CNN Architectures for Image Classification

Shubhra Aich
s.aich.72@gmail.com

January 17, 2018

Architectures

LeNet5
AlexNet
VGG
Inception
ResNet
DenseNet

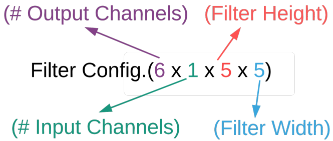
LeNet5(1998)

Input
1x32x32

	Output Size
Conv(6x1x5x5), S=1, P=0	6x28x28
Subsampling(2x2,S=2)	6x14x14
Conv(16x6x5x5), S=1, P=0	16x10x10
Subsampling(2x2,S=2)	16x5x5
Linear(16x5x5=400, 120)	120
Linear(120, 84)	84
Linear(84, 10)	10

Unnormalized Class Scores

Score	1.5	6.9	7.6	-7.1	-3.8	-3.5	-1.9	8.2	-2.8	7.2
Class	0	1	2	3	4	5	6	7	8	9



(# Output Channels)

(Filter Height)

Filter Config.(6 x 1 x 5 x 5)

(# Input Channels)

(Filter Width)

***6 filters/kernels of dimension 1x5x5

$$M_{new} = \frac{M_{old} + 2P - F}{S} + 1$$

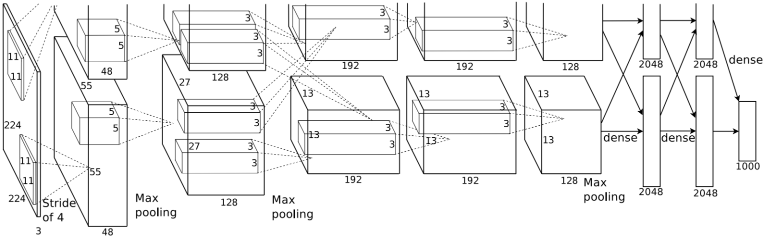
$M_{old} = 32,$

$Stride(S) = 1,$

$Padding(P) = 0$

$$M_{new} = \frac{32 + 2 \times 0 - 5}{1} + 1 = 28$$

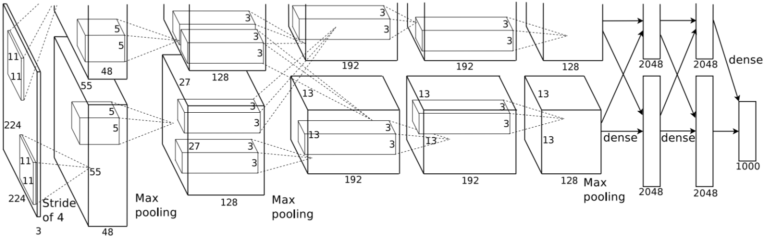
AlexNet(2012)



Input (3x224x224)

	Output Size
Conv(96x3x11x11), S=4, P=2	96x55x55
Max-Pool(3x3,S=2)	96x27x27
Local Response Norm(N=5)	96x27x27
2[Conv(128x48x5x5), S=1, P=2]	2[128x27x27]
Max-Pool(3x3,S=2)	2[128x13x13]
Local Response Norm(N=5)	2[128x13x13]
Conv(384x256x3x3), S=1, P=1]	384x13x13
2[Conv(192x192x3x3), S=1, P=1]	2[192x13x13]
2[Conv(128x192x3x3), S=1, P=1]	2[128x13x13]
Max-Pool(3x3,S=2)	256x6x6=9216
Dropout(0.5)	9216
Linear(9216, 4096) (38M)	4096
Dropout(0.5)	4096
Linear(4096, 4096) (16M)	4096
Linear(4096, 1000)	1000

AlexNet(2012)



Input (3x224x224)

	Output Size
Conv(96x3x11x11), S=4, P=2	96x55x55
Max-Pool(3x3,S=2)	96x27x27
Local Response Norm(N=5)	96x27x27
2[Conv(128x48x5x5), S=1, P=2]	2[128x27x27]
Max-Pool(3x3,S=2)	2[128x13x13]
Local Response Norm(N=5)	2[128x13x13]
Conv(384x256x3x3), S=1, P=1]	384x13x13
2[Conv(192x192x3x3), S=1, P=1]	2[192x13x13]
2[Conv(128x192x3x3), S=1, P=1]	2[128x13x13]
Max-Pool(3x3,S=2)	256x6x6=9216
Dropout(0.5)	9216
Linear(9216, 4096) (38M)	4096
Dropout(0.5)	4096
Linear(4096, 4096) (16M)	4096
Linear(4096, 1000)	1000

Novelties/Significance

- GPU implementation (2xNVIDIA GTX 580 3GB GPUs).
- ReLU as nonlinearity.
- Local Response Normalization (LRN).
- Data augmentation.
- $\sim$ 10% improvement on standard benchmark compared to traditional approaches.

Data Augmentation

- Trained on ImageNet dataset comprising $\sim 1.2M$ training samples.
- 5 (224×224) patch extraction from each training samples.
  - 4 from 4 corners + 1 from the center.
  - horizontal flipping of 5 patches gives 10 patches/image.
  - averaging the scores from 10 (224×224) patches in test phase.
- Obtained illumination invariance of the object identities using the PCA trick :

$$\begin{bmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{bmatrix} += \begin{bmatrix} | & | & | \\ e_1 & e_2 & e_3 \\ | & | & | \end{bmatrix} \begin{bmatrix} \alpha_1\lambda_1 \\ \alpha_2\lambda_2 \\ \alpha_3\lambda_3 \end{bmatrix}$$

$\alpha_i \sim \mathcal{N}(0, 0.1)$

Local Response Normalization

$$b^k_{x,y} = \frac{a^k_{x,y}}{\left(\gamma + \alpha \sum_{i=max(0,N/2)}^{min(D-1,k-N/2)} (a^i_{x,y})^2\right)^{\beta}} \tag{1}$$

$\mathrm{a}^k_{x,y}$ = unnormlized activity generated by the kernel k at position (x, y)
$b^k_{x,y}$ = normalized activity corresponding to $a^k_{x,y}$
D = total number of kernels/feature maps
α, β, γ, N– determined using the validation set.

- Imposes lateral inhibition amongst the neighboring elements in the feature maps.

Results-AlexNet

- Trained for 90 epochs on $1.2M$ images for $5-6$ days using 2xNVIDIA GTX 580 3GB GPUs

Table: ILSVRC 2010 Test Set

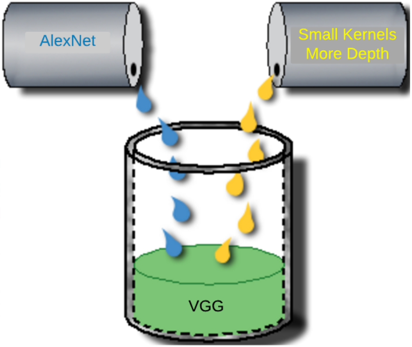
Method	Error (%)	
	Top-1	Top-5
SIFT + FV	45.7	25.7
AlexNet	37.5	17.0

VGG(2014)

- Equivalence of spatial coverage :
  - 1 (5×5) convolution $\equiv$ 2 (3×3) convolutions
  - 1 (7×7) convolution $\equiv$ 3 (3×3) convolutions
- Using smaller convolution kernels is computationally efficient :
  - 1 (5×5) convolution has 25 training parameters, whereas 2 (3×3) convolutions comprise 18 training parameters.
  - 1 (7×7) convolution has 49 training parameters, whereas 3 (3×3) convolutions comprise 27 training parameters.
- (Smaller kernels + More Depth) $\equiv$ More Nonlinearity $\equiv$ Better Modeling.

VGG(2014)

- Equivalence of spatial coverage :
  - 1 (5×5) convolution $\equiv$ 2 (3×3) convolutions
  - 1 (7×7) convolution $\equiv$ 3 (3×3) convolutions
- Using smaller convolution kernels is computationally efficient :
  - 1 (5×5) convolution has 25 training parameters, whereas 2 (3×3) convolutions comprise 18 training parameters.
  - 1 (7×7) convolution has 49 training parameters, whereas 3 (3×3) convolutions comprise 27 training parameters.
- (Smaller kernels + More Depth) $\equiv$ More Nonlinearity $\equiv$ Better Modeling.



AlexNet vs. VGG16

Input (3x224x224)

	Output Size
Conv(96x3x11x11), S=4, P=2	96x55x55
Max-Pool(3x3,S=2)	96x27x27
Local Response Norm(N=5)	96x27x27
2[Conv(128x48x5x5), S=1, P=2]	2[128x27x27]
Max-Pool(3x3,S=2)	2[128x13x13]
Local Response Norm(N=5)	2[128x13x13]
Conv(384x256x3x3), S=1, P=1]	384x13x13
2[Conv(192x192x3x3), S=1, P=1]	2[192x13x13]
2[Conv(128x192x3x3), S=1, P=1]	2[128x13x13]
Max-Pool(3x3,S=2)	256x6x6=9216
Dropout(0.5)	9216
Linear(9216, 4096) (38M)	4096
Dropout(0.5)	4096
Linear(4096, 4096) (16M)	4096
Linear(4096, 1000)	1000

Input (3x224x224)

	Output Size
Conv(64x3x3x3), S=1, P=1	64x224x224
Conv(64x64x3x3), S=1, P=1	64x224x224
Max-Pool(2x2,S=2)	64x112x112
Conv(128x64x3x3), S=1, P=1	128x112x112
Conv(128x128x3x3), S=1, P=1	128x112x112
Max-Pool(2x2,S=2)	128x56x56
Conv(256x128x3x3), S=1, P=1	256x56x56
Conv(256x256x3x3), S=1, P=1	256x56x56
Conv(256x256x3x3), S=1, P=1	256x56x56
Max-Pool(2x2,S=2)	256x28x28
Conv(512x256x3x3), S=1, P=1	512x28x28
Conv(512x512x3x3), S=1, P=1	512x28x28
Conv(512x512x3x3), S=1, P=1	512x28x28
Max-Pool(2x2,S=2)	512x14x14
Conv(512x512x3x3), S=1, P=1	512x14x14
Conv(512x512x3x3), S=1, P=1	512x14x14
Conv(512x512x3x3), S=1, P=1	512x14x14
Max-Pool(2x2,S=2)	512x7x7=25088
Linear(25088, 4096) (102M)	4096
Dropout(0.5)	4096
Linear(4096, 4096) (16M)	4096
Dropout(0.5)	4096
Linear(4096, 1000)	1000

AlexNet vs. VGG16

Input (3x224x224)

	Output Size
Conv(96x3x11x11), S=4, P=2	96x55x55
Max-Pool(3x3,S=2)	96x27x27
Local Response Norm(N=5)	96x27x27
2[Conv(128x48x5x5), S=1, P=2]	2[128x27x27]
Max-Pool(3x3,S=2)	2[128x13x13]
Local Response Norm(N=5)	2[128x13x13]
Conv(384x256x3x3), S=1, P=1]	384x13x13
2[Conv(192x192x3x3), S=1, P=1]	2[192x13x13]
2[Conv(128x192x3x3), S=1, P=1]	2[128x13x13]
Max-Pool(3x3,S=2)	256x6x6=9216
Dropout(0.5)	9216
Linear(9216, 4096) (38M)	4096
Dropout(0.5)	4096
Linear(4096, 4096) (16M)	4096
Linear(4096, 1000)	1000

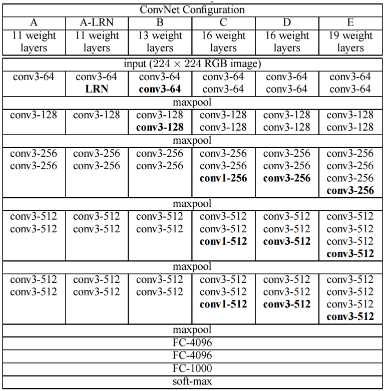
Input (3x224x224)

	Output Size
Conv(64x3x3x3), S=1, P=1	64x224x224
Conv(64x64x3x3), S=1, P=1	64x224x224
Max-Pool(2x2,S=2)	64x112x112
Conv(128x64x3x3), S=1, P=1	128x112x112
Conv(128x128x3x3), S=1, P=1	128x112x112
Max-Pool(2x2,S=2)	128x56x56
Conv(256x128x3x3), S=1, P=1	256x56x56
Conv(256x256x3x3), S=1, P=1	256x56x56
Conv(256x256x3x3), S=1, P=1	256x56x56
Max-Pool(2x2,S=2)	256x28x28
Conv(512x256x3x3), S=1, P=1	512x28x28
Conv(512x512x3x3), S=1, P=1	512x28x28
Conv(512x512x3x3), S=1, P=1	512x28x28
Max-Pool(2x2,S=2)	512x14x14
Conv(512x512x3x3), S=1, P=1	512x14x14
Conv(512x512x3x3), S=1, P=1	512x14x14
Conv(512x512x3x3), S=1, P=1	512x14x14
Max-Pool(2x2,S=2)	512x7x7=25088
Linear(25088, 4096) (102M)	4096
Dropout(0.5)	4096
Linear(4096, 4096) (16M)	4096
Dropout(0.5)	4096
Linear(4096, 1000)	1000

- AlexNet: variable size kernels based on heuristics, hard to modify.
- VGG: deeper network with fixed size kernels, so comparatively easier to play with, no use of LRN.

Training VGG

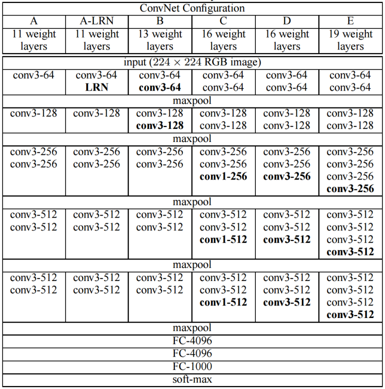
- VGG is deeper – hard to train with Gaussian initialization.
- Pre-initialization with smaller nets (version A).
- Training with multi-scale inputs.
- Dense evaluations (150 per sample) at the test time.



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Training VGG

- VGG is deeper – hard to train with Gaussian initialization.
- Pre-initialization with smaller nets (version A).
- Training with multi-scale inputs.
- Dense evaluations (150 per sample) at the test time.



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Possible to train from scratch with Xavier-Glorot(2010) initialization !!!

Results-VGG

- Training on 1.2M ImageNet samples with 4xNVIDIA TITAN Black 6GB GPUs takes $2-3$ weeks depending on the architecture.

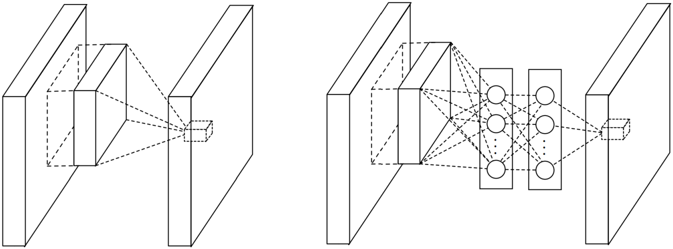
Method	Error (%)	
	Top-1	Top-5
SIFT + FV	45.7	25.7
AlexNet	37.5	17.0
VGG16	24.4	7.2
VGG19	24.4	7.1

We Need More Nonlinearity

- Using small kernels (3×3) $\implies$ VGG.
- Theoretically, high-dimensional data space demands more non-linearity in the models.
  - Training deeper networks (did not work until 2015).
  - Replacement of linear convolution operation with a non-linear operation – **Nested Network** or **Network in Network**.

We Need More Nonlinearity

- Using small kernels $(3 \times 3) \implies$ VGG.
- Theoretically, high-dimensional data space demands more non-linearity in the models.
  - Training deeper networks (did not work until 2015).
  - Replacement of linear convolution operation with a non-linear operation – **Nested Network** or **Network in Network**.



(a) Linear convolution layer

(b) Mlpconv layer

Network In Network (NIN)

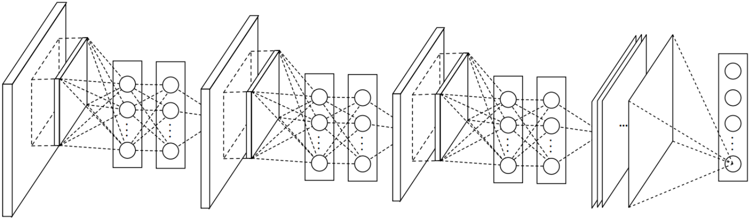


Figure 2: The overall structure of Network In Network. In this paper the NINs include the stacking of three mlpconv layers and one global average pooling layer.

- Replacement of linear convolution with MLP-NN.
- Replacement of HUGE (!!!) Fully-Connected (FC) layers with a single Global Average Pooling (GAP) layer
  - More than 90 – 95% reduction of training parameters $\Longrightarrow$ much less prone to overfitting.
  - More emphasis on convolutional features (head of the architecture).

Paradigm Shift

Heavy-Tail (AlexNet, VGG) → Heavy-Head (NIN, Inception, ...)

Paradigm Shift

Heavy-Tail (AlexNet, VGG) → Heavy-Head (NIN, Inception, ...)

Breaking down Convolution ... (1)

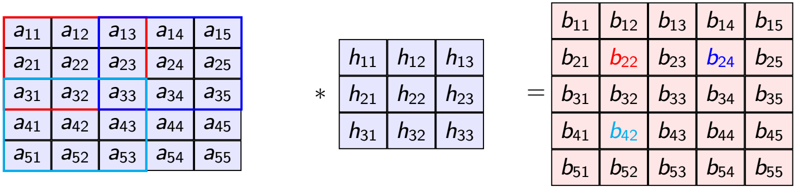
- Straightforward 5 × 5 convolution
  M ≡ Multiplication and A ≡ Addition

$$\begin{array}{|c|c|c|c|c|}\hline a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ \hline a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ \hline a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ \hline a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ \hline \end{array} \quad * \quad \begin{array}{|c|c|c|c|c|}\hline h_{11} & h_{12} & h_{13} & h_{14} & h_{15} \\ \hline h_{21} & h_{22} & h_{23} & h_{24} & h_{25} \\ \hline h_{31} & h_{32} & h_{33} & h_{34} & h_{35} \\ \hline h_{41} & h_{42} & h_{43} & h_{44} & h_{45} \\ \hline h_{51} & h_{52} & h_{53} & h_{54} & h_{55} \\ \hline \end{array}$$

$output_{33} = a_{11}h_{11} + a_{12}h_{12} + \cdots + a_{31}h_{31} + a_{32}h_{32} + \cdots + a_{54}h_{54} + a_{55}h_{55}$
$\equiv 25\text{M} + 24\text{A}$
The filter (h_{ij}) will slide over the image (a_{ij}) for each position once totalling 25 times.
So, total number of gross multiplication and addition $= 25(25\text{M} + 24\text{A}) = 625\text{M} + 600\text{A}$

Breaking down Convolution … (2)

- Breaking down 5 × 5 convolution into 2 consecutive 3 × 3 convolutions
  - Step - 01: Convolving 5 × 5 matrix with 3 × 3*filter*



$$\begin{array}{|c|c|c|c|c|}\hline a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ \hline a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ \hline a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ \hline a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ \hline \end{array} * \begin{array}{|c|c|c|}\hline h_{11} & h_{12} & h_{13} \\ \hline h_{21} & h_{22} & h_{23} \\ \hline h_{31} & h_{32} & h_{33} \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|}\hline b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ \hline b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ \hline b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ \hline b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \\ \hline b_{51} & b_{52} & b_{53} & b_{54} & b_{55} \\ \hline \end{array}$$

$b_{22} = a_{11}h_{11} + a_{12}h_{12} + \cdots + a_{32}h_{32} + a_{33}h_{33}$
$\equiv 9M + 8A$
This set of operation is performed 25 times, once for each a_{ij}
So, total multiplication + addition in the first step
$= 25(9M + 8A) = 225M + 200A$

Breaking down Convolution ... (3)

- Breaking down 5 × 5 convolution into 2 consecutive 3 × 3 convolutions
    - Step - 02: Convolving 3 × 3 intermediate matrix with another 3 × 3 filter

$$\begin{array}{|c|c|c|c|c|}\hline b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ \hline b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ \hline b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ \hline b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \\ \hline b_{51} & b_{52} & b_{53} & b_{54} & b_{55} \\ \hline \end{array} * \begin{array}{|c|c|c|}\hline k_{11} & k_{12} & k_{13} \\ \hline k_{21} & k_{22} & k_{23} \\ \hline k_{31} & k_{32} & k_{33} \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|}\hline o_{11} & o_{12} & o_{13} & o_{14} & o_{15} \\ \hline o_{21} & o_{22} & o_{23} & o_{24} & o_{25} \\ \hline o_{31} & o_{32} & o_{33} & o_{34} & o_{35} \\ \hline o_{41} & o_{42} & o_{43} & o_{44} & o_{45} \\ \hline o_{51} & o_{52} & o_{53} & o_{54} & o_{55} \\ \hline \end{array}$$

$o_{22} = b_{11}k_{11} + b_{12}k_{12} + \cdots + b_{32}k_{32} + b_{33}k_{33}$

$\equiv$ 9M + 8A

Like before, this set of operation is performed 25 times, once for each b_{ij}

So, total multiplication + addition in the first step

= 25(9M + 8A) = 225M + 200A

Breaking down Convolution ... (4)

$$\begin{array}{|c|c|c|c|c|}\hline a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ \hline a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ \hline a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ \hline a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ \hline \end{array} \quad * \quad \begin{array}{|c|c|c|c|c|}\hline h_{11} & h_{12} & h_{13} & h_{14} & h_{15} \\ \hline h_{21} & h_{22} & h_{23} & h_{24} & h_{25} \\ \hline h_{31} & h_{32} & h_{33} & h_{34} & h_{35} \\ \hline h_{41} & h_{42} & h_{43} & h_{44} & h_{45} \\ \hline h_{51} & h_{52} & h_{53} & h_{54} & h_{55} \\ \hline \end{array}$$

- Computational cost of straightforward 5×5 convolution $= 625\text{M} + 600\text{A}$.

$$\begin{array}{|c|c|c|c|c|}\hline a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ \hline a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ \hline a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ \hline a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ \hline \end{array} \quad * \quad \begin{array}{|c|c|c|}\hline h_{11} & h_{12} & h_{13} \\ \hline h_{21} & h_{22} & h_{23} \\ \hline h_{31} & h_{32} & h_{33} \\ \hline \end{array} \quad * \quad \begin{array}{|c|c|c|}\hline k_{11} & k_{12} & k_{13} \\ \hline k_{21} & k_{22} & k_{23} \\ \hline k_{31} & k_{32} & k_{33} \\ \hline \end{array}$$

- Computational cost of 5×5 convolution using consecutive 3×3 convolutions $= 450\text{M} + 400\text{A}$.

Breakding down Convolution ... (5)

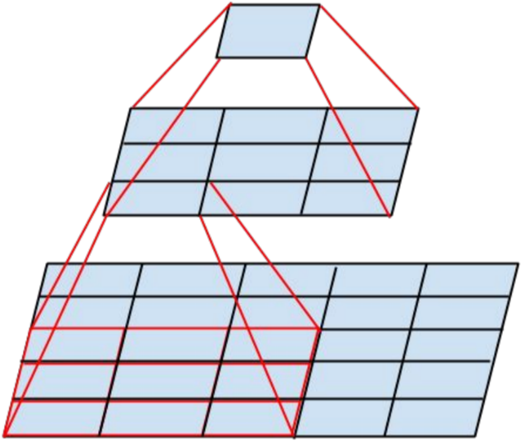


Figure 1. Mini-network replacing the 5 × 5 convolutions.

Breakding down Convolution ... (6)

Breakding down 3 × 3 convolution into 3 × 1 and 1 × 3 convolutions (assymetric breakdown).

- Let us calculate the average of a 3 × 3 matrix.

$$\begin{array}{|c|c|c|}\hline 9 & 8 & 7 \\ \hline 6 & 5 & 4 \\ \hline 3 & 2 & 1 \\ \hline \end{array} \quad * \quad \begin{array}{|c|c|c|}\hline \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \hline \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \hline \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \hline \end{array} \quad = \frac{9}{9} + \frac{8}{9} + \cdots + \frac{2}{9} + \frac{1}{9} \equiv 9(9\text{M} + 8\text{A})$$

$$\begin{array}{|c|c|c|}\hline \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \hline \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \hline \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \hline \end{array} \quad = \quad \begin{array}{|c|}\hline \frac{1}{3} \\ \hline \frac{1}{3} \\ \hline \frac{1}{3} \\ \hline \end{array} \quad \begin{array}{|c|c|c|}\hline \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \hline \end{array} \quad \begin{aligned} &\equiv 9(3\text{M} + 2\text{A}) + 9(3\text{M} + 2\text{A}) \\ &= 9(6\text{M} + 4\text{A}) \end{aligned}$$

- The average kernel is a separable filter (rank-1 matrix).

Inception Modules ... (1)

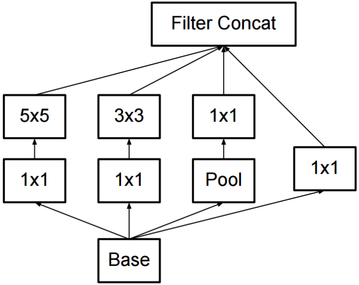
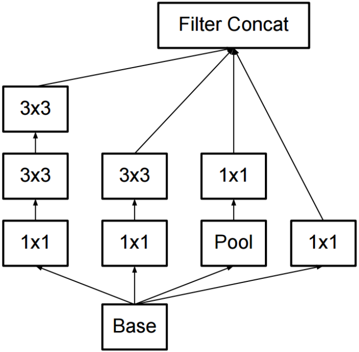
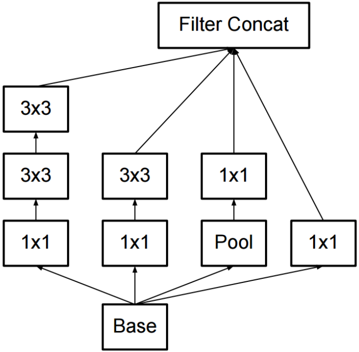


Figure 4. Original Inception module as described in [20].



Inception Modules ... (2)



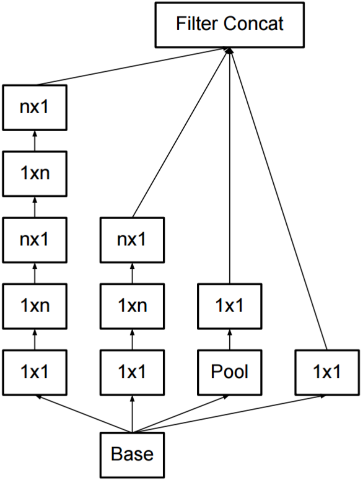
Filter Concat

3x3

3x3 3x3 1x1

1x1 1x1 Pool 1x1

Base



Filter Concat

nx1

1xn

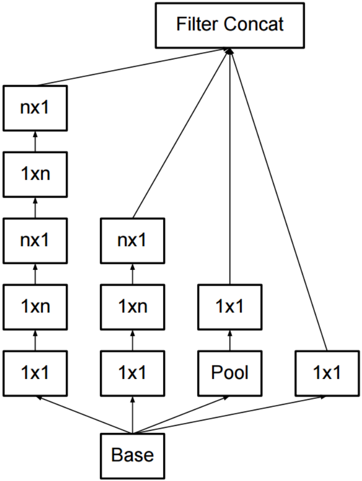
nx1 nx1

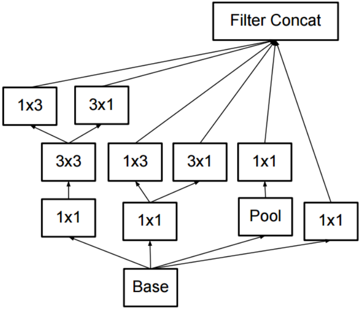
1xn 1xn 1x1

1x1 1x1 Pool 1x1

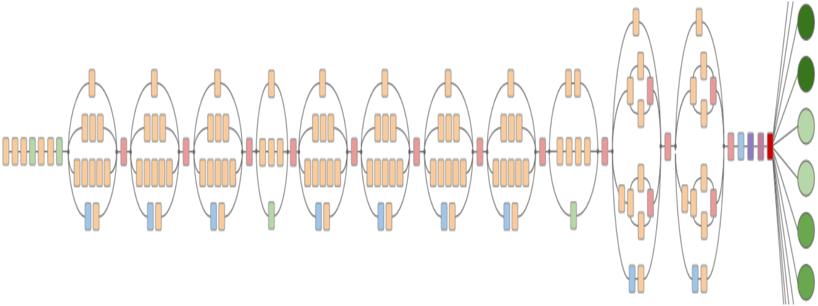
Base

Inception Modules - Final

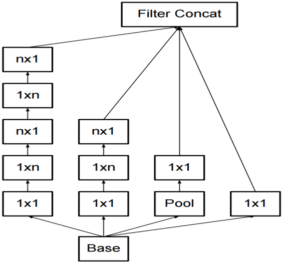


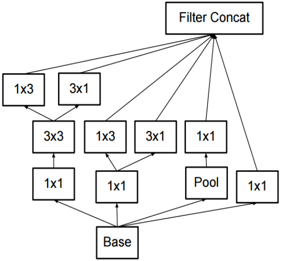


Inception-v3(2014/2016)



- 🟧 Convolution
- 🟦 AvgPool
- 🟩 MaxPool
- 🟥 Concat
- 🟪 Dropout
- 🟪 Fully connected
- 🟥 Softmax





Results-Inception

- Trained for 100 epochs on NVIDIA Kepler GPUs.

Method	Error (%)	
	Top-1	Top-5
SIFT + FV	45.7	25.7
AlexNet	37.5	17.0
VGG16	24.4	7.2
VGG19	24.4	7.1
Inception-v3	18.8	4.2

Analysis of Network Depth

- Simply stacking up lots of convolutional layers make performance worse.

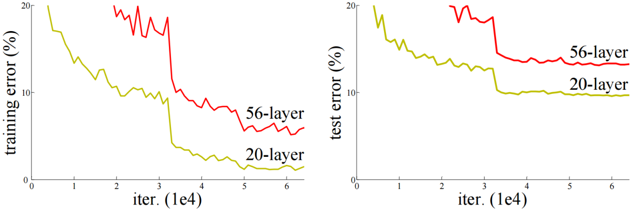


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

- Current gradient-descent solvers are bad at optimizing the training parameters for identity mapping.

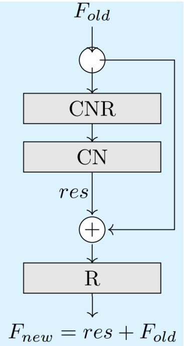
Learning by Comparison

- Rather than memorizing from scratch, memorizing the changes with respect to something known makes learning faster.
- Example: You are fluent in English and are now interested to learn French.

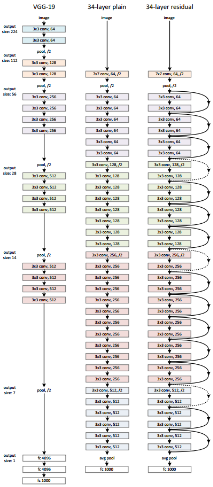
  University (English) – Université (French)

Learning by Comparison

- Rather than memorizing from scratch, memorizing the changes with respect to something known makes learning faster.
- Example: You are fluent in English and are now interested to learn French.
  University (English) – Université (French)
- This idea of "Learning by Comparison" can be implemented in CNN using shortcut/bypass connections.



ResNet(2015)



- Possible to train of much deeper models than before, e.g. ResNet50, ResNet101, ResNet152.
- Deeper models work better.
- Use of Global Average Pooling (GAP) instead of Fully Connected (FC) layers.
- Use of Batch-Normalization (running estimation of mini-batch mean and std and normalization afterward) instead of Dropout.

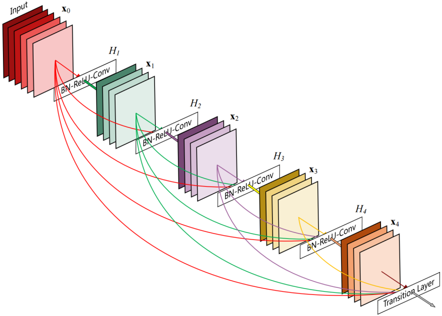
Method	Error (%)	
	Top-1	Top-5
SIFT + FV	45.7	25.7
AlexNet	37.5	17.0
VGG16	24.4	7.2
VGG19	24.4	7.1
Inception-v3	18.8	4.2
ResNet152	-	3.57

Existing Problems in Deeper Networks

- Gradienet might still vanish by the time it reaches the end (Vanishing Gradient problem).
  - Possible solution: More Dense connection than shortcut to maximize information flow.
- Many layers in ResNet contribute very little and can be stochastically dropped.
  - Possible solution: Feature reuse from the previous layers
- Computations in ResNet layers are explicit and requires extra memory.
  - Possible solution: Memory reuse.

Existing Problems in Deeper Networks

- Gradienet might still vanish by the time it reaches the end (Vanishing Gradient problem).
  - Possible solution: More Dense connection than shortcut to maximize information flow.
- Many layers in ResNet contribute very little and can be stochastically dropped.
  - Possible solution: Feature reuse from the previous layers
- Computations in ResNet layers are explicit and requires extra memory.
  - Possible solution: Memory reuse.



DenseNet(2017)

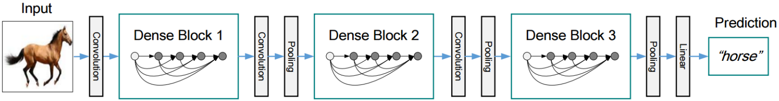
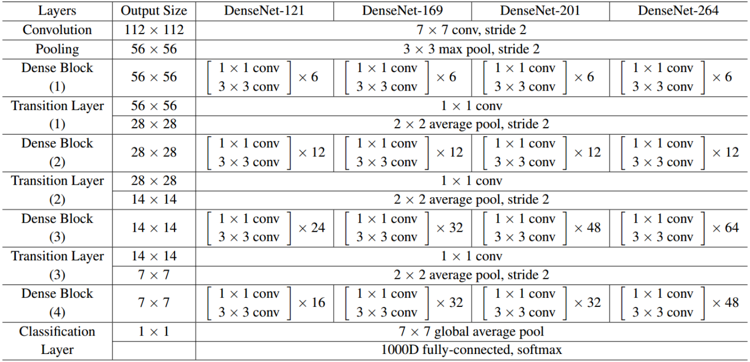


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.



Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	$\begin{bmatrix} 1\times 1 \text{ conv} \\ 3\times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times 1 \text{ conv} \\ 3\times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times 1 \text{ conv} \\ 3\times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times 1 \text{ conv} \\ 3\times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	$\begin{bmatrix} 1\times 1 \text{ conv} \\ 3\times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times 1 \text{ conv} \\ 3\times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times 1 \text{ conv} \\ 3\times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times 1 \text{ conv} \\ 3\times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	$\begin{bmatrix} 1\times 1 \text{ conv} \\ 3\times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1\times 1 \text{ conv} \\ 3\times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times 1 \text{ conv} \\ 3\times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1\times 1 \text{ conv} \\ 3\times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	$\begin{bmatrix} 1\times 1 \text{ conv} \\ 3\times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1\times 1 \text{ conv} \\ 3\times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times 1 \text{ conv} \\ 3\times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times 1 \text{ conv} \\ 3\times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Table 1: DenseNet architectures for ImageNet. The growth rate for all the networks is $k = 32$. Note that each "conv" layer shown in the table corresponds the sequence BN-ReLU-Conv.

Final Results - Single Crop

Table: Accuracy on Single 224 × 224 Crop

Method	Error (%)	
	Top-1	Top-5
AlexNet	43.45	20.91
VGG16 + BN	26.63	8.50
VGG19 + BN	25.76	8.15
Inception-v3	22.55	6.44
ResNet152	21.69	5.94
DenseNet161	22.35	6.20

References I

- Xavier Glorot and Yoshua Bengio.
  Understanding the difficulty of training deep feedforward neural networks.
  In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

- K. He, X. Zhang, S. Ren, and J. Sun.
  Deep residual learning for image recognition.
  In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016.

- G. Huang, Z. Liu, L. v. d. Maaten, and K. Q. Weinberger.
  Densely connected convolutional networks.
  In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, July 2017.

- Sergey Ioffe and Christian Szegedy.
  Batch normalization: Accelerating deep network training by reducing internal covariate shift.
  CoRR, abs/1502.03167, 2015.

References II

- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton.
  Imagenet classification with deep convolutional neural networks.
  In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

- Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner.
  Gradient-based learning applied to document recognition.
  Proceedings of the IEEE, 86(11):2278–2324, Nov 1998.

- Min Lin, Qiang Chen, and Shuicheng Yan.
  Network in network.
  CoRR, abs/1312.4400, 2013.

- K. Simonyan and A. Zisserman.
  Very deep convolutional networks for large-scale image recognition.
  CoRR, abs/1409.1556, 2014.

References III

C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich.
Going deeper with convolutions.
In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, June 2015.

C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna.
Rethinking the inception architecture for computer vision.
In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, June 2016.