# Deep Learning and Neural Networks

Demetrio Labate

March 4, 2024

## Part 2
## Convolutional Neural Networks

## 2.3 Advanced CNN design

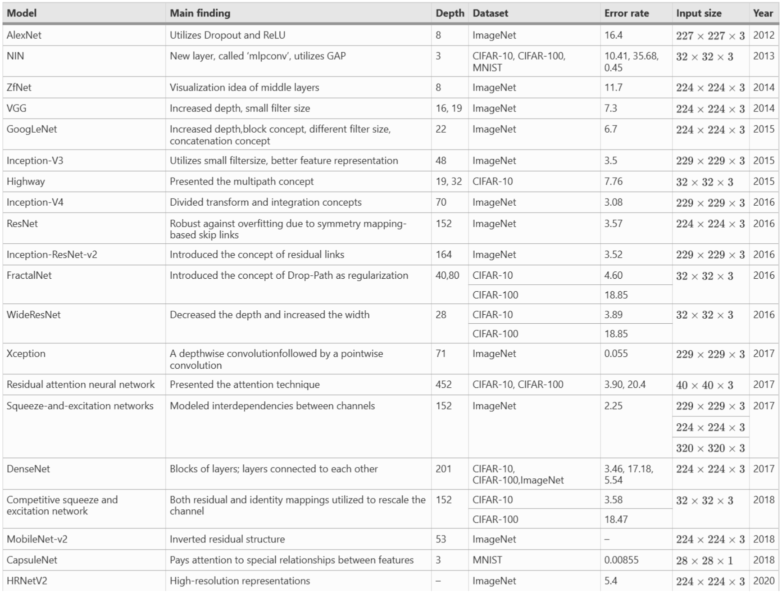
## Convolutional Neural Networks

Since the introduction of CNNs in 1989, various modifications have been achieved in CNN architecture including structural reformulation, regularization and parameter optimization.

Some key upgrades in CNN performance occurred due to

1. the use of network depth;
2. processing-unit reorganization;
3. development of novel blocks.

# Convolutional Neural Networks



| Model | Main finding | Depth | Dataset | Error rate | Input size | Year |
|---|---|---|---|---|---|---|
| AlexNet | Utilizes Dropout and ReLU | 8 | ImageNet | 16.4 | $227 \times 227 \times 3$ | 2012 |
| NIN | New layer, called 'mlpconv', utilizes GAP | 3 | CIFAR-10, CIFAR-100, MNIST | 10.41, 35.68, 0.45 | $32 \times 32 \times 3$ | 2013 |
| ZfNet | Visualization idea of middle layers | 8 | ImageNet | 11.7 | $224 \times 224 \times 3$ | 2014 |
| VGG | Increased depth, small filter size | 16, 19 | ImageNet | 7.3 | $224 \times 224 \times 3$ | 2014 |
| GoogLeNet | Increased depth,block concept, different filter size, concatenation concept | 22 | ImageNet | 6.7 | $224 \times 224 \times 3$ | 2015 |
| Inception-V3 | Utilizes small filtersize, better feature representation | 48 | ImageNet | 3.5 | $229 \times 229 \times 3$ | 2015 |
| Highway | Presented the multipath concept | 19, 32 | CIFAR-10 | 7.76 | $32 \times 32 \times 3$ | 2015 |
| Inception-V4 | Divided transform and integration concepts | 70 | ImageNet | 3.08 | $229 \times 229 \times 3$ | 2016 |
| ResNet | Robust against overfitting due to symmetry mapping-based skip links | 152 | ImageNet | 3.57 | $224 \times 224 \times 3$ | 2016 |
| Inception-ResNet-v2 | Introduced the concept of residual links | 164 | ImageNet | 3.52 | $229 \times 229 \times 3$ | 2016 |
| FractalNet | Introduced the concept of Drop-Path as regularization | 40,80 | CIFAR-10 | 4.60 | $32 \times 32 \times 3$ | 2016 |
| | | | CIFAR-100 | 18.85 | | |
| WideResNet | Decreased the depth and increased the width | 28 | CIFAR-10 | 3.89 | $32 \times 32 \times 3$ | 2016 |
| | | | CIFAR-100 | 18.85 | | |
| Xception | A depthwise convolutionfollowed by a pointwise convolution | 71 | ImageNet | 0.055 | $229 \times 229 \times 3$ | 2017 |
| Residual attention neural network | Presented the attention technique | 452 | CIFAR-10, CIFAR-100 | 3.90, 20.4 | $40 \times 40 \times 3$ | 2017 |
| Squeeze-and-excitation networks | Modeled interdependencies between channels | 152 | ImageNet | 2.25 | $229 \times 229 \times 3$ | 2017 |
| | | | | | $224 \times 224 \times 3$ | |
| | | | | | $320 \times 320 \times 3$ | |
| DenseNet | Blocks of layers; layers connected to each other | 201 | CIFAR-10, CIFAR-100,ImageNet | 3.46, 17.18, 5.54 | $224 \times 224 \times 3$ | 2017 |
| Competitive squeeze and excitation network | Both residual and identity mappings utilized to rescale the channel | 152 | CIFAR-10 | 3.58 | $32 \times 32 \times 3$ | 2018 |
| | | | CIFAR-100 | 18.47 | | |
| MobileNet-v2 | Inverted residual structure | 53 | ImageNet | – | $224 \times 224 \times 3$ | 2018 |
| CapsuleNet | Pays attention to special relationships between features | 3 | MNIST | 0.00855 | $28 \times 28 \times 1$ | 2018 |
| HRNetV2 | High-resolution representations | – | ImageNet | 5.4 | $224 \times 224 \times 3$ | 2020 |

## Common Test Sets

Standard datasets used to benchmark deep learning algorithms include:

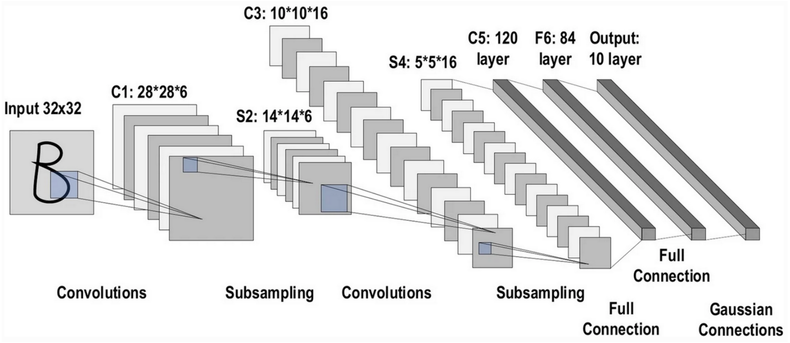
- MNIST. Database of handwritten digits including a training set of 60,000 samples, and a test set of 10,000 samples. The digits are size-normalized and centered in a fixed-size 28x28-pixel grey-scale image. [Link]
- CIFAR-10. This dataset consists of 60,000 32x32-pixel colour (RGB) images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images.
- CIFAR-100. This dataset is like CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in CIFAR-100 are grouped into 20 superclasses. [Link]

## Common Test Sets

- ImageNet. There are different versions of ImageNet. The most highly-used subset is the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012-2017 image classification and localization dataset; this version is also referred to in the literature as ImageNet-1K or ILSVRC2017. It contains RGB images spanning 1000 object classes that inclue 1,281,167 training images, 50,000 validation images and 100,000 test images. Images vary in dimensions and resolution (often applications resize/crop images to 256x256 pixels). [Link]
  The full original dataset is referred to as ImageNet-21K. It contains 14,197,122 images divided into 21,841 classes. Some papers round this up and name it ImageNet-22k.
- Other sets: Microsoft Common Objects in Context (MS COCO) data set (images), Places (images), PASCAL VOC (images), Caltech Pedestrian (videos), Intel Image Classification (images), MIT Indoor Scenes (images).

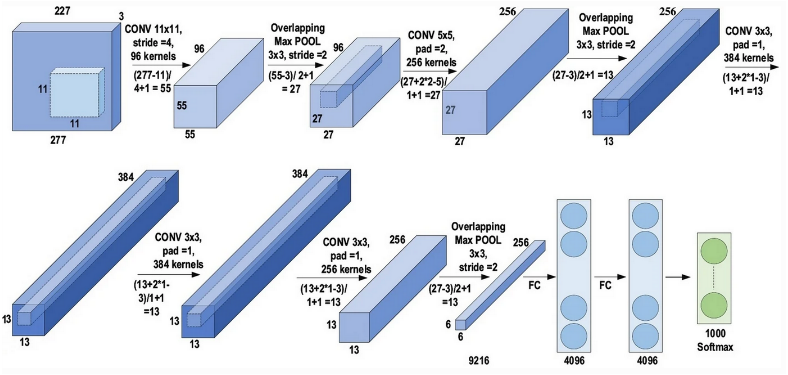
## Convolutional Neural Networks - LeNet

The history of deep CNNs began with **LeNet** [LeCun et al, 1995] that was aimed at solving the handwritten digit recognition tasks. However, it did not work well with more general image classes.



# Convolutional Neural Networks - AlexNet

**AlexNet** [Krizhevsky, Sutskever, Hinton, 2017] significantly improved the CNN learning ability by increasing depth and implementing several parameter optimization and regularization strategies.



227
3
11
11
277
CONV 11x11, stride =4, 96 kernels
(277-11)/4+1 = 55
96
55
55
Overlapping Max POOL 3x3, stride =2
(55-3)/ 2+1 = 27
96
27
27
CONV 5x5, pad =2, 256 kernels
(27+2*2-5)/1+1 =27
256
27
27
Overlapping Max POOL 3x3, stride =2
(27-3)/2+1 =13
256
13
13
CONV 3x3, pad =1, 384 kernels
(13+2*1-3)/1+1 =13
384
13
13
CONV 3x3, pad =1, 384 kernels
(13+2*1-3)/1+1 =13
384
13
13
CONV 3x3, pad =1, 256 kernels
(13+2*1-3)/1+1 =13
256
13
13
Overlapping Max POOL 3x3, stride =2
(27-3)/2+1 =13
256
6
6
9216
FC
4096
FC
4096
1000 Softmax

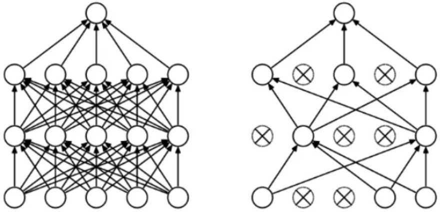
## Convolutional Neural Networks - AlexNet

To improve on the performance of previous networks, AlexNet included several new design choices.

- The number of feature extraction stages was increased from 5 in LeNet to 7 in AlexNet, with a larger number of convolution kernels (95 in first layer of AlexNet, as compared to 6 in LeNet). This significantly enhances expressive power but increases the risk of overfitting.
- To prevent or reduce the chance of overfitting, Alexnet adopts a strategy relying on ReLU activation functions and dropout regularization proposed by Dahl, Sainath and Hinton (2013).
- To improve performance, large-size filters are used in the earlier layers, namely $11 \times 11$ and $5 \times 5$.
- ReLU is utilized as a non-saturating activation function that enhances the rate of convergence by reducing the vanishing gradient problem.

## Convolutional Neural Networks - Dropout

**Dropout** refers to dropping out some nodes (in input or hidden layer) in a neural network so that all the forward and backwards connections with a dropped node are temporarily removed, thus creating a new network architecture out of the parent network. The nodes are dropped by a **dropout probability** of $p$.



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

## Convolutional Neural Networks - Dropout

Example:

Suppose we have an input $x : \{1, 2, 3, 4, 5\}$ to a fully connected layer and a dropout layer with probability p = 0.2 (or keep probability = 0.8).

During the forward propagation (training) from the input x, 20% of the nodes would be dropped, i.e. the $x$ could become $\{1, 0, 3, 4, 5\}$ or $\{1, 2, 0, 4, 5\}$ and so on. Similarly, it applied to the hidden layers.

For instance, if the hidden layers have 1000 neurons (nodes) and a dropout is applied with drop probability $p = 0.5$, then 500 neurons would be randomly dropped in every iteration (batch).

Generally, for the input layers, the keep probability, i.e. 1- drop probability, is closer to 1, 0.8 being the best as suggested by the authors.
For the hidden layers, the greater the drop probability more sparse the model, where 0.5 is heuristically determined to be optimal.

## Convolutional Neural Networks - Dropout

How is dropout producing regularization?

Overfitting occurs when the model learns the statistical noise in the training data rather than the true pattern.
This results in poor performance when the model is evaluated on new data, e.g. a test dataset.

Dropout approximates the operation of averaging the training process over a large number of neural networks with different architectures in parallel.

During training, some number of layer outputs are randomly ignored or “dropped out.” This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer. In fac, each update to a layer during training is performed with a variation of the configured layer.

## Convolutional Neural Networks - Dropout

In overfitting, a node may change in a way that fixes up the mistakes of the other nodes.

This leads to complex co-adaptations, which in turn leads to the overfitting problem because this complex co-adaptation fails to generalise on the unseen dataset.

Dropout has the effect of preventing nodes to fix up the mistake of other nodes, thus preventing co-adaptation, as in every iteration the presence of a node is highly unreliable.
So by randomly dropping a few units (nodes), it forces the layers to take more or less responsibility for the input by taking a probabilistic approach.

This ensures that the model is getting generalised and hence reducing the overfitting problem.

## Convolutional Neural Networks - Dropout

How is dropout implemented?

Dropout is implemented per-layer in a neural network.

It can be used with most types of layers, including dense fully connected layers, convolutional layers, and recurrent layers.

Dropout may be implemented on any or all hidden layers in the network and in the input layer. It is not used on the output layer.

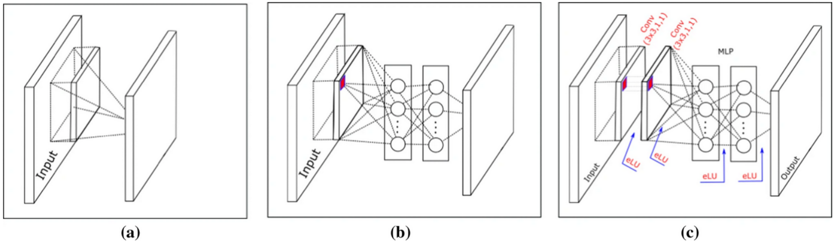
A common dropout probability is $p = 0.5$ for retaining the output of each node in a hidden layer and a value close to 1.0, such as 0.8, for retaining inputs from the input layer.

Note: Dropout is not used after training when making a prediction with the fit network. The weights of the network will be larger than normal because of dropout. Therefore, before finalizing the network, the weights are first scaled by the chosen dropout rate.

# Convolutional Neural Networks - NiN

**Network-in-network** [Lin, Chen, Yan, 2014] introduced two innovative concepts.

- The first idea was to employ multiple layers of The **MLPconv layers** consisting of a linear convolution layer and a two-layer MLP with a ReLU used as an activation function.[1]

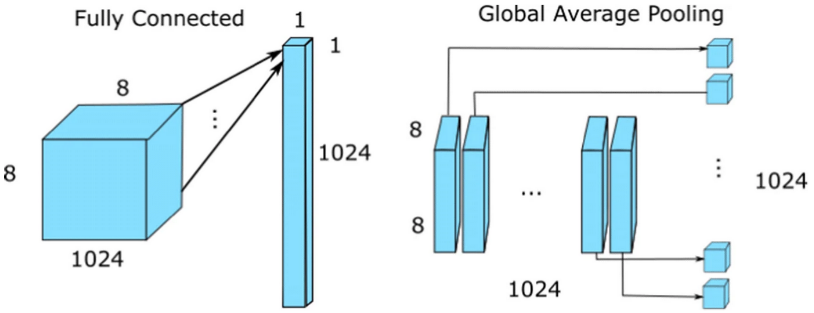


**a** Linear convolution layer, **b** MLPconv layer, **c** Deep MLPconv layer

[1]Some authors call it *perception convolution* and describe it as consisting of convolutions with a $1 \times 1$ filter followed by ReLU nonlinearities.

## Convolutional Neural Networks - NiN

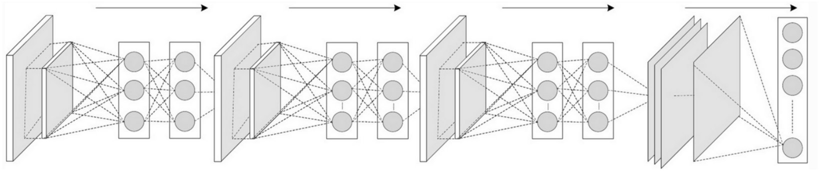
- The second idea is the application of **Global Average Pooling** enabling a significant reduction in the number of model parameters. This solution generates a feature map for each classification category. The average of each feature map and the resulting vector are then directly fed into the softmax layer. One advantage is that there is no parameter to optimize which avoids over-adjustments at this layer.



Example of fully connected layer versus global average pooling layer

## Convolutional Neural Networks - NiN

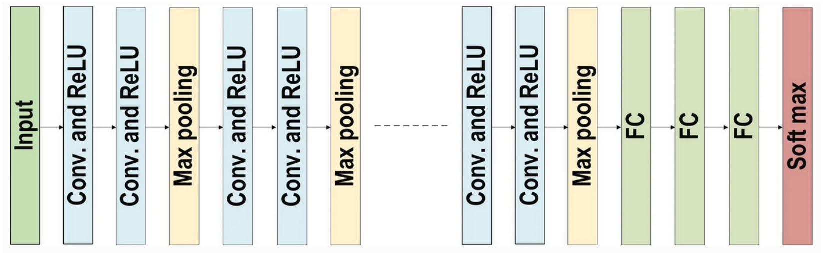
Network-in-network is implemented as a cascade of perception convolution layers followed in the last stage by Global Average Pooling.



The implementation also includes the use of dropout to improve performance.

## Convolutional Neural Networks - VGG

The **Visual Geometry Group (VGG)** networks [Simonyan, Zisserman, 2014] introduced an innovative architecture cascading many more layers (16 to 19 layers) than prior constructions.



## Convolutional Neural Networks - VGG

VGG uses stacks of convolutional layers with $3 \times 3$ filters to replace single convolutional layers with $11 \times 11$ or $7 \times 7$ filters such as those used in AlexNet and ZefNet.

What is the advantage of using **smaller filters** while adding **more layers**?

- First, by incorporating three non-linear rectification layers instead of a single one, it makes the decision function more discriminative.

- Second, it decreases the number of parameters.

Assuming that both the input and the output of a three-layer $3 \times 3$ convolution stack has $C$ channels, the stack is parametrised by $3(3^2C^2) = 27C^2$ weights.
At the same time, a single $7 \times 7$ convolutional layer would require $7^2C^2 = 49C^2$ parameters, i.e. 81% more.

## Convolutional Neural Networks - VGG

It was shown experimentally that cascading layers with small-size filters could produce the same representation power as larger-size filters.

This can be interpreted as a regularisation on the larger convolutional filters, forcing them to have a decomposition through $3 \times 3$ filters.

VGG also incorporates $1 \times 1$ **convolutional layers** of the type introduced in the Network-in-network architecture.
Even though in this case the $1 \times 1$ convolution is essentially a linear projection onto the space of the same dimensionality (the number of input and output channels is the same), an additional non-linearity is introduced by the rectification function.

**Padding** is also implemented to maintain the spatial resolution of the input signal

## Convolutional Neural Networks - VGG

VGG obtained significant results for localization problems and image classification and became very popular due to its enlarged depth, homogenous topology, and simplicity.

Historically, the VGG network established that **representation depth** is beneficial for the classification accuracy. It also produced a research trend for working with small-size filters in CNN.

However, VGG's computational cost for training was very intensive due to its utilization of around 140 million parameters. This represented its main shortcoming.

## GoogLeNet

**GoogLeNet**, also called **Inception-V1**, emerged in 2014 [Szegedy et al, 2015] by winning the 2014-ILSVRC competition.

GoogLeNet is a 22-layer deep CNN whose architecture is very different from previous state-of-the-art architectures such as AlexNet and ZfNet.

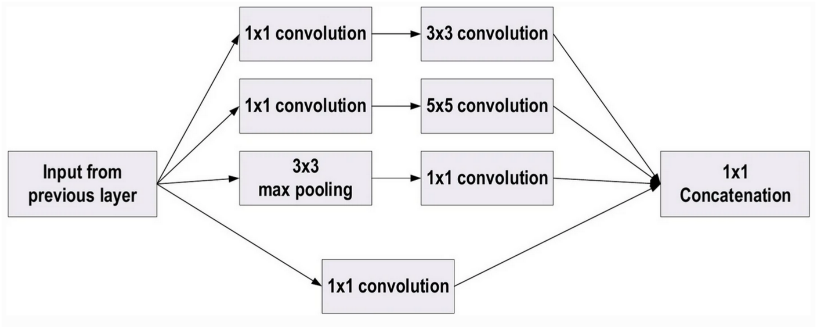
It includes several ideas to improve performance, most notably:

- a novel **inception block (or module)** that combines multiple-scale convolutional transformations by employing merge, transform, and split functions for feature extraction;
- $1 \times 1$ convolutions;
- global average pooling.

These methods are aimed at creating a deep architecture while controlling the total number of weights.

## GoogLeNet

In prior CNN architectures, there was a fixed convolution size for each layer. In the Inception module, filters of different sizes, namely $5 \times 5, 3 \times 3, 1 \times 1$ and $3 \times 3$ max pooling are performed **in parallel** and then stacked together to generated final output.



The idea behind convolution filters of different sizes is to better capture channel information together with spatial information at diverse ranges of spatial resolution.

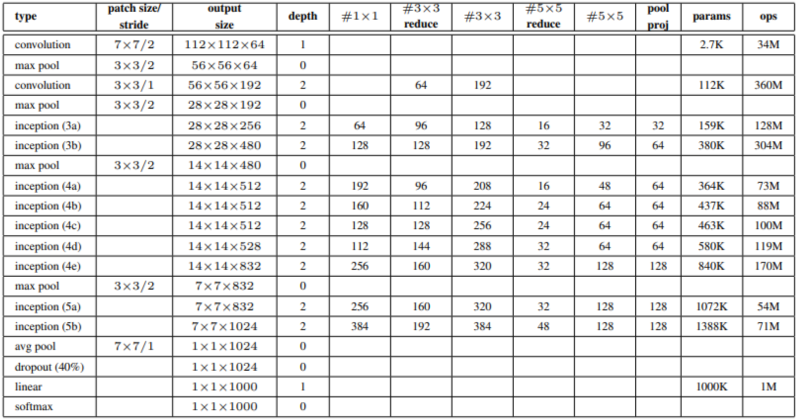
## GoogLeNet

- As above, $1 \times 1$ convolutions are used to decrease the number of parameters of the architecture, hence allowing to increase the depth of the architecture.
- Global average pooling is also used to decrease the number of trainable parameters.
- **Auxiliary Classifier for Training:**
  The inception architecture uses some intermediate classifier branches in the middle of the architecture during training. Each branch consist of (1) a $5 \times 5$ average pooling layer with a stride of 3, (2) a $1 \times 1$ convolutions with 128 filters, (3) two fully connected layers of 1024 outputs and (4) a softmax classification layer with 1000 outputs. It includes dropout regularization with dropout probability 0.7.

These methods help in combating gradient vanishing problem and provide regularization.

## GoogLeNet

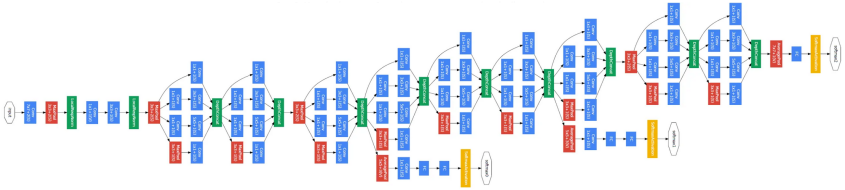
Below is a layer-by-layer description of GoogLeNet architecture



| type | patch size/ stride | output size | depth | #1×1 | #3×3 reduce | #3×3 | #5×5 reduce | #5×5 | pool proj | params | ops |
|---|---|---|---|---|---|---|---|---|---|---|---|
| convolution | 7×7/2 | 112×112×64 | 1 | | | | | | | 2.7K | 34M |
| max pool | 3×3/2 | 56×56×64 | 0 | | | | | | | | |
| convolution | 3×3/1 | 56×56×192 | 2 | | 64 | 192 | | | | 112K | 360M |
| max pool | 3×3/2 | 28×28×192 | 0 | | | | | | | | |
| inception (3a) | | 28×28×256 | 2 | 64 | 96 | 128 | 16 | 32 | 32 | 159K | 128M |
| inception (3b) | | 28×28×480 | 2 | 128 | 128 | 192 | 32 | 96 | 64 | 380K | 304M |
| max pool | 3×3/2 | 14×14×480 | 0 | | | | | | | | |
| inception (4a) | | 14×14×512 | 2 | 192 | 96 | 208 | 16 | 48 | 64 | 364K | 73M |
| inception (4b) | | 14×14×512 | 2 | 160 | 112 | 224 | 24 | 64 | 64 | 437K | 88M |
| inception (4c) | | 14×14×512 | 2 | 128 | 128 | 256 | 24 | 64 | 64 | 463K | 100M |
| inception (4d) | | 14×14×528 | 2 | 112 | 144 | 288 | 32 | 64 | 64 | 580K | 119M |
| inception (4e) | | 14×14×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 840K | 170M |
| max pool | 3×3/2 | 7×7×832 | 0 | | | | | | | | |
| inception (5a) | | 7×7×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 1072K | 54M |
| inception (5b) | | 7×7×1024 | 2 | 384 | 192 | 384 | 48 | 128 | 128 | 1388K | 71M |
| avg pool | 7×7/1 | 1×1×1024 | 0 | | | | | | | | |
| dropout (40%) | | 1×1×1024 | 0 | | | | | | | | |
| linear | | 1×1×1000 | 1 | | | | | | | 1000K | 1M |
| softmax | | 1×1×1000 | 0 | | | | | | | | |

## GoogLeNet

The overall architecture is 22 layers deep. The architecture contains two auxiliary classifier layer connected to the output of Inception (4a) and Inception (4d) layers.



## ResNet

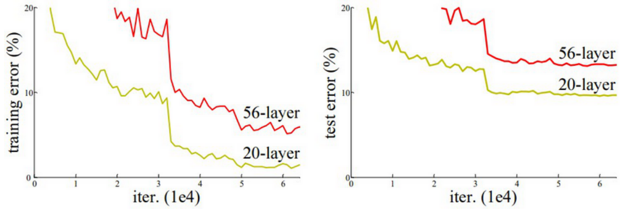
ResNet (Residual Network) [He et al 2016], with an architecture consisting of **152 layers**, was the winner of ILSVRC 2015.

Remark: After the first CNN-based architecture, AlexNet, won the ImageNet competition in 2012, every subsequent winning architecture used more layers in a deep neural network to reduce the error rate.

However, increasing the network depth revealed a problem in deep learning associated with the **vanishing gradient** (or exploding gradient). This phenomenon causes the gradient to become 0 (or too large) so that training and test error rate increase when the number of layers is getting too large.

## ResNet

To illustrate the vanishing gradient problem, the following numerical experiments compares the training and test error of two standard CNNs, with 20 and 56 layers respectively.

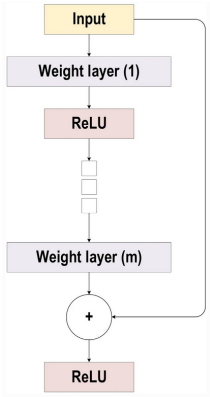


In the above plot, we can observe that a 56-layer CNN gives more error rate on both training and testing dataset than a 20-layer CNN architecture.

The analysis of the error rate reveals that it is caused by vanishing/exploding gradients.

## ResNet

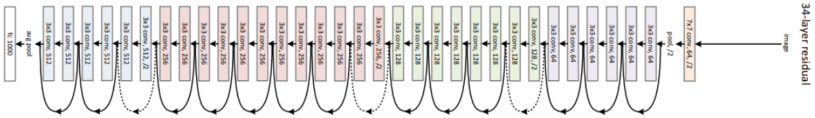
To solve the problem of the vanishing/exploding gradient, the novel idea of ResNet is the use of **skip connections** (also called bypass pathways), originally introduced in the Highway Nets [Srivastava et al, 2015].



## ResNet

The skip connection connects activations of a layer to further layers by skipping some layers in between. This forms a **residual block**.

Resnets are made by stacking these residual blocks together.

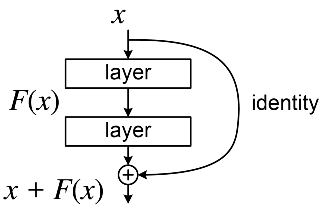


ResNet prevents the problem of diminishing gradients as the shortcut connections (residual links) accelerate the deep network convergence.

This is achieved at relatively low computational cost.
In comparison with VGG, ResNet has lower computational complexity, even with enlarged depth.

## ResNet

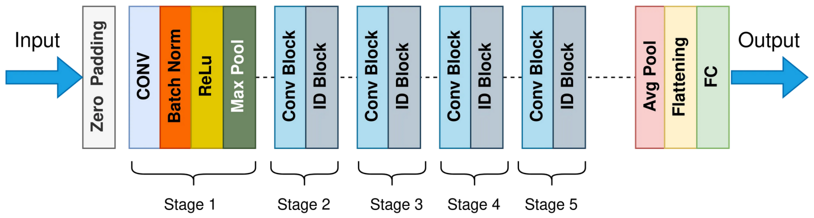
The conceptual novelty of ResNet is that, instead of learning the underlying mapping, we allow the network to **fit the residual mapping**.



Consider a residual block with a certain number of stacked layers. Denote the underlying function performed by this subnetwork as $H(x)$ where $x$ is the input to this subnetwork. The idea of **Residual Learning** re-parameterizes this subnetwork and lets the parameter layers represent a residual function $F(x) := H(x) - x$, so that the ouptut of the subnetwork is $y = F(x) + x$.

## ResNet

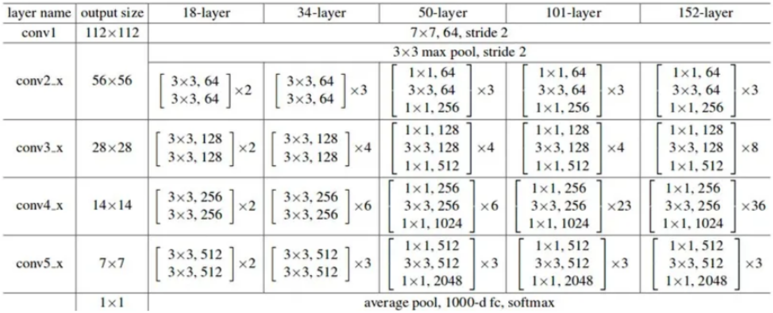
The ResNet architecture is organized into an input stage, followed by 5 processing stages and an output stage.



- Input image is zero-padded.
- Different versions of the ResNet architecture use a varying number of residual blocks at the different stage levels.
- The output layers include Average Pooling, Flattening and finally a fully connected layer reducing its input to the number of classes using a softmax activation.

## ResNet

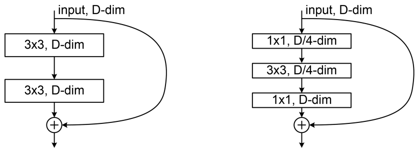
A detailed, informative listing of the structure of the residual blocks for different ResNet architectures is given below



| layer name | output size | 18-layer | 34-layer | 50-layer | 101-layer | 152-layer |
|---|---|---|---|---|---|---|
| conv1 | 112×112 | 7×7, 64, stride 2 | | | | |
| conv2_x | 56×56 | 3×3 max pool, stride 2 | | | | |
| | | $\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$ | $\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$ | $\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$ | $\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$ | $\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$ |
| conv3_x | 28×28 | $\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$ | $\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$ | $\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$ | $\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$ | $\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$ |
| conv4_x | 14×14 | $\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$ | $\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$ | $\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$ | $\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$ | $\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$ |
| 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 | | | | |

## ResNet

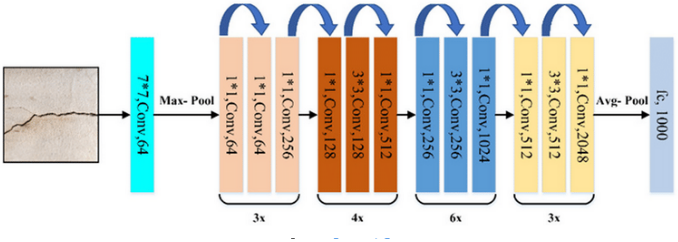
The models of ResNet-50, ResNet-101, and ResNet-152 are all based on **Bottleneck Blocks**.



Left: A **Basic Block** consists of two sequential $3 \times 3$ convolutional layers and a residual connection.
Right: A **Bottleneck Block** consists of three sequential convolutional layers and a residual connection. The first layer is a $1 \times 1$ convolution for dimension reduction, e.g., to $1/4$ of the input dimension; the second layer performs a $3 \times 3$ convolution; the last layer is another $1 \times 1$ convolution for dimension restoration

## ResNet

Let us examine the architecture of ResNet50



It contains 1 input convolutional layer, $3 \cdot 3 + 4 \cdot 3 + 6 \cdot 3 + 3 \cdot 3 = 48$ layers in the stages 1-4 and 1 output fully connected layer.

Resnet50 in PyTorch

## ResNet

**Stage 1**: It includes 1 convolution layer, batch normalization, ReLU and max pooling.

The convolutional layer takes as input an image with dimensions (224, 224, 3). This layer uses 64 filters, each with a kernel size of (7, 7) and a stride of (2, 2) to downsample the input image by a factor of 2 in both the width and height dimensions. It outputs a feature map with dimensions (112, 112, 64).

After the convolutional layer, a batch normalization layer is applied to normalize the activations, followed by a ReLU.

Finally, a max pooling layer with a pool size of (3, 3) and a stride of (2, 2) is applied to further downsample the feature map by a factor of 2 in both dimensions.
This produces a feature map with dimensions (56, 56, 64), which serves as the input to the subsequent convolutional block.

## ResNet

**Stage 2**: It consists of 3 Residual blocks containing 3 layers each. The size of kernels used to perform the convolution operation in all 3 layers of the block are 64, 64 and 256 respectively. The size of the feature output is the same as the input, (56, 56, 64).
**Stage 3,4,5**: They consist of 4,6,3 Residual blocks respectively, containing 3 layers each. For each stage, the first residual block of the stage is modified with the the convolution operation in the Residual Block performed with stride 2. Hence, the size of input is reduced to half in terms of height and width but the channel width will be doubled.
As we progress from one stage to another, the channel width is doubled and the size of the input is reduced to half, so that the output of Stage 5 is (7, 7, 512).
**Output Stage**: Finally, the network has an Average Pooling layer followed by a fully connected layer having 1000 neurons (ImageNet class output).

## Inception-ResNet

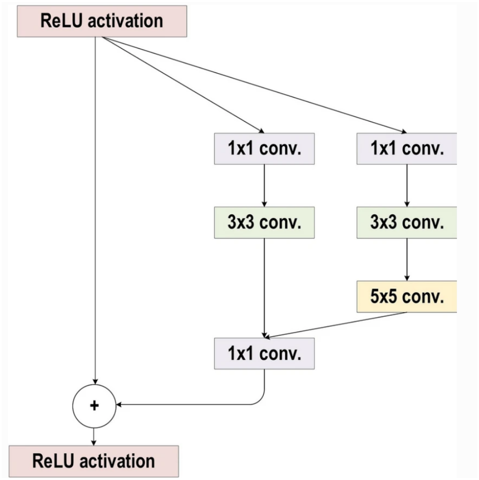
Szegedy et al. proposed **Inception-ResNet**, **Inception V3**, **Inception V4** as upgraded versions of GoogLeNet/Inception-V1 (and V2).

Most notably, Inception-ResNet (also called Inception-4 with residual connections) [Szegedy et al, 2016] bring together the inception block and the residual learning power by replacing the filter concatenation with the residual connection.

Inception-ResNet can achieve a similar generalization power to Inception-V4 with enlarged width and depth and without residual connections. Using residual connections in training it significantly accelerates the Inception network training, highlighting the **benefit of using residual connections**.

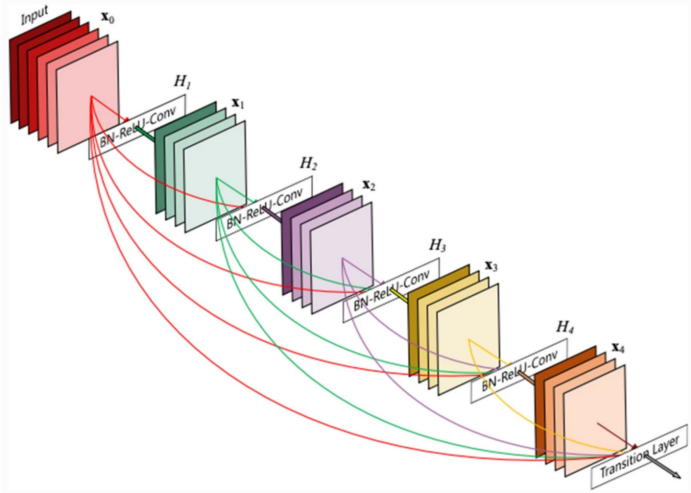
## Inception-ResNet

The block diagram of the **Inception Residual unit** shows the branches of the inception block together with a skip connection.



## DenseNet

**DenseNet**, which was proposed by Huang et al. in 2017, follows the same direction as ResNet and the Highway network, generalizing the notion of skip connection.



## DenseNet

DenseNet attempts to address the following limitations of ResNet:

- ResNet has a large number of weights, as each layer has an isolated group of weights.
- Several layers contribute extremely little or no information.

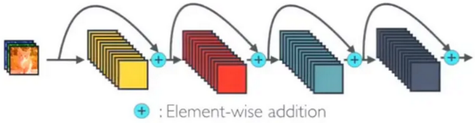
The key idea of DenseNet it to employ cross-layer connectivity by **connecting each layer to all layers** in the network using a feed-forward approach.

This way, the feature maps of each previous layer are employed to input into all of the following layers.

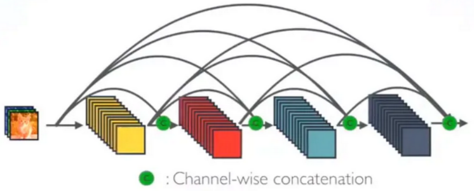
Since each layer receives feature maps from all preceding layers, the **number of units in each layer can be fewer**.

## DenseNet

In ResNet, **element-wise addition** is used to pass a state from one ResNet module to another one.



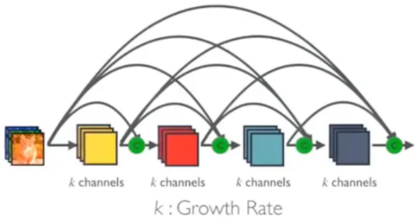
In DenseNet, each layer obtains additional inputs from all preceding layers **by concatenation** and passes on its own feature-maps to all subsequent layers.



Each layer is receiving a “collective knowledge” from all preceding layers.

## DenseNet

Since each layer receives feature maps from all preceding layers and the feature maps are concatenated after each block, the unit/channel dimension is increasing.



If the output of a layer produces $k$ feature maps every time, then for the $l$-th layer we have

$$k_l = k_o + k(l-1)$$

This hyperparameter $k$ is the **growth rate**.

## DenseNet

The **growth rate** $k$ is the additional number of units for each layer of a DenseNet.

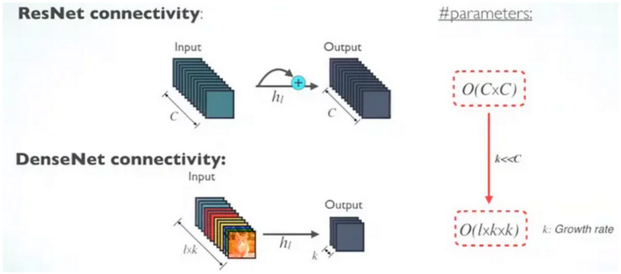
The growth rate regulates how much information is added to the network each layer.

Since the unit/channel dimension is increasing, network can be thinner, i.e. number of units/channels can be fewer.

Higher computational efficiency and memory efficiency w.r. to ReNet.

## DenseNet

For each layer, the number of parameters in ResNet is directly proportional to $C \times C$ (where $C$ is the number of units) while number of parameters in DenseNet is directly proportional to $l \times k \times k$ where $k$ is the growth rate and $l$ is the number of preceding layers.



Since $k \ll C$, DenseNet has much smaller size than ResNet.

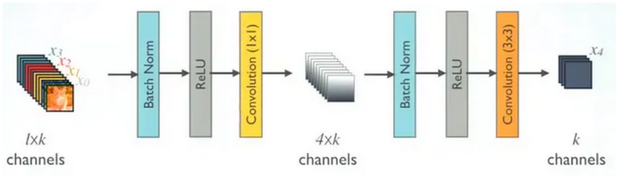
ResNet50: 25.6M parameters; DenseNet-201: 20.0M parameters.

## DenseNet Architecture

The architecture of a DenseNet includes:

- **Dense Blocks**, where the dimensions of the feature maps remains constant within a block, but the number of filters changes between them;
- **Transition Layers**, used between different Dense Blocks to take care of the downsampling applying a batch normalization, a 1x1 convolution and a 2x2 pooling layers.

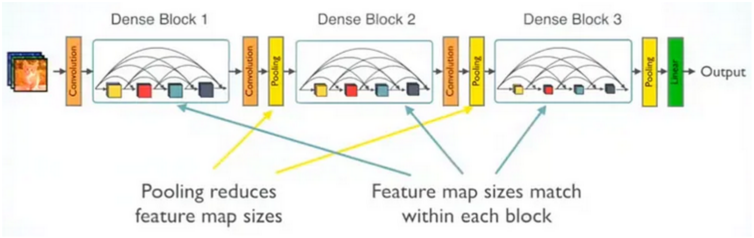
A Dense Block consists of several Dense Layers, also called DenseNets-B (Bottleneck), that employ $1 \times 1$ convolution to reduce the feature maps size before the $3 \times 3$ convolution.



DenseNet-B

# DenseNet Architecture

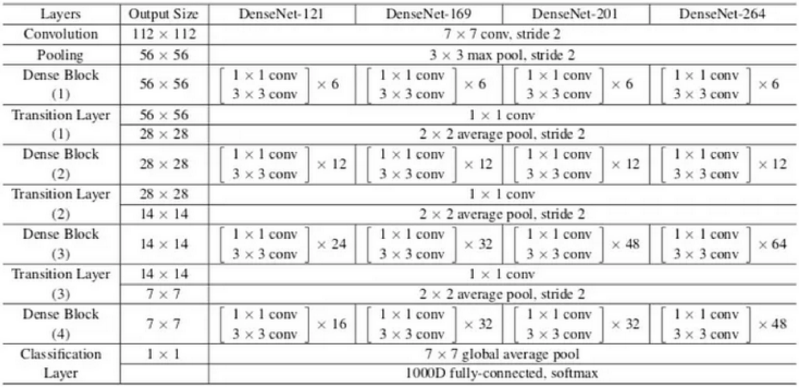
Multiple Dense Blocks are combined with Transition Layers



- Feature map sizes are the same within the dense block so that they can be concatenated together.
- Transition layers reduce feature map sizes
- At the end of the last dense block, a global average pooling is performed followed by a softmax classifier.

## DenseNet - Architectures

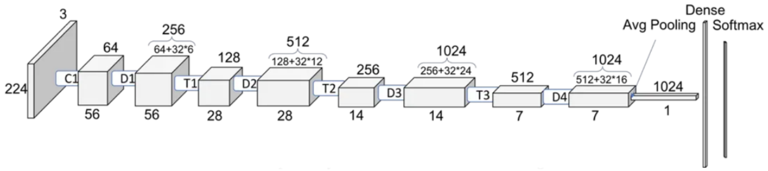
Sizes of outputs and convolutional kernels for different DenseNets architectures, designed for ImageNet classification (output: 1,000 classes)



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

## DenseNet-121

Architecture of DenseNet-121, with input $224 \times 224 \times 3$.



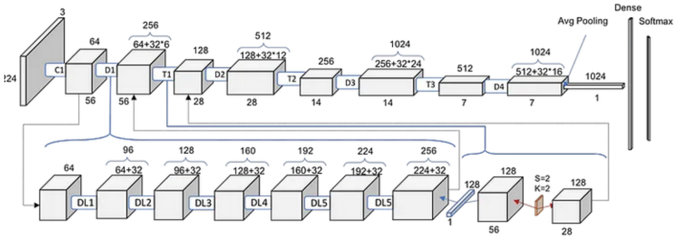
C1: Convolution Block; Dx: Dense Block x; Tx: Transition Block x.

The numbers under each volume represent the sizes of the width and depth, the numbers on top represent the feature maps dimension.

Note: width and depth decrease in successive blocks.

## DenseNet-121

The volume after every Dense Block increase by the growth rate times the number of Dense Layers within that Dense Block.

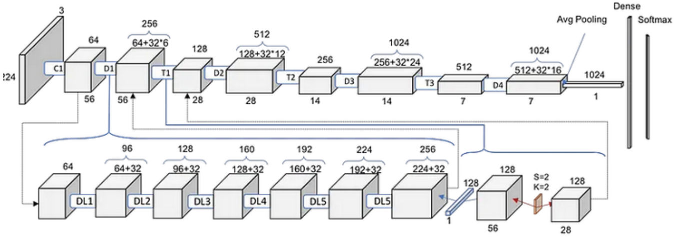


DLx: Dense Layer x; Growth rate = 32

Every layer is adding to the previous volume 32 new feature maps. This is why we go from 64 to 256 after 6 layers.

## DenseNet-121

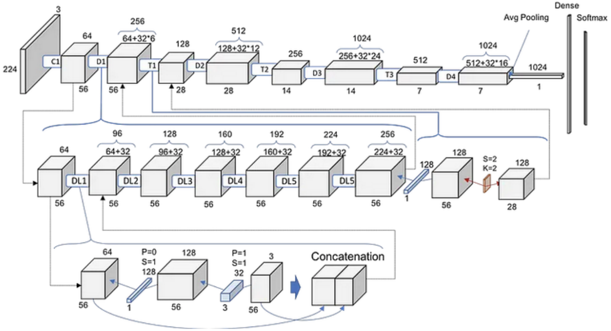
The volume and the feature maps are halved after every Transition Block



The Transition Block performs as $1 \times 1$ convolution with 128 filters, followed by a $2 \times 2$ pooling with a stride of 2, resulting on dividing the size of the volume and the number of feature maps on half. In the visulized block, it goes from 256 to 128.

## DenseNet-121

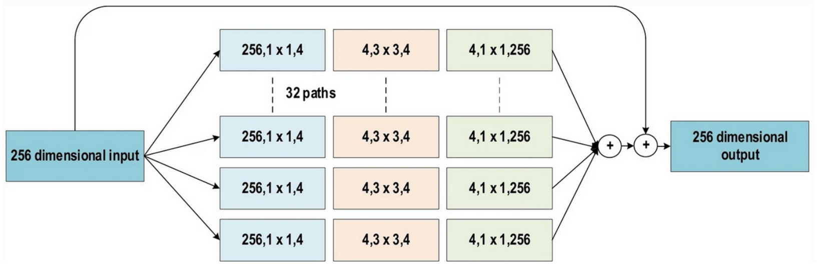
The figure shows how the first Dense Layer within the first Dense Block is adding 32 times the number of layers.



First, we perform a $1 \times 1$ convolution with 128 filters to reduce the feature maps size and next we perform a $3 \times 3$ convolution (with padding, to ensure dimensions remain constant) with this chosen 32 number of feature maps. Finally, the input volume and the result of the two operations are concatenated.

## ResNext

**ResNext** [Xie et al, 2017] is an enhanced version of the Inception Network also known as the Aggregated Residual Transform Network.



It combines ideas from ResNet, VGG, and Inception.

## ResNext

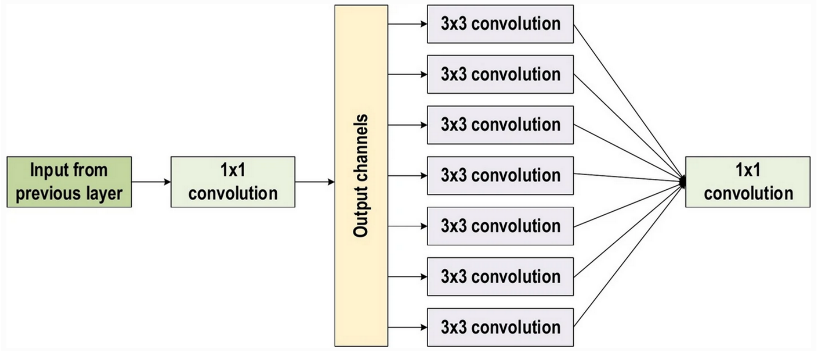
ResNeXt includes

- skip connections from the previous block to next block, adapted from ResNet;
- stacking layers to build a deep architecture model, adapting VGG;
- split-transform-merge strategy adapted from Inception, where input is split into multiple blocks and merged blocks later.

It reports improved classification performance on ImageNet with respect to ResNet using similar network depth and paramater size.

## Xception

The Xception [Chollet et al, 2017] model (meaning Extreme Version of Inception) adjusts the original inception block by making it wider and rearranging it.



As the original inception, it uses a **depthwise convolution** consisting of a channel-wise $n \times n$ spatial convolution.

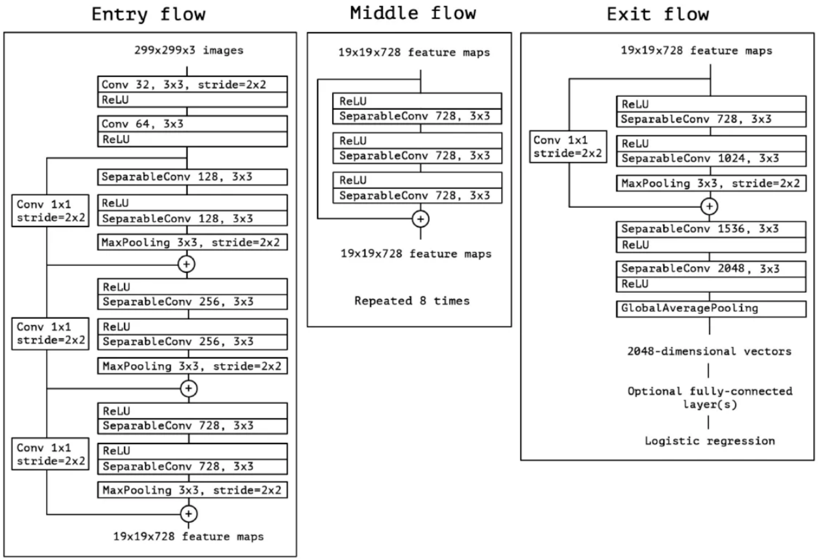
## Xception

The original depthwise separable convolution was inplemented depthwise convolution followed by a $1 \times 1$ convolution.

In the modified depthwise separable convolution, the $1 \times 1$ convolution convolution is followed by a depthwise convolution.

The Xception architecture also includes skip connections

## Xception - Architecture



## CapsNets

**Capsule Neural Networks** or **CapsNets** [Sabour, Frosst, Hinton, 2017] were introduce to overcome the limitations of CNNs related to learning hierarchical structures and spatial relationships.

CapsNets are designed to emulate **hierarchical relationships**, drawing inspiration from the hierarchical organization observed in biological neural systems.

The fundamental building block of a CapsNet is called a **capsule.**

A capsule is a group of neurons whose outputs represent different properties of the same entity.

Unlike neurons that work with scalars, capsules process inputs by encapsulating the result in informative **vectors** through an affine transformation.

## CapsNets

Compared to CNNs, CapsNets have a number of advantages.

- **Improved handling of hierarchies.** By capturing the spatial hierarchies among features, CapsNets improve representation learning by handling hierarchical relationships in data efficiently.
- **Reduced need for data augmentation.** Compared to CNNs, CapsNets frequently require less data augmentation.
- **Improved robustness to adversarial attacks.** CapsNets demonstrate heightened resilience against adversarial attacks due to their use of vector representations, making it challenging for attackers to influence the network with minor input perturbations.
- **Explicit pose information.** CapsNets encode pose information, enhancing structured representations for tasks like object recognition and pose estimation, where understanding spatial relationships is crucial.