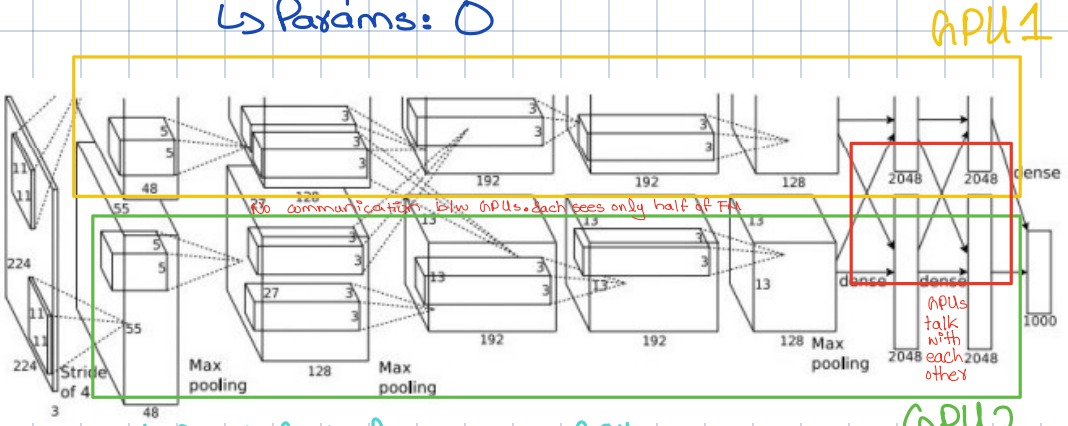
CNN Architecture

1) AlexNet: Error: 16.4% (2012)

CONV1 ⟶ 96 11x11 filters at stride 4
↳ Output Volume: 55x55x96
↳ Parameters: (11 * 11 * 3) * 96 = 35k

MAXPOOL 1 ⟶ 3x3 filters with stride 2
NORM 1 ↳ Output Volume: 27 x 27 x 96
CONV 2 ↳ Params: 0
MAXPOOL 2
NORM 2
CONV 3
CONV 4
CONV 5
MAXPOOL 3
FC 6
FC 7
FC 8

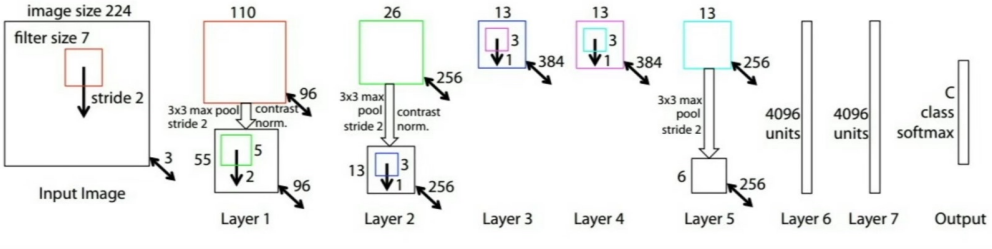


↳ Due to lack of memory on GPUs, feature maps were split into two

[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details:
↳ Used ReLU
↳ Used Norm layers
↳ Heavy data augmentation
↳ Dropout 0.5
↳ Batch size 128
↳ SGD Momentum 0.9
↳ LR: 1e-2, reduced by 10 when val accuracy plateaus
↳ L2 weight decay 5e-4
↳ 7 CNN ensemble: 18.2% → 15.4%

ZFNet: Error: 11.7 (2013)



Hyper tuned params of AlexNet

VGGNet: Error: 7.3% (2014)

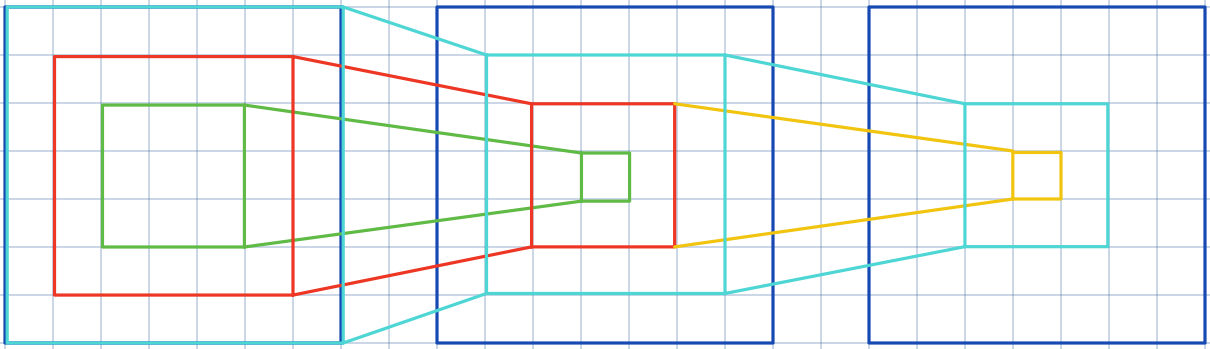
↳ Filter size: 3x3

Q. Why use smaller filters?

↳ Stack of three 3x3 filters conv (stride 1) layers has the same effective receptive field as one 7x7 conv layer
↳ Maximum area that a CNN can captured process

Q. What is the receptive field of three 3x3 conv (stride 1) layers?

↳



↳ So, effective receptive field of 3 3x3 filter is same as one 7x7 filter

↳ Less params also. $3 \times (3 \times 3 \times C \times C) = 27C^2$ vs $7 \times 7 \times C \times C = 49C^2$

(Total no. of FM)

(not counting biases)

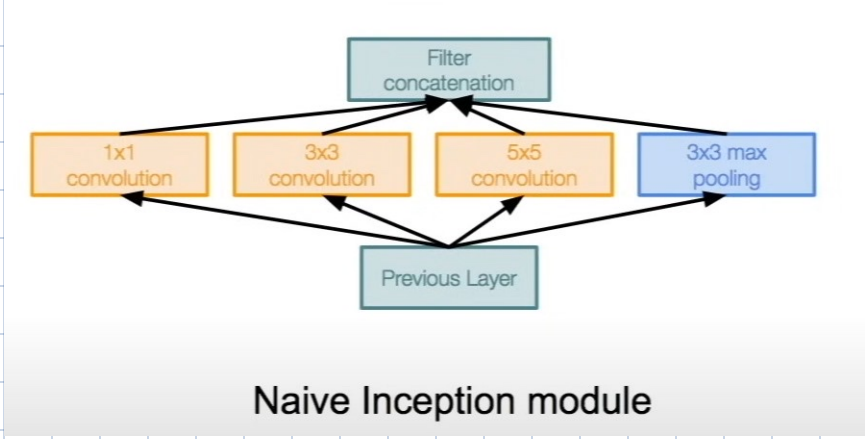
INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M 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

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

Details:
↳ 2nd Place
↳ No Local Response Norm
↳ VGG 16 or VGG19
↳ Use ensemble for best results
↳ FC7 good feature representation

GoogleNet:

↳ Used Inception modules



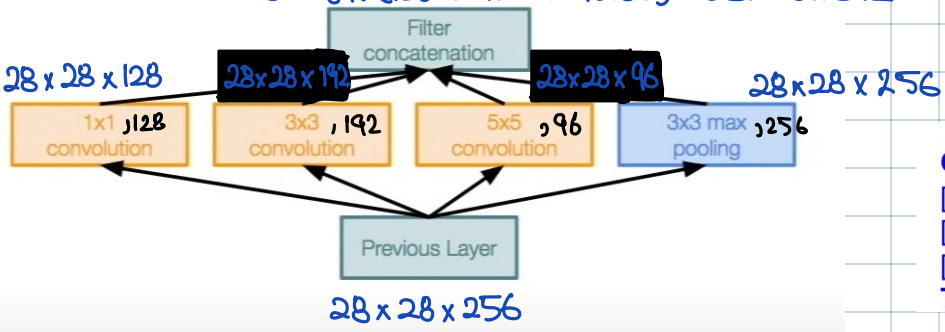
Naive Inception module

↳ Apply parallel filter operations on the input from previous layer.
  ↳ Multiple receptive field sizes for conv (1x1, 3x3, 5x5)
  ↳ Pooling op (3x3)
↳ Concatenate all filter outputs depth wise

Q. What's the issue?

↳ Computational complexity

28 x 28 x (128 + 192 + 96 + 256) = 28 x 28 x 672

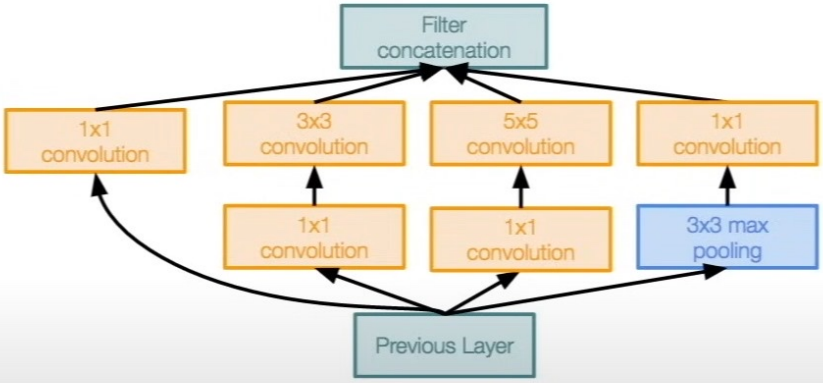


Naive Inception module

Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256
Total: 854M ops

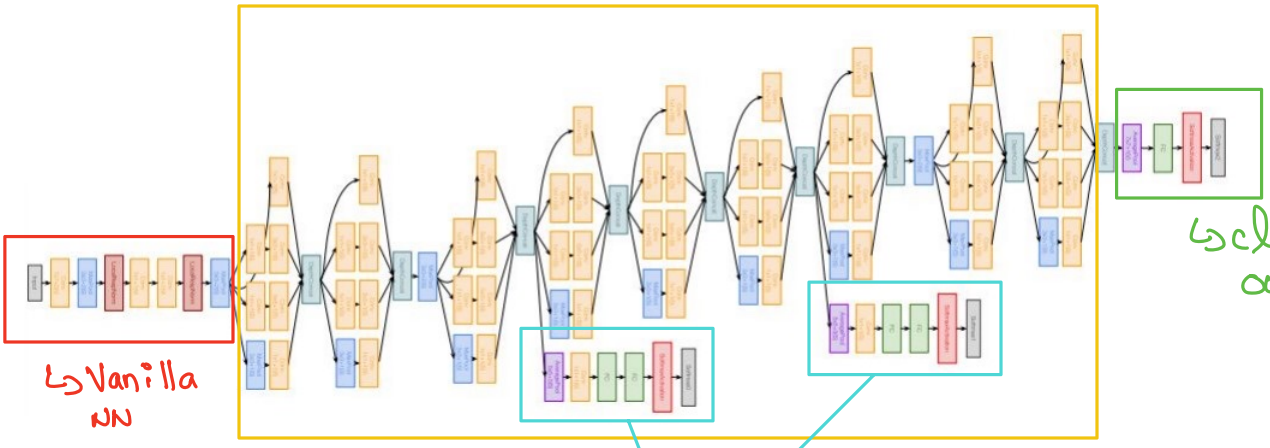
Solution:

↳ Project input depth to lower depth using 1x1 convs



Inception module with dimension reduction

Conv Ops:
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: 358M ops



↳ Vanilla NN

↳ classifier output

↳ stacked modules

Details:
- ↳ 22 layers
- ↳ Efficient "Inception" module
- ↳ No FC layers
- ↳ 12x less params than AlexNet
- ↳ 6.7% error

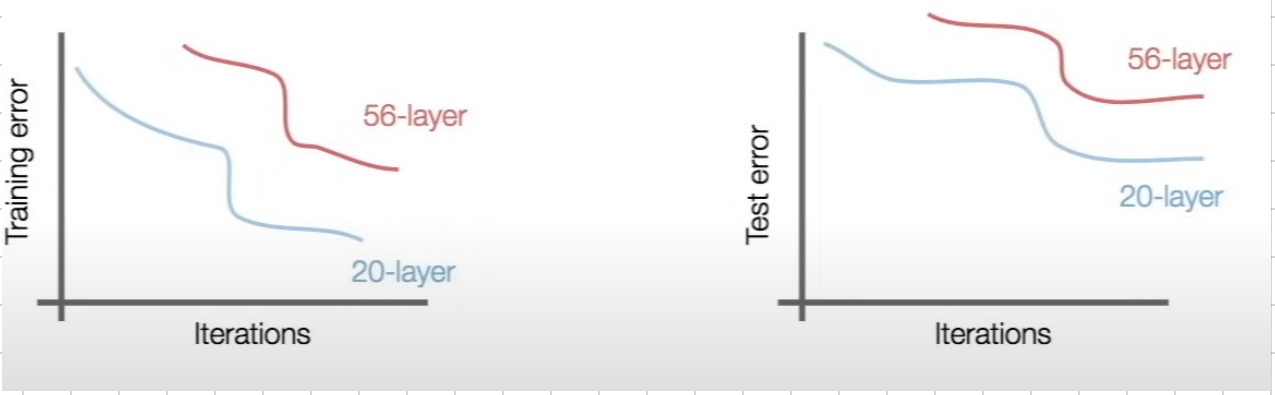
Aux classification outputs. Mini NN. Helps amplifying grads as the grads from end may end up becoming minimized when moving backward

ResNet: (2015)

↳ 152 layer model
↳ 3.57% error

Q. What happens when we continue stacking deeper layers on a plain convolutional neural network?

↳ No.



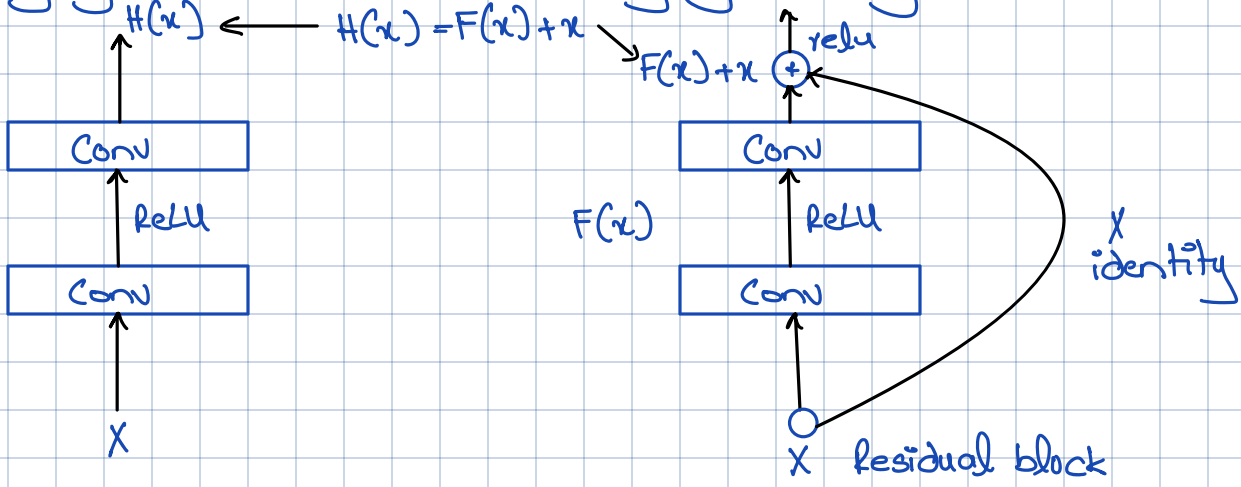
↳ Even though 56 layer model performs worse, it's NOT due to over-fitting

↳ The problem is an optimization problem

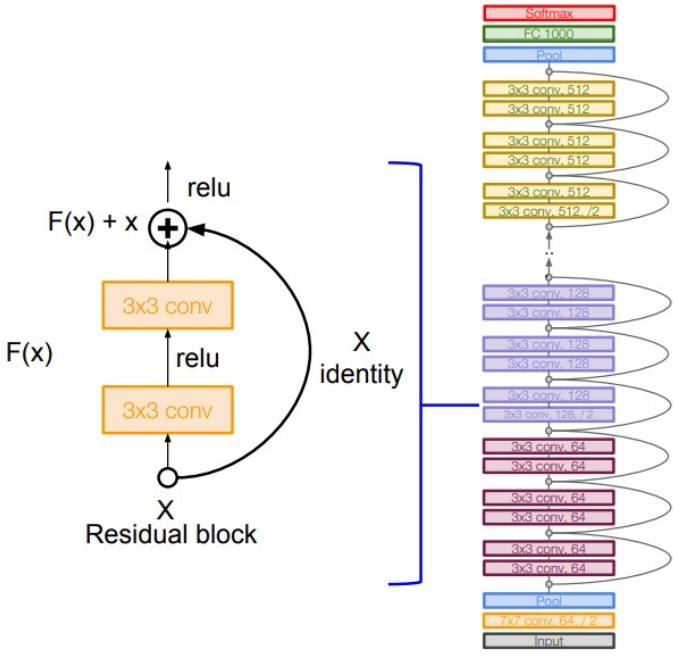
↳ A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping

Solution:

↳ Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



↳ Use layers to fit residual $F(x) = H(x) - x$ instead of $H(x)$ directly



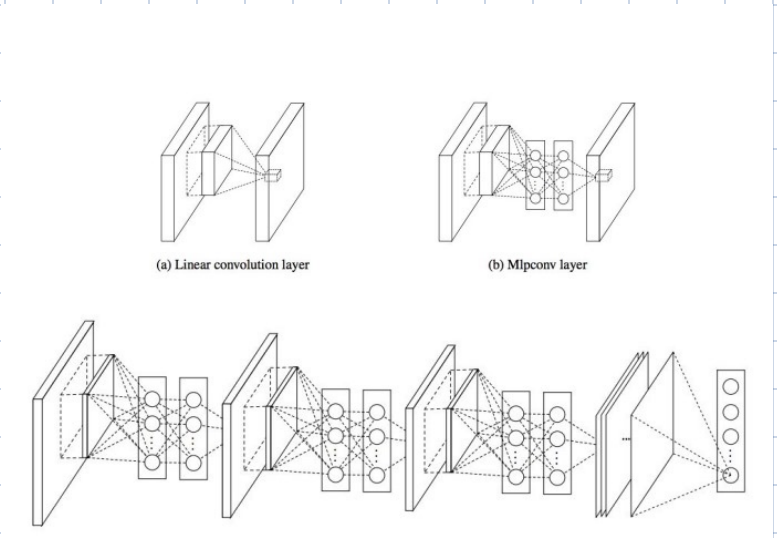
↳ Stack Residual blocks
↳ Every residual block has two 3x3 conv layers
↳ Periodically, doulbe no. of filters & downsample spatially using stride 2
↳ Additional conv layer at the beginning
↳ No FC
↳ Total depths: 34, 50, 152
↳ For ResNet 50+, use 1x1 conv layers for efficiency

Training ResNet:

↳ Batch Norm after every CONV layer
↳ Xavier initialization
↳ SGD + momentum (0.9)
↳ LR: 0.1, / by 10 when VE plateaus
↳ Batch: 256
↳ Weight decay 1e-5
↳ No dropout

Network in Network:

↳ Convoles a more complex hierarchical filter

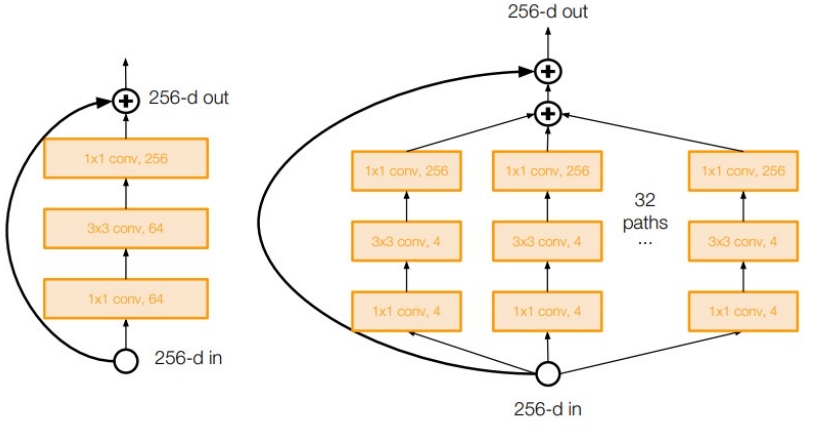


Wider Residual Networks:

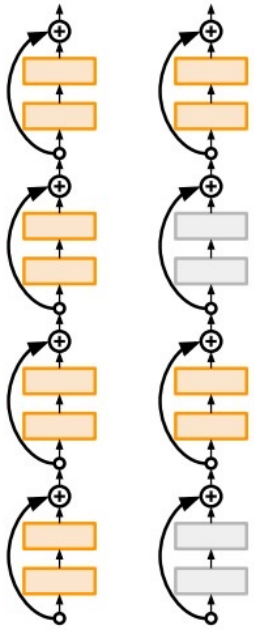
↳ Argued residuals were important & NOT depth of network
↳ They used wider residual blocks (F x k filters instead of F) i.e More filters
↳ 50 layer wide ResNet outperforms 152 layer original ResNet

ResNeXt:

↳ Increases width of residual block through multiple parallel pathways "Cardinality"
↳ Parallel pathways similar in spirit to inception modules



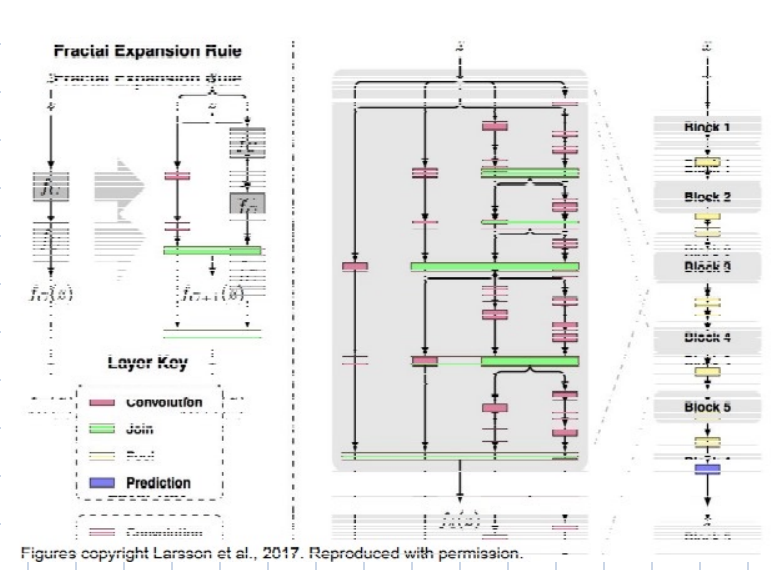
Stochastic Depth:



- Reduce vanishing grads
- Randomly drop a subset of layers
- Bypass with Identify function
- Use full deep networks at test time

FractalNet:

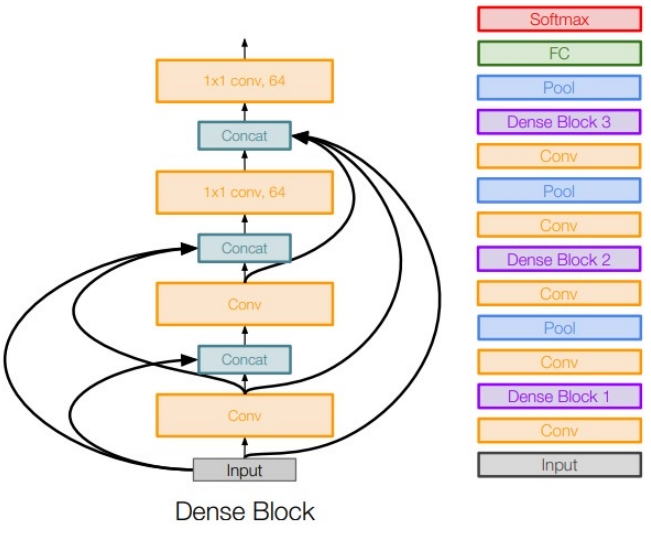
- key is transitioning from shallow to deep networks & NOT residual representations
- Fractal architecture with both shallow & deep pathways to output
- Trained with dropping out sub-paths
- Full network at test time



Figures copyright Larsson et al., 2017. Reproduced with permission.

DenseNets:

- Dense blocks where each layer is connected to every other layer in feedforward fashion.
- Alleviates vanishing grads, strengthens feature prop, encourages feature use.



Dense Block

SqueezeNet: → efficient architectures

↳ Fire modules consisting of a squeeze layer with 1×1 filters feeding an 'expand' layer with 1×1 & 3×3 filters

↳ AlexNet accuracy with 50× fewer params

↳ 500× smaller than AlexNet