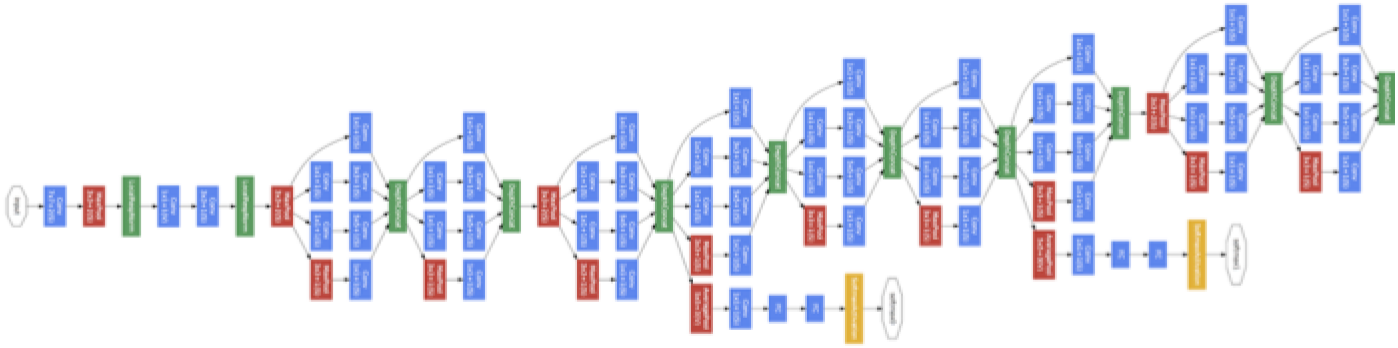
**Inception(2014) **

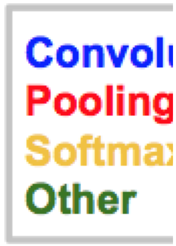
nxnx32 * (3x3x32) x 64 -> (n-2)x(n-2)x64 -> Params = 3x3x32x64 = 18432

nxnx32 * (1x1x32) x 8 -> nxnx8 * (3x3x8) x 64 -> (n-2)x(n-2)x64 -> Params = 1x1x32x8 + 3x3x8x64 = 256 + 4608 = 4864



Adding multiple layers is facilitated by appropriate padding





Multiple versions of the architecture are released Inceptionv1, Inceptionv2, Inceptionv3

Has multiple paths in a block and hence can act as a multi-feature extractor

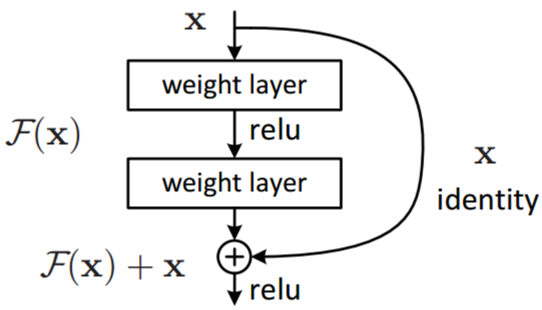
Multiple receptive field helps in capturing different sizes of objects

Also label smoothing was introduced here

Resnet(2015)

Before Resnet, deeper networks had lesser accuracy after a certain depth which is counterintuitive

The problem is vanishing gradient

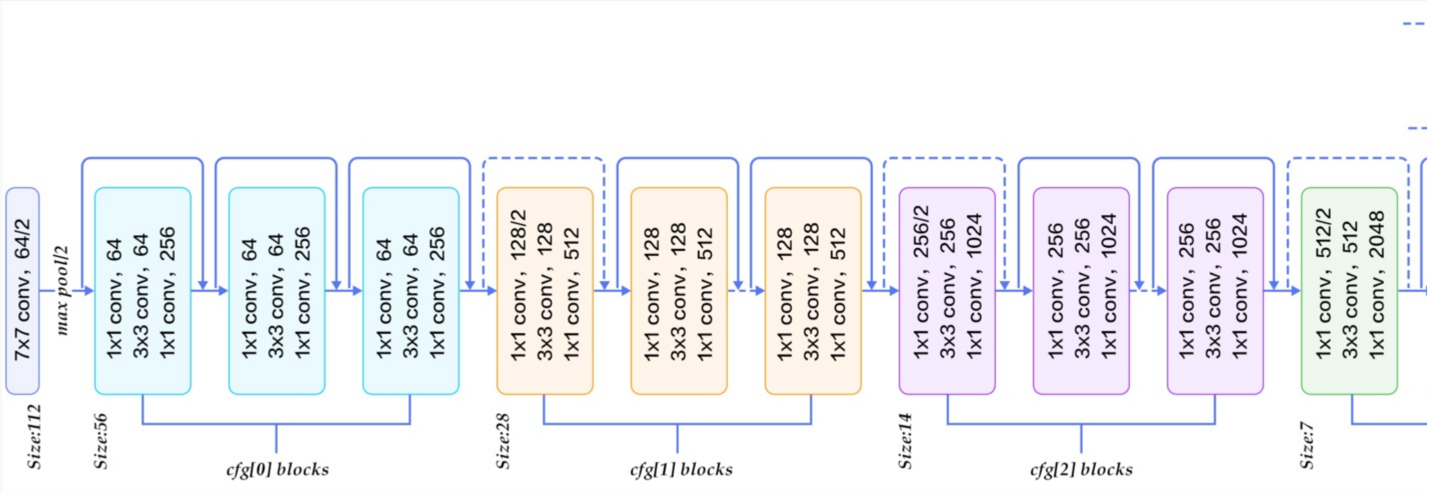


Implemented residual connections

Helped in training deeper neural networks

o2 = F(o1)

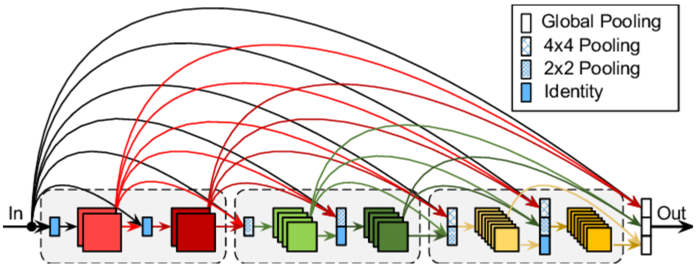
o2 = F(x1) + x1



DenseNet(2016)

Inspired from Resnet

Connections from every layer to every other layer in a block



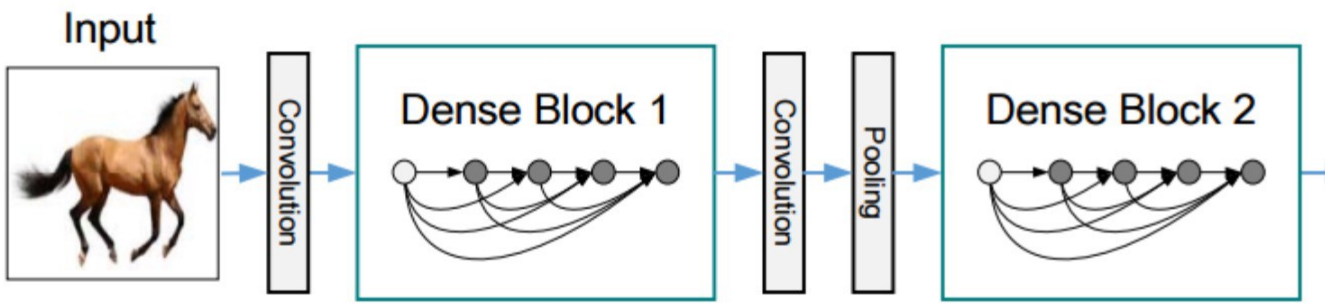


Figure 2. A deep DenseNet with three dense blocks. The layers between two ad
feature map sizes via convolution and pooling.

Mobilenet

For mobile phones

Advanced techniques

LR reduce with Plateau

Reduce learning rate whenever there is a plateau in validation loss

Finding best lr to start with is a tough concept. Fixing a single learning rate can result in getting stuck in local minima

LR Scheduler

0-50 0.01

50-100 0.001

100-200 0.0001

Schedule fixed learning rate at certain epochs using a callback

For example keep learning rate 0.1 at start and drop to 0.01 at epoch 40 and drop to 0.001 at epoch 100

Checkpointing

Colab noetbook can get disconnected from time to time and you might lose your work

Your best accuracy model might not be the one you reach at end of training

Save your model after every epoch to drive

Transfer Learning

Use a network trained on a huge similar dataset as starting point and fine-tune the network afterwards

Further Steps

Blog and publish regularly on Medium, Linkedin

Create a portfolio of real-world projects

Continue learning advanced concepts in deep learning i.e object detection, segmentation etc

https://www.themtank.org/a-year-in-computer-vision

Do as many internships as possible