VIETNAM GENERAL CONFEDERATION OF LABOR
TON DUC THANG UNIVERSITY
INFORMATION TECHNOLOGY FACULTY

HOMEWORK 1

DEEP LEARNING

CNN Models

Instructor: **Prof. Lê Anh Cường**
Name: **Đỗ Phạm Quang Hưng**
Student ID: **520K0127**

HO CHI MINH CITY, 2023

Table of content

1. LeNet-5

1.1. Introduction

LeNet was introduced in the research paper "*Gradient-Based Learning Applied To Document Recognition" in the year **1998** by Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Ma*ny of the listed authors of the paper have gone on to provide several significant academic contributions to the field of deep learning.

Yann LeCun, Leon Bottou, Patrick Haffner, and Yoshua Bengio

LeNet-5 was one of the earliest convolutional neural networks and promoted the development of deep learning. Since 1988, after years of research and many successful iterations, the pioneering work has been named LeNet-5.

LeNet-5 was used on large scale to *automatically classify hand-written digits* on bank cheques in the United States. Another reason why LeNet is an important architecture is that before it was invented, character recognition had been done mostly by using feature engineering by hand, followed by a machine learning model to learn to classify hand engineered features. LeNet made hand engineering features redundant, because the network learns the best internal representation from raw images automatically.

1.2. Structure

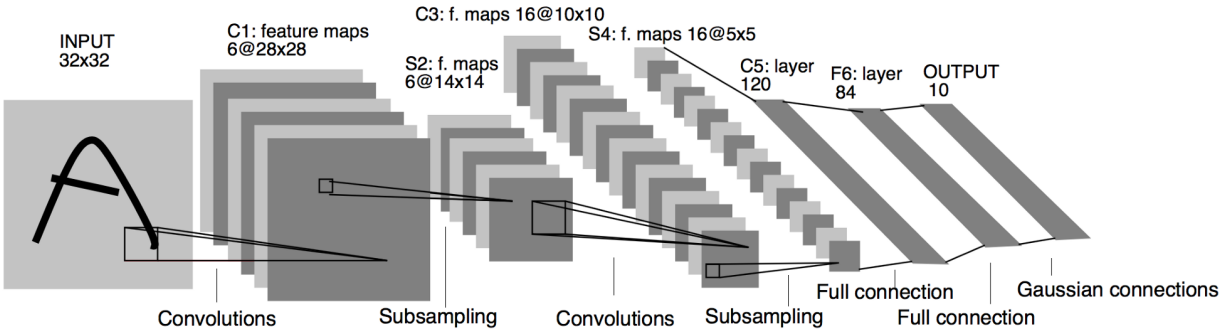


Figure 1: LeNet-5 architecture

As a representative of the early convolutional neural network, LeNet possesses the basic units of convolutional neural network, such as convolutional layer, pooling layer and full connection layer, laying a foundation for the future development of convolutional neural network. As shown in the figure (input image data with 32*32 pixels) : LeNet-5 consists of seven layers. In addition to input, every other layer can train parameters. In the figure, Cx represents convolution layer, Sx represents sub-sampling layer, Fx represents complete connection layer, and x represents layer index.[2][5][6]

- **Layer C1** is a convolution layer with six convolution kernels of 5x5 and the size of feature mapping is 28x28, which can prevent the information of the input image from falling out of the boundary of convolution kernel.
- **Layer S2** is the subsampling/pooling layer that outputs 6 feature graphs of size 14x14. Each cell in each feature map is connected to 2x2 neighborhoods in the corresponding feature map in C1.
- **Layer C3** is a convolution layer with 16 5-5 convolution kernels. The input of the first six C3 feature maps is each continuous subset of the three feature maps in S2, the input of the next six feature maps comes from the input of the four continuous subsets, and the input of the next three feature maps comes from the four discontinuous subsets. Finally, the input for the last feature graph comes from all feature graphs of S2.
- **Layer S4** is similar to S2, with size of 2x2 and output of 16 5x5 feature graphs.
- **Layer C5** is a convolution layer with 120 convolution kernels of size 5x5. Each cell is connected to the 5*5 neighborhood on all 16 feature graphs of S4. Here, since the feature graph size of S4 is also 5x5, the output size of C5 is 1*1. So S4 and C5 are completely connected. C5 is labeled as a convolutional layer instead of a fully connected layer, because if LeNet-5 input becomes larger and its structure

remains unchanged, its output size will be greater than 1x1, i.e. not a fully connected layer.

- **F6 layer** is fully connected to C5, and 84 feature graphs are output.

1.3. Features

- Every convolutional layer includes three parts: convolution, pooling, and nonlinear activation functions
- Using convolution to extract spatial features (Convolution was called receptive fields originally)
- Subsampling average pooling layer
- tanh activation function
- Using MLP as the last classifier
- Sparse connection between layers to reduce the complexity of computation

1.4. Application

Recognizing simple digit images is the most classic application of LeNet as it was created because of that.

Yann LeCun et al. created the initial form of LeNet in 1989. The paper Backpropagation Applied to Handwritten Zip Code Recognition demonstrates how such constraints can be integrated into a backpropagation network through the architecture of the network. And it had been successfully applied to the recognition of handwritten zip code digits provided by the U.S. Postal Service.

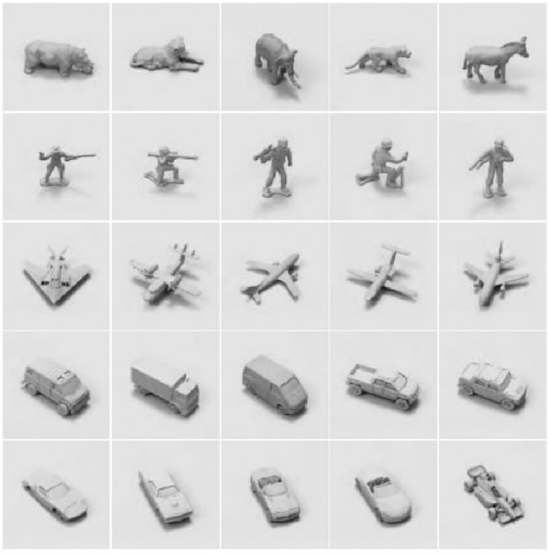
LeNet-5 was able to achieve error rate below 1% on the MNIST data set, which was very close to the state of the art at the time

2. AlexNet

2.1. Introduction

Alexnet won the **Imagenet large-scale visual recognition (ILSVR)** challenge in 2012. The model was proposed in 2012 in the research paper named Imagenet Classification with Deep Convolution Neural Network by **Alex Krizhevsky** and his colleagues.

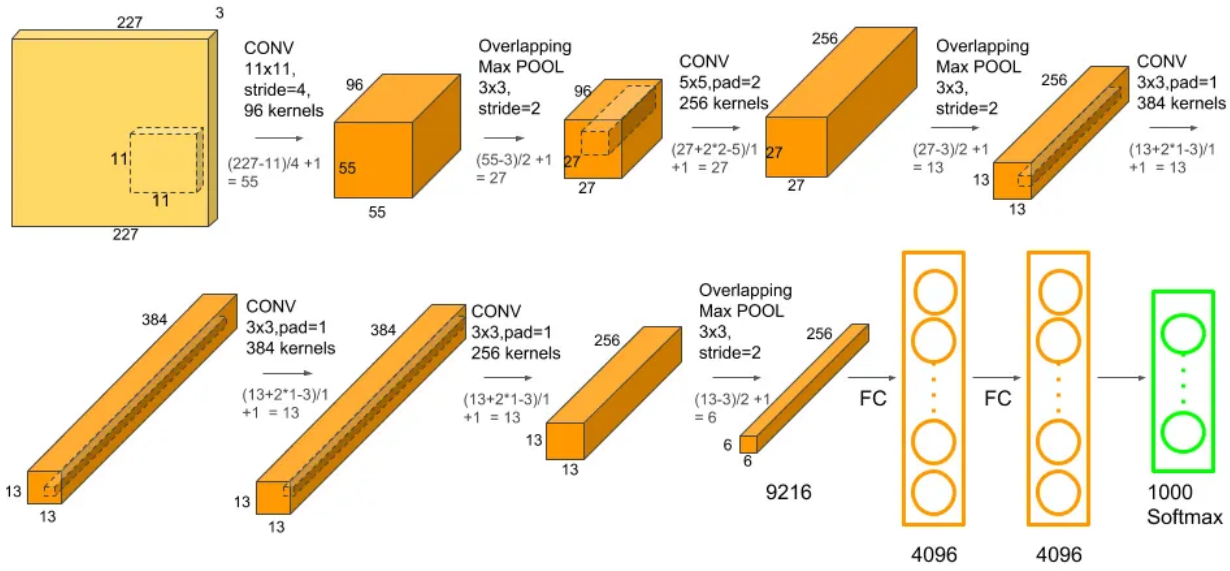
A few years back, we still used small datasets like **CIFAR** and **NORB** consisting of tens of thousands of images. These datasets were sufficient for machine learning models to learn basic recognition tasks. However, real life is never simple and has many more variables than are captured in these small datasets. The recent availability of large datasets like **ImageNet**, which consist of hundreds of thousands to millions of labeled images, have pushed the need for an extremely capable deep learning model. Then came **AlexNet**.



NORB dataset

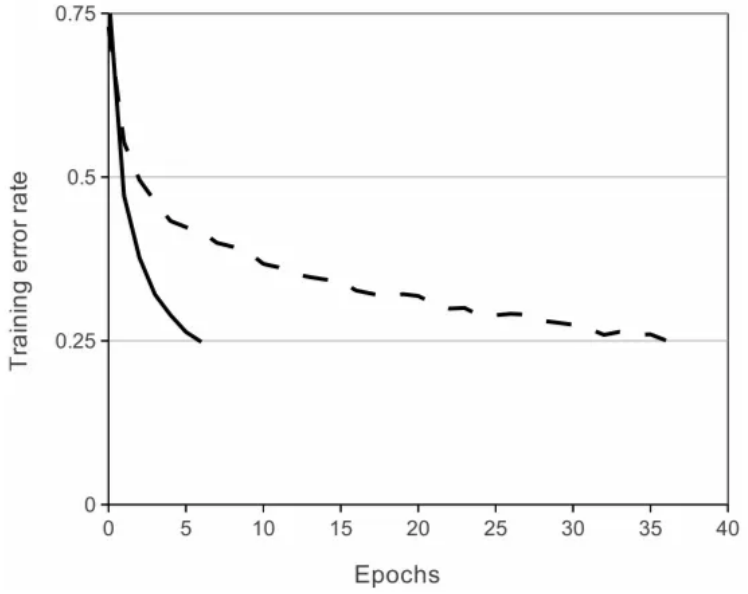
CIFAR dataset

2.2. AlexNet Architecture



The architecture consists of **eight** layers: *five convolutional layers and three fully-connected layers*. But this isn't what makes AlexNet special; these are some of the features used that are new approaches to convolutional neural networks:

- **ReLU Nonlinearity**. AlexNet uses Rectified Linear Units (ReLU) instead of the tanh function, which was standard at the time. ReLU's advantage is in training time; a CNN using ReLU was able to reach a 25% error on the CIFAR-10 dataset six times faster than a CNN using tanh.



Convolutional Neural Networks that use ReLU achieved a 25% error rate on CIFAR-10 six times faster than those that used tanh. Image credits to Krizhevsky et al., the original authors of the AlexNet paper.

- **Multiple GPUs**. Back in the day, GPUs were still rolling around with 3 gigabytes of memory (nowadays those kinds of memory would be rookie numbers). This was especially bad because the training set had 1.2 million images. AlexNet allows for multi-GPU training by putting half of the model's neurons on one GPU and the other half on another GPU. Not only does this mean that a bigger model can be trained, but it also cuts down on the training time.
- **Overlapping Pooling.** CNNs traditionally "pool" outputs of neighboring groups of neurons with no overlapping. However, when the authors introduced overlap, they saw a reduction in error by about 0.5% and found that models with overlapping pooling generally find it harder to overfit.
-

Layer	# filters / neurons	Filter size	Stride	Padding	Size of feature map	Activation function
Input	-	-	-	-	227 x 227 x 3	-
Conv 1	96	11 x 11	4	-	55 x 55 x 96	ReLU
Max Pool 1	-	3 x 3	2	-	27 x 27 x 96	-
Conv 2	256	5 x 5	1	2	27 x 27 x 256	ReLU
Max Pool 2	-	3 x 3	2	-	13 x 13 x 256	-
Conv 3	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 4	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 5	256	3 x 3	1	1	13 x 13 x 256	ReLU
Max Pool 3	-	3 x 3	2	-	6 x 6 x 256	-
Dropout 1	rate = 0.5	-	-	-	6 x 6 x 256	-

Figure: AlexNet layers' detail

The Overfitting Problem. AlexNet had 60 million parameters, a major issue in terms of overfitting. Two methods were employed to reduce overfitting:

- **Data Augmentation**. The authors used label-preserving transformation to make their data more varied. Specifically, they generated image translations and horizontal reflections, which increased the training set by a factor of 2048. They also performed Principle Component Analysis (PCA) on the RGB pixel values to change the intensities of RGB channels, which reduced the top-1 error rate by more than 1%.
- **Dropout**. This technique consists of "turning off" neurons with a predetermined probability (e.g. 50%). This means that every iteration uses a different sample of the model's parameters, which forces each neuron to have more robust features that can be used with other random neurons. However, dropout also increases the training time needed for the model's convergence.

2.3. Pros of AlexNet

- AlexNet is considered as the milestone of CNN for image classification.
- Many methods, such as the conv+pooling design, dropout, GPU, parallel computing, ReLU, are still the industrial standard for computer vision.
- The unique advantages AlexNet is the direct image input to the classification model.

- The convolution layers can automatically extract the edges of the images and fully connected layers learning these features
- Theoretically the com plexity of visual patterns canbe effective extracted by adding more conv layer

2.4. Cons of AlexNet

- AlexNet is NOT deep enough compared to the later model, such as VGGNet, GoogLENet, and ResNet.
- The use of large convolution filters (5*5) is not encouraged shortly after that.
- Use normal distribution to initiate the weights in the neural networks, can not effectively solve the problem of gradient vanishing, replaced by the Xavier method later.
- The performance is surpassed by more complex models such as GoogLENet (6.7%), and ResNet (3.6%)

3. VGG 16

3.1. Introduction

ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) is an annual event to showcase and challenge computer vision models. In the 2014 ImageNet challenge, *Karen Simonyan & Andrew Zisserman from* ***Visual Geometry Group***, Department of Engineering Science, University of Oxford showcased their model in the paper titled "VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION," which won the 1st and 2nd place in object detection and classification.

VGG16 is considered to be one of the best computer vision models to date. The creators of this model evaluated the networks and increased the depth using an architecture with very small (3 × 3) convolution filters, which showed a significant improvement on the prior-art configurations. They pushed the depth to 16–19 weight layers making it approx **— 138 milions trainable parameters**.
VGG16 is object detection and classification algorithm which is able to classify 1000 images of 1000 different categories with 92.7% accuracy. It is one of the popular algorithms for image classification and is easy to use with transfer learning.

VGG – The Idea

So what was new in this model compared to the top-performing models AlexNet-2012 and ZFNet-2013 of the past years? First and foremost, compared to the large receptive fields in the first convolutional layer, this model proposed the use of a very small 3 x 3 receptive field (filters) throughout the entire network with the stride of 1 pixel. Please note that the receptive field in the first layer in AlexNet was 11 x 11 with stride 4, and the same was 7 x 7 in ZFNet with stride 2.

The idea behind using 3 x 3 filters uniformly is something that makes the VGG stand out. Two consecutive 3 x 3 filters provide for an effective receptive field of 5 x 5. Similarly, three 3 x 3 filters make up for a receptive field of 7 x 7. This way, a combination of multiple 3 x 3 filters can stand in for a receptive area of a larger size.

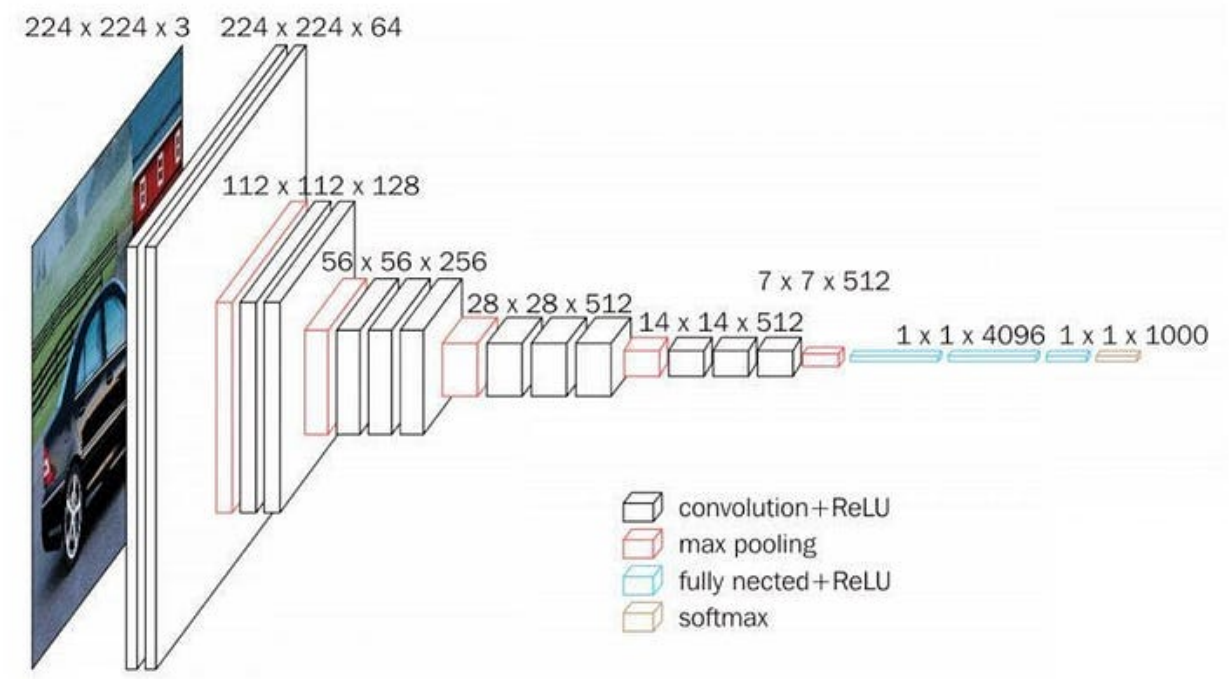
But then, what is the benefit of using three 3 x 3 layers instead of a single 7 x 7 layer? Isn't it increasing the no. of layers, and in turn, the complexity unnecessarily? No. In addition to the three convolution layers, there are also three non-linear activation layers instead of a single one you would have in 7 x 7. This makes the decision functions more discriminative. It would impart the ability to the network to converge faster.

Secondly, it also reduces the number of weight parameters in the model significantly. Assuming that the input and output of a three-layer 3 x 3 convolutional stack have C channels, the total number of weight parameters will be 3 * 32 C2 = 27 C2. If we compare this to a 7 x 7 convolutional layer, it would require 72 C2 = 49 C2, which is almost twice the 3 x 3 layers. Additionally, this can be seen as a regularization on the 7 x 7 convolutional filters forcing them to have a decomposition through the 3 x 3 filters, with, of course, the non-linearity added in-between by means of ReLU activations. This would reduce the tendency of the network to over-fit during the training exercise.

Another question is – can we go lower than 3 x 3 receptive size filters if it provides so many benefits? The answer is "No." 3 x 3 is considered to be the smallest size to capture the notion of left to right, top to down, etc. So lowering the filter size further could impact the ability of the model to understand the spatial features of the image.

The consistent use of 3 x 3 convolutions across the network made the network very simple, elegant, and easy to work with.

3.2. VGG-16 Architecture



The 16 in VGG16 refers to 16 layers that have weights. In VGG16 there are thirteen convolutional layers, five Max Pooling layers, and three Dense layers which sum up to 21 layers but it has only sixteen weight layers i.e., learnable parameters layer.

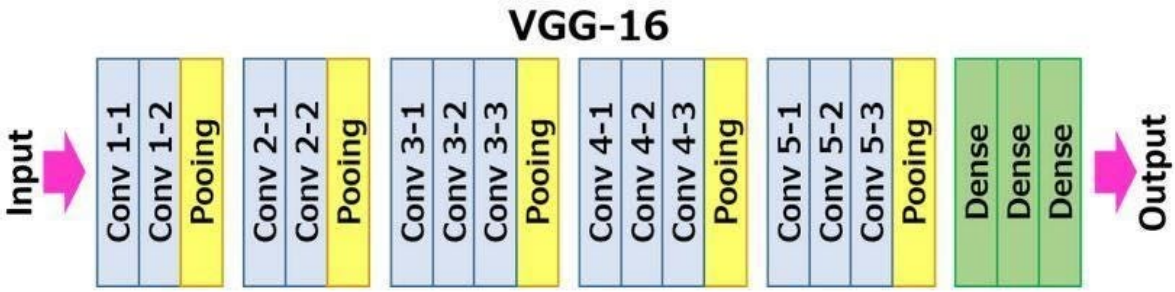
- VGG16 takes input tensor size as 224, 244 with 3 RGB channel.
- VGG16 takes input tensor size as 224, 244 with 3 RGB channel
- Most unique thing about VGG16 is that instead of having a large number of hyper-parameters they focused on having convolution layers of 3x3 filter with stride 1 and always used the same padding and maxpool layer of 2x2 filter of stride 2.
- The convolution and max pool layers are consistently arranged throughout the whole architecture
- Conv-1 Layer has 64 number of filters, Conv-2 has 128 filters, Conv-3 has 256 filters, Conv 4 and Conv 5 has 512 filters.
- Three Fully-Connected (FC) layers follow a stack of convolutional layers: the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer.

3.3. Challenges

Though this is a very simple, elegant, and easy-to-use model, there are some challenges associated with it. The total number of parameters in this model is over 138M, and the size of the model is over 500MB. This puts some serious limitations on the usage of the model, specifically in edge computing, as the inference time required is higher.

Secondly, there is no specific measure available to control the problem of vanishing or exploding gradients. This problem was addressed in GoogLeNet using inception modules and in ResNet using skip connections.

3.4. VGG 16 Use Cases



VGG16 keeps the data scientists and researchers worldwide interested despite the advent of many new and better scoring models since the time VGG was originally proposed. Here are a few use cases where you may find VGG16 practically in use.

- Image Recognition or Classification – VGG16 can be used for disease diagnosis using medical imaging like x-ray or MRI. It can also be used in recognizing street signs from a moving vehicle.
- Image Detection and Localization – We didn't discuss the detection abilities of VGG16 earlier, but it can perform really well in image detection use cases. In fact, it was the winner of ImageNet detection challenge in 2014 (where it ended up as first runner up for classification challenge)
- Image Embedding Vectors – After popping out the top output layer, the model can be used to train to create image embedding vectors which can be used for a problem like face verification using VGG16 inside a Siamese network.

4. Inception

4.1. Introduction

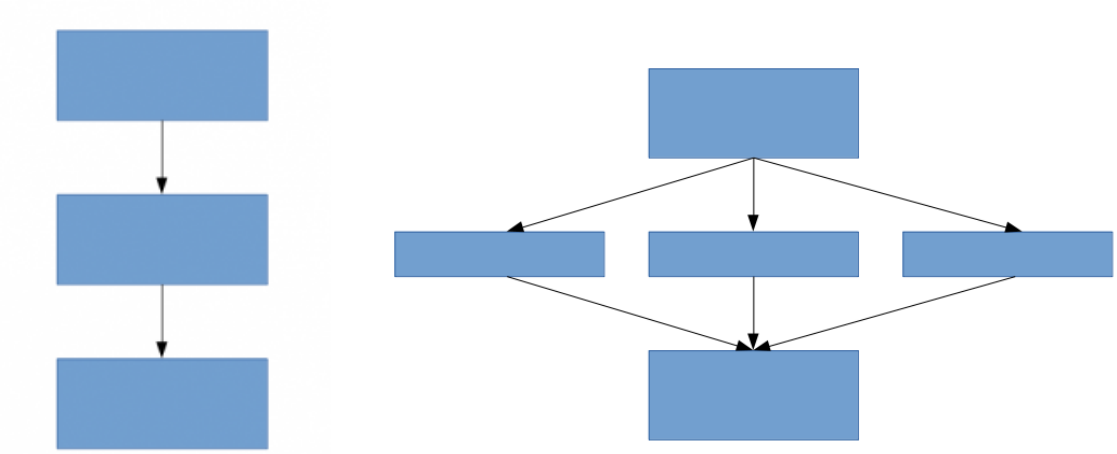
Inception network was once considered a state-of-the-art deep learning architecture (or model) for solving image recognition and detection problems. It put forward a breakthrough performance on the ImageNet Visual Recognition Challenge (in **2014**), which is a reputed platform for benchmarking image recognition and detection algorithms. Along with this, it set off a ton of research in the creation of new deep learning architectures with innovative and impactful ideas.

The origin of the name '**Inception Network**' is very interesting since it comes from the famous movie **Inception, directed by Christopher Nolan**. The movie concerns the idea of dreams embedded into other dreams and turned into the famous internet meme below:

There's a simple but powerful way of creating better deep learning models. *You can just make a bigger model, either in terms of deepness, i.e., number of layers, or the number of neurons in each layer*. But as you can imagine, this can often create complications:

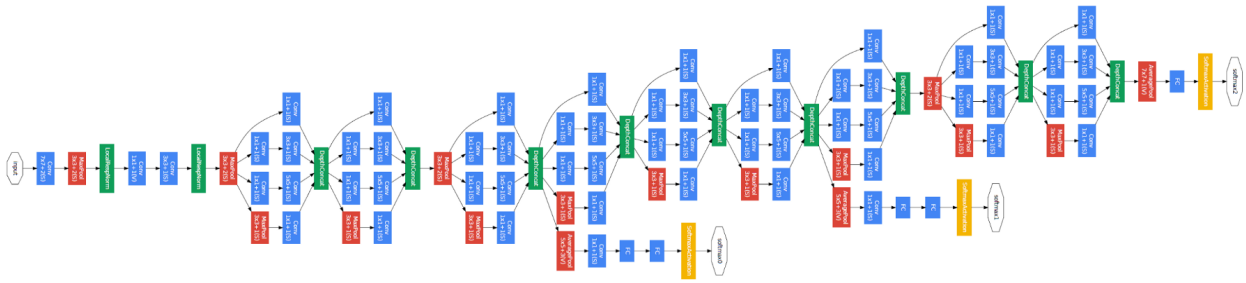
- Bigger the model, more prone it is to overfitting. This is particularly noticeable when the training data is small
- Increasing the number of parameters means you need to increase your existing computational resources

A solution for this, as the paper suggests, is to move on to sparsely connected network architectures which will replace fully connected network architectures, especially inside convolutional layers. This idea can be conceptualized in the below images:



Densely connected architecture *Sparsely connected architecture*

The paper **"Going deeper with convolutions"** from which the hallmark idea of inception network came out. This paper proposes a new idea of creating deep architectures. This approach lets you maintain the "computational budget", while increasing the depth and width of the network. Sounds too good to be true! This is how the conceptualized idea looks:

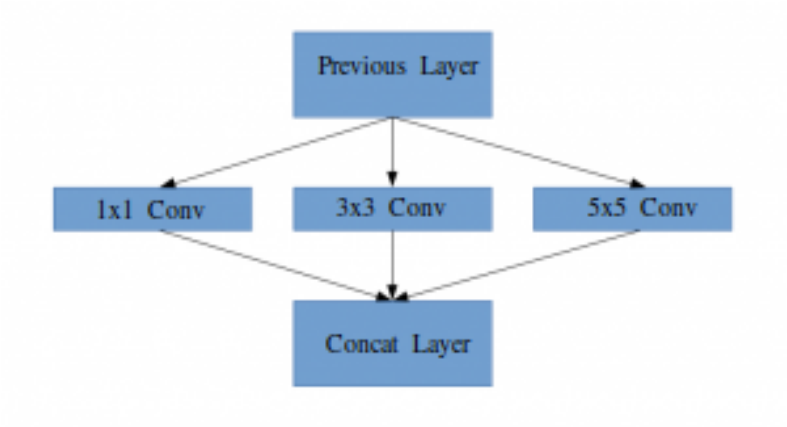


4.2. Proposed Architectural Details

The paper proposes a new type of architecture – GoogLeNet or Inception v1. It is basically a convolutional neural network (CNN) which is 27 layers deep. Below is the model summary:

convolution
max pool
convolution
max pool
inception (3a)
inception (3b)
max pool
inception (4a)
inception (4b)
inception (4c)
inception (4d)
inception (4e)
max pool
inception (5a)
inception (5b)
avg pool
dropout (40%)
linear
softmax

Notice in the above image that there is a layer called inception layer. This is actually the main idea behind the paper's approach. The inception layer is the core concept of a sparsely connected architecture.



Let me explain in a bit more detail what an inception layer is all about. Taking an excerpt from the paper:

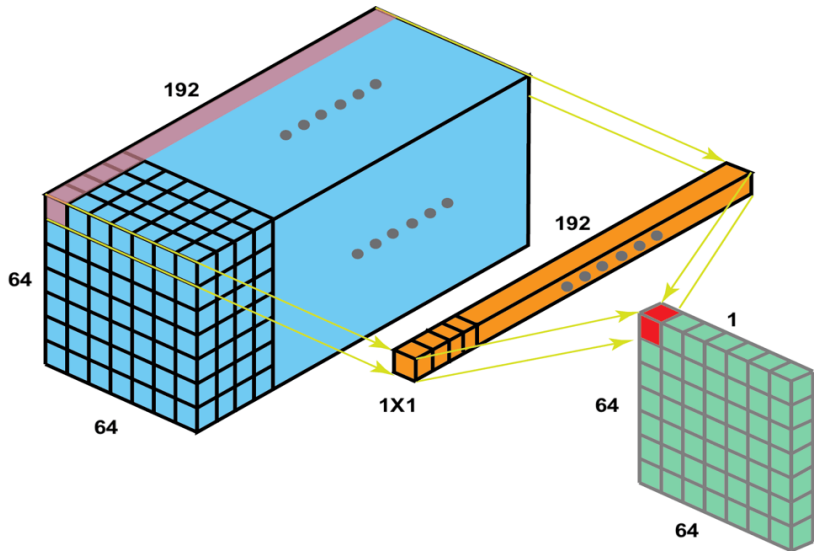
"(Inception Layer) is a combination of all those layers (namely, 1×1 Convolutional layer, 3×3 Convolutional layer, 5×5 Convolutional layer) with their output filter banks concatenated into a single output vector forming the input of the next stage."

4.2.1. 1 x 1 Convolution

The goal of a 1x1 convolution is to reduce the dimensions of the input data by channel-wise pooling. In this way, the depth of the network can increase without running the risk of overfitting.

In a 1 x 1 convolution layer (also referred to as the bottleneck layer), we compute the convolution between each pixel of the image and the filter in the channel dimension. **As a result, the output will have the same height and width as the input, but the number of output channels will change.**

In the image below, we can see an example of a 1 x 1 convolution. The input dimension is (64, 64, 192) while the output dimension is (64, 64, 1). So, the dimension of the output feature map is (64, 64, number of filters):



We can also observe that 1 x 1 convolutions help the network to learn depth features that span across the channels of the image.

4.2.2. 3 x 3 and 5 x 5 Convolutions

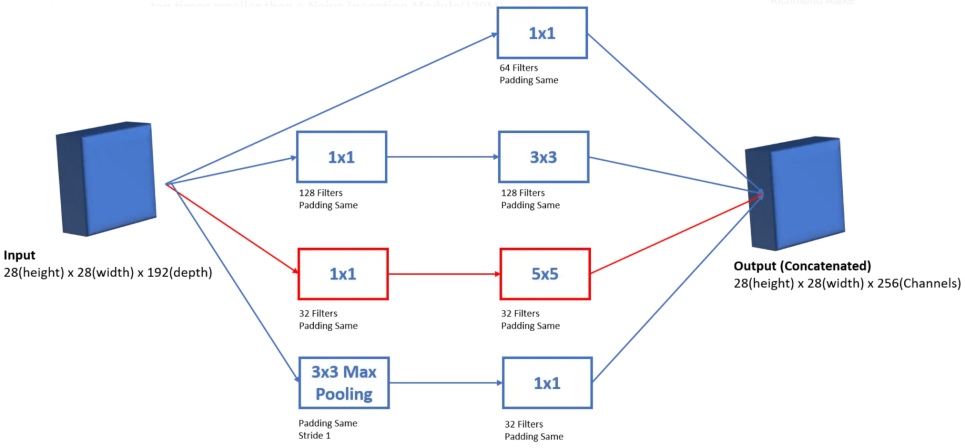
The goal of 3 x 3 and 5 x 5 convolutions is to learn spatial features at different scales. Specifically, by leveraging convolutional filters of different sizes, the network learns spatial patterns at many scales as the human eye does. As we can easily understand, the 3x3 convolution learns features at a small scale while the 5x5 convolution learns features at a larger scale.

4.3. Overall Architecture

Overall, every inception architecture consists of the above inception blocks that we mentioned, along with a max-pooling layer that is present in every neural network and a concatenation layer that joins the features extracted by the inception blocks.
Now, we'll describe two Inception architectures starting from a naive one and moving on to the original one, which is an improved version of the first.

4.3.1. Naive Inception

A naive implementation is to apply every inception block in the input separately and then concatenate the output features in the channel dimension. To do this, we need to ensure that the extracted features have the same width and height dimensions. So, we apply the same padding in every convolution.
Below, we can see an example where the input has dimensions (28, 28, 192) and the output has dimensions (28, 28, channels) where the channels are equal to the sum of the channels extracted from each residual block:

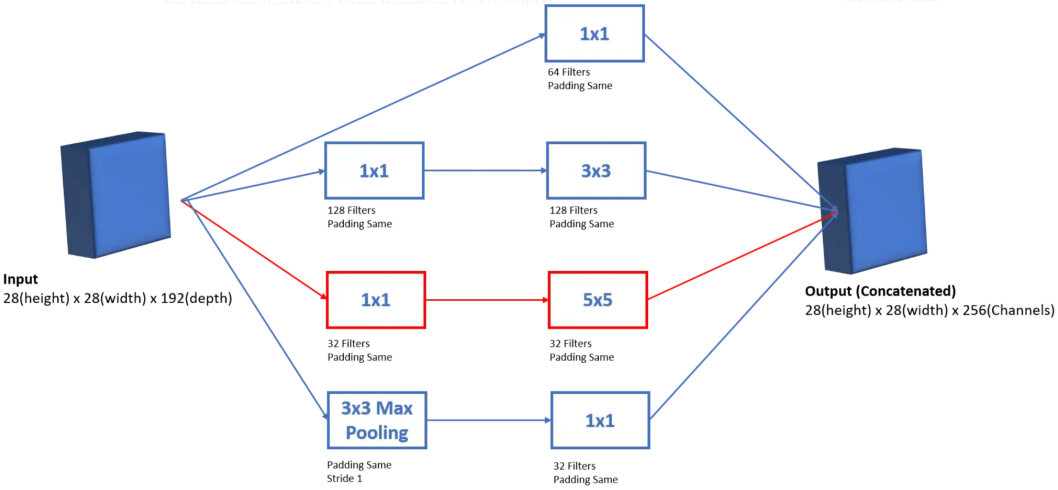


4.3.2. Optimized Inception

The drawback of the naive implementation is the computational cost due to a large number of parameters in the convolutional layers.

The proposed solution is to take advantage of 1x1 convolutions that are able to reduce the input dimensions resulting in much lower computational cost.

Below, we can see how the improved inception architecture is defined. We can see that before each 3x3 and 5x5 convolution we add a 1 \times 1 convolution to reduce the size of the features:



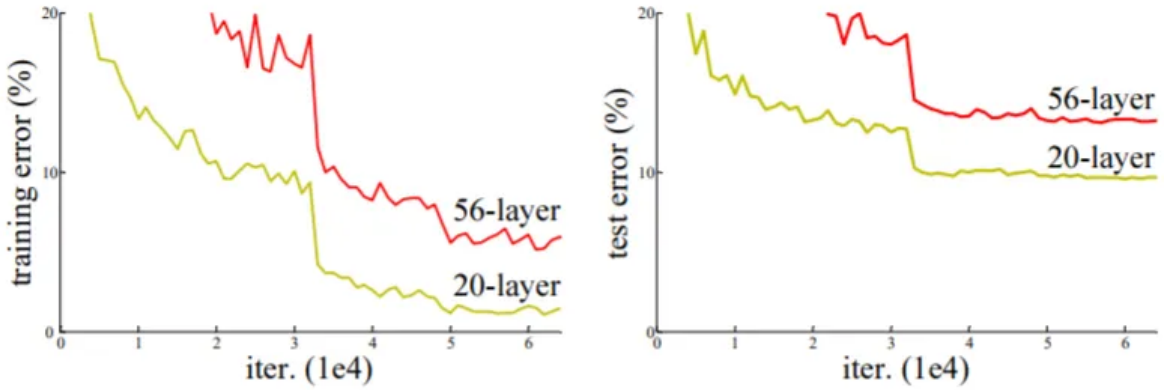
5. ResNet

ResNets are the backbone of many Computer Vision applications

5.1. Introduction

ResNet stands for **Residual Network.** It is an innovative neural network that was first introduced by *Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun* in their 2015 computer vision research paper titled 'Deep Residual Learning for Image Recognition'. This model was immensely successful, as can be ascertained from the fact that its ensemble won the top position at the ILSVRC 2015 classification competition with an error of only 3.57%. Additionally, it also came first in the ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation in the ILSVRC & COCO competitions of 2015.

ResNets or Residual networks are the reason we could finally go very, very deep in neural networks. Everybody needs to know why they work, so, they can take better decisions and make sense of why neural networks work. **AI is not a black box**. Before the advent of **ResNets**, *neural networks with more layers consistently underperformed than the ones having fewer layers (degradation problem)*.

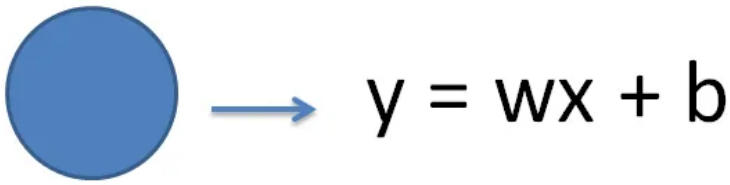


Comparison of 20-layer vs 56-layer architecture (ResNet Paper)

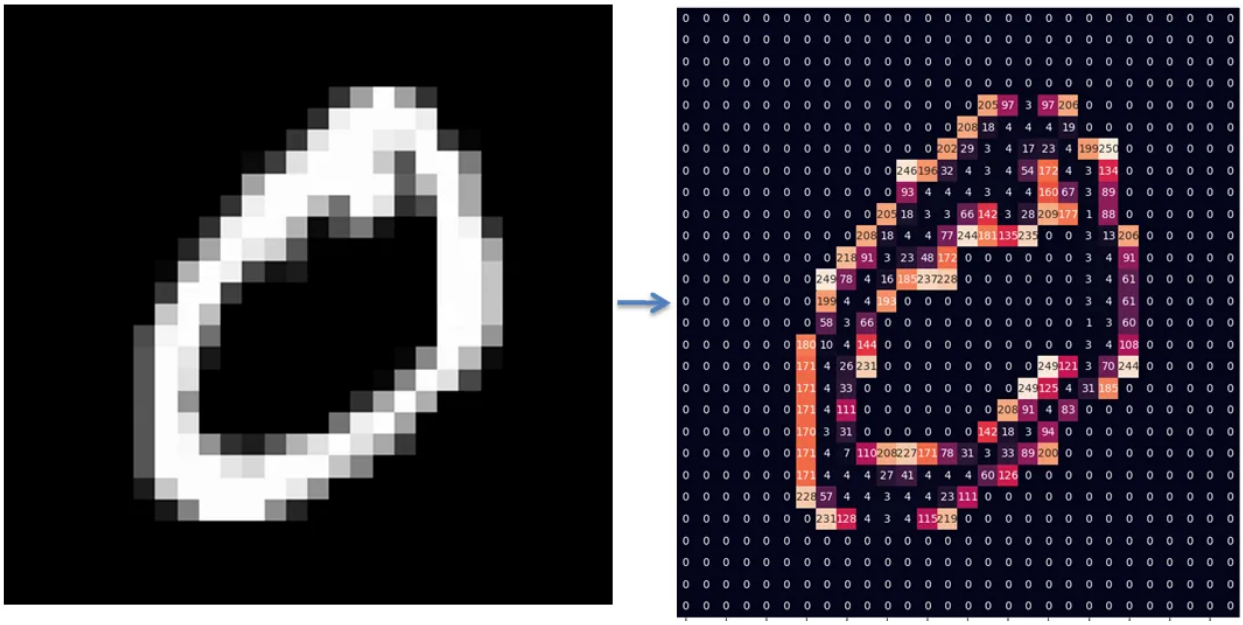
In the above plot, we can observe that a 56-layer CNN gives more error rate on both training and testing dataset than a 20-layer CNN architecture. After analyzing more on error rate the authors were able to reach conclusion that it is caused by vanishing/exploding gradient.

What was the problem?

Let's start from a single neuron.



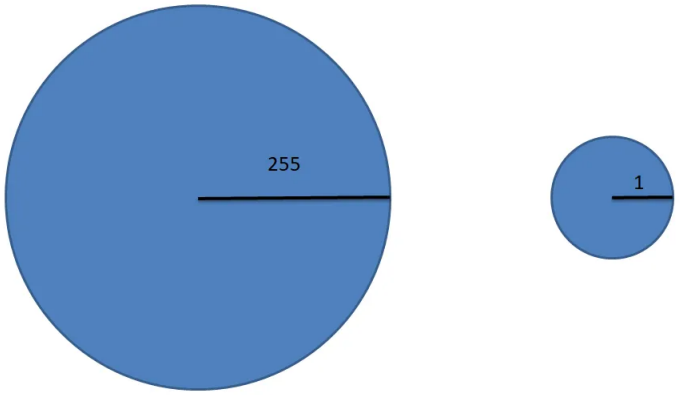
A single neuron has a weight and a bias. The bias is there is keep it alive, lest it be dead when x = 0. Weight actually decides how important the input is. Lower the weight, the less important the input which is coming to it.
The caveat is both weight and bias have values between 0 and 1.
The x is the input. In the case of ResNets, the x will be an image. Let's see what an image looks like before we give it to ResNet.



An image is made up of pixels and each pixel takes a value from 0 to 255. Neural networks take in images as these numbers. Above is an image of numeral 0 converted to numbers for ingestion by a neural network.

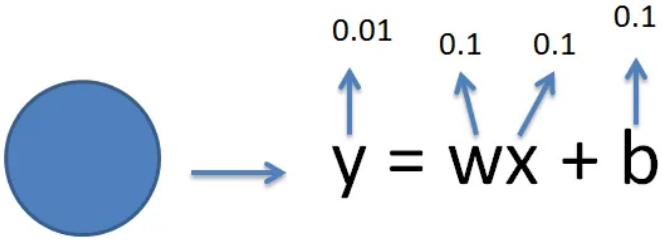
But, before the network drinks this numerical soup, we need to do something more.

`Would you rather run 255 miles in less time or 1 mile?`

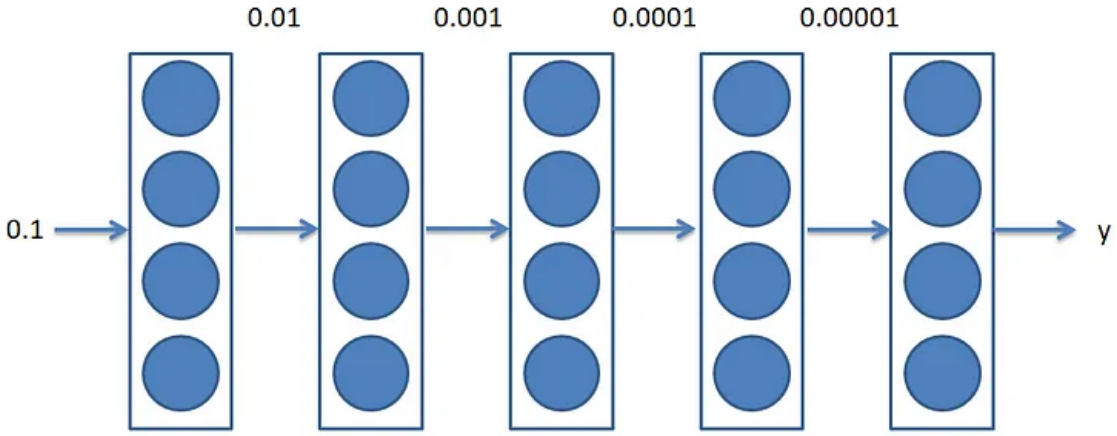


Neural networks also run to the solution, it is easier if they have to run less. **All values are divided by 255** (the maximum). We call this **rescaling** but it is essentially taking the numbers representing the image from circle A to B. It *thus reduces the effort* on the part of the neural network.

What goes into the network is therefore something in the range 0 to 1. Let's see what happens to this value as it goes through a single neuron. Both **w and b are in the range between 0 and 1**. Hence, **y lies in the range 0 and 0.1**.



And if we visualize this process as the data goes through a network, we see that the ***values keep on getting small***.

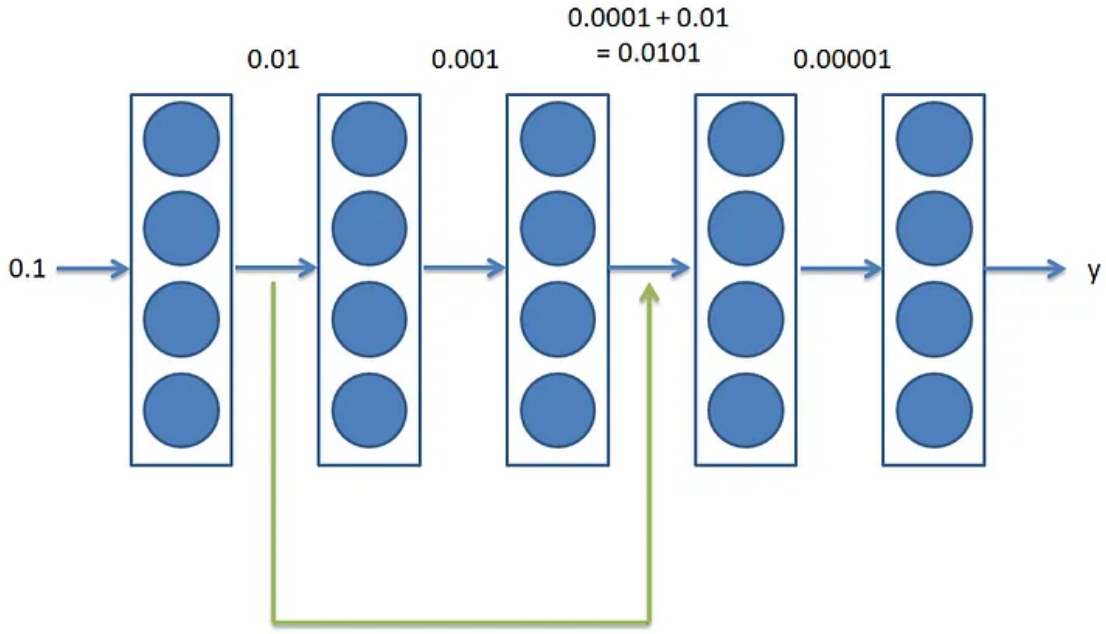


We would finally run into very small values (0.00001). If our values are so small, what we would learn (backpropagated error) will also be less, thus leading to the high training error the big networks give out.

The Solution

Above we understood the values start dying. What do we do when a person's heart starts to die? **CPR**.

Now, when you do CPR for a neural network, it looks like this:



The green line resuscitating the weights is the residual connection which gives the ResNets their name. This is how you describe residual connections in ResNets in a single line:

CPR for neural networks

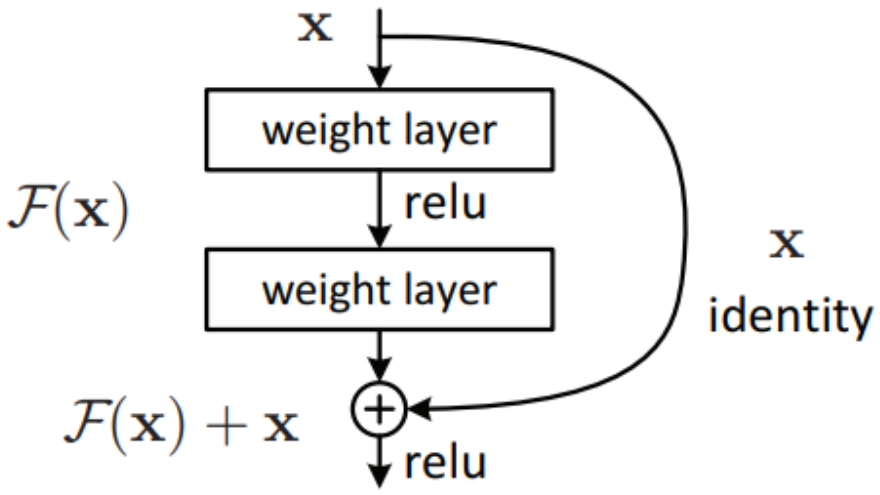
Thus, with residual connections, the values stop dying and what we end up with are some significant values leading to some significant learning in the neural network. That is how and why ResNets are able to be very-very deep while still giving great results.

5.2. Residual block

In order to solve the problem of the vanishing/exploding gradient, this architecture introduced the concept called Residual Blocks. In this network, we use a technique called skip connections. 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.

The approach behind this network is instead of layers learning the underlying mapping, we allow the network to fit the residual mapping. So, instead of say H(x), initial mapping, let the network fit,

F(x) := H(x) - x which gives H(x) := F(x) + x.

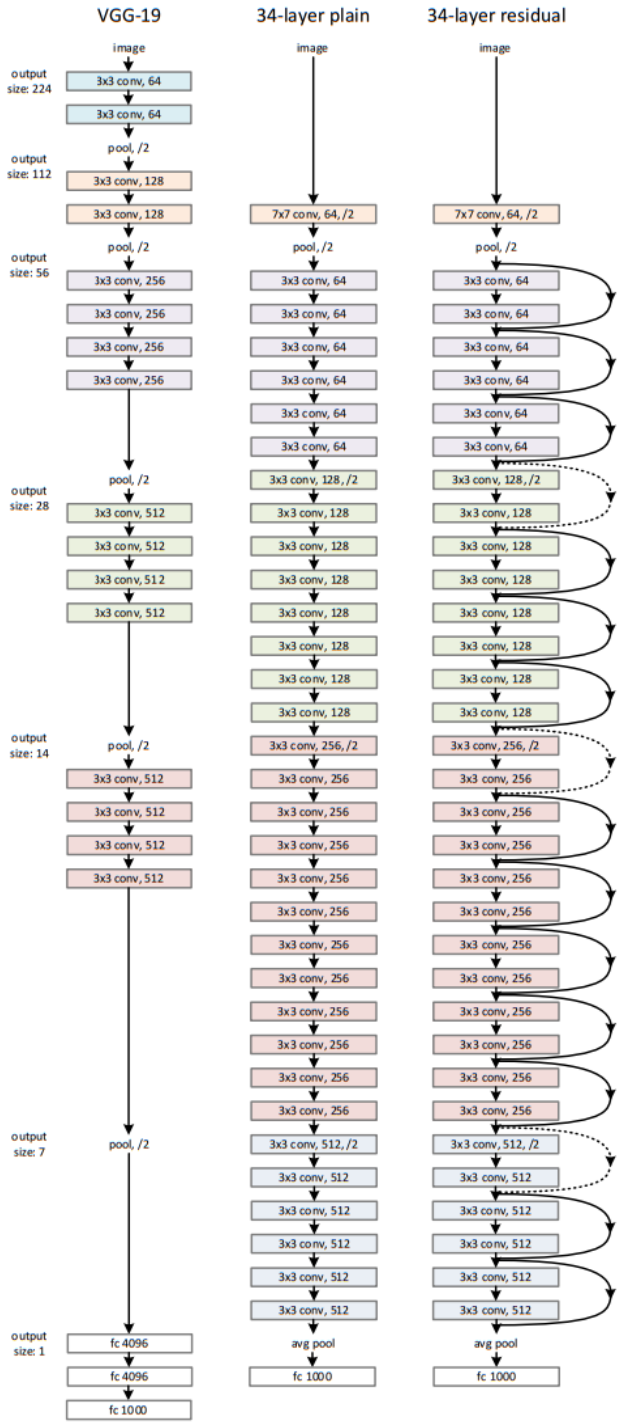


Skip (Shortcut) connection

The advantage of adding this type of skip connection is that if any layer hurt the performance of architecture then it will be skipped by regularization. So, this results in training a very deep neural network without the problems caused by vanishing/exploding gradient. The authors of the paper experimented on 100-1000 layers of the CIFAR-10 dataset.

5.3. Architecture of ResNet

This network uses a 34-layer plain network architecture inspired by VGG-19 in which then the shortcut connection is added. These shortcut connections then convert the architecture into a residual network.



ResNet-50 Architecture

While the Resnet50 architecture is based on the above model, there is one major difference. In this case, the building block was modified into a bottleneck design due to concerns over the time taken to train the layers. This used a stack of 3 layers instead of the earlier 2.
Therefore, each of the 2-layer blocks in Resnet34 was replaced with a 3-layer bottleneck block, forming the Resnet 50 architecture. This has much higher accuracy than the 34-layer ResNet model. The 50-layer ResNet achieves a performance of 3.8 bn FLOPS.

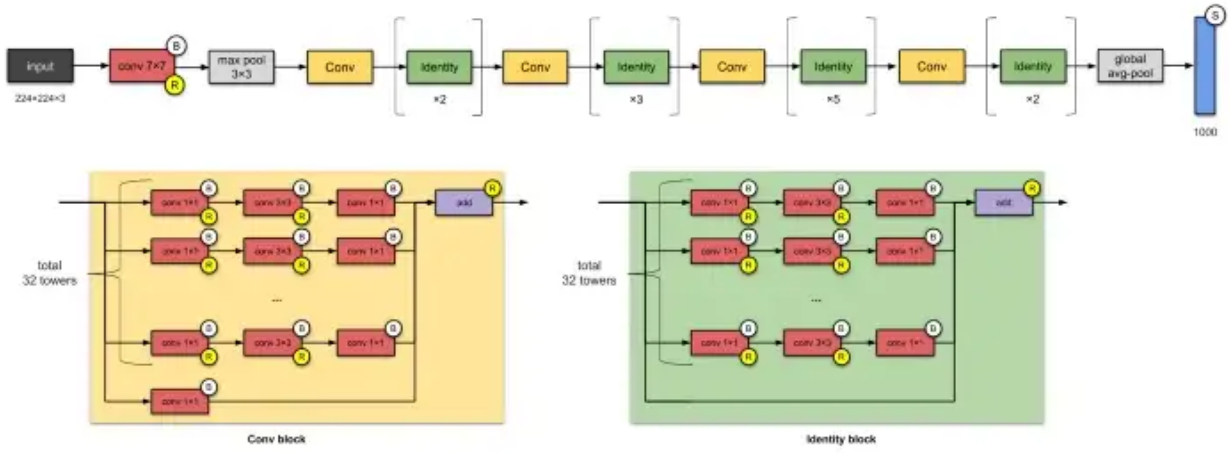
ResNet-101 and ResNet-152 Architecture

Large Residual Networks such as 101-layer ResNet101 or ResNet152 are constructed by using more 3-layer blocks. And even at increased network depth, the 152-layer ResNet has much lower complexity (at 11.3bn FLOPS) than VGG-16 or VGG-19 nets (15.3/19.6bn FLOPS).

6. ResNeXt

6.1. Introduction

ResNext is a simple network for image classification, it uses the same block of layers multiple times that contains a set of transformation functions that helps in classifying the image. As the ResNeXt is made up of same repeated blocks , the architecture is homogenous and the hyperparameters are reduced.



6.2. ResNeXt Architecture Review

ResNeXt won **2nd place in ILSVRC 2016** classification task and also showed performance improvements in Coco detection and ImageNet-5k set than their ResNet counter part.
This is a very simple paper to read which introduces a new term called “cardinality”. The paper simply explains this term and make use of it in ResNet networks and does various ablation studies.

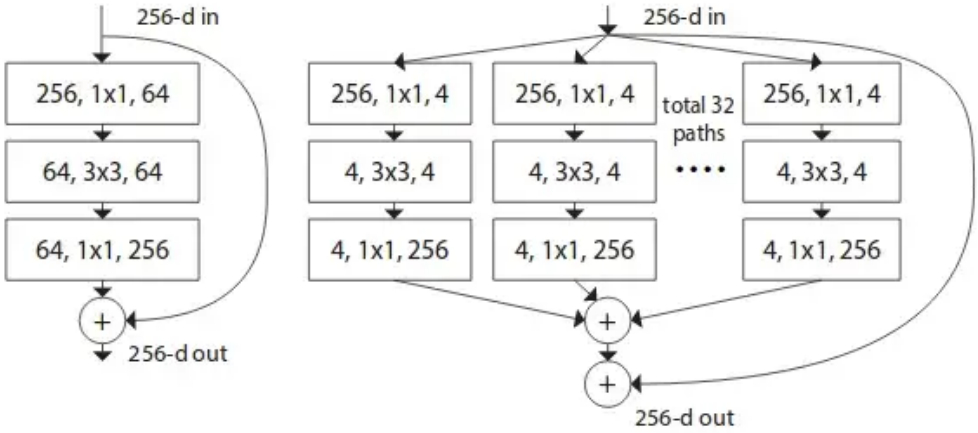


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

- The above diagram distinguishes between a simple ResNet block and ResNeXt blog.
- It follows a split-transform-aggregate strategy.
- The number of paths inside the ResNeXt block is defined as cardinality. In the above diagram C=32
- All the paths contain the same topology.
- Instead of having high depth and width, Having high cardinality helps in decreasing validation error.
- ResNeXt tries embed more subspaces compared to its ResNet counter part.
- Both the architectures have different width. Layer-1 in ResNet has one conv layer with 64 width, while layer-1 in ResNext has 32 different conv layers with 4 width (32*4 width). Despite the larger overall width in ResNeXt, both the architectures have the same number of parameters(~70k) (ResNet 256*64+3*3*64*64+64*26) (ResNeXt C*(256*d+3*3*d*d+d*256), with C=32 and d=4)

Below is the difference in architecture between ResNet and ResNeXt

stage	output	ResNet-50	**ResNeXt-50 (32×4d)**
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128, C=32 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256, C=32 \\ 1\times1, 512 \end{bmatrix}\times4$
conv4	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512, C=32 \\ 1\times1, 1024 \end{bmatrix}\times6$
conv5	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 1024 \\ 3\times3, 1024, C=32 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		$\mathbf{25.5}\times10^6$	$\mathbf{25.0}\times10^6$
FLOPs		$\mathbf{4.1}\times10^9$	$\mathbf{4.2}\times10^9$

Table 1. (**Left**) ResNet-50. (**Right**) ResNeXt-50 with a 32×4d template (using the reformulation in Fig. 3(c)). Inside the brackets are the shape of a residual block, and outside the brackets is the number of stacked blocks on a stage. "C=32" suggests grouped convolutions [24] with 32 groups. *The numbers of parameters and FLOPs are similar between these two models.*

Studies

Cardinality vs width: with C increasing from 1 to 32, we can clearly see a descrease in top-1 % error rate. Therefore, Increasing the C by decreasing the width has improved the performance of the model.

	setting	top-1 error (%)
ResNet-50	1 × 64d	23.9
ResNeXt-50	2 × 40d	23.0
ResNeXt-50	4 × 24d	22.6
ResNeXt-50	8 × 14d	22.3
ResNeXt-50	32 × 4d	**22.2**
ResNet-101	1 × 64d	22.0
ResNeXt-101	2 × 40d	21.7
ResNeXt-101	4 × 24d	21.4
ResNeXt-101	8 × 14d	21.3
ResNeXt-101	32 × 4d	**21.2**

Table 3. Ablation experiments on ImageNet-1K. (**Top**): ResNet-50 with preserved complexity (∼4.1 billion FLOPs); (**Bottom**): ResNet-101 with preserved complexity (∼7.8 billion FLOPs). The error rate is evaluated on the single crop of 224×224 pixels.

Increasing Cardinality vs Deeper/Wider:

Basically 3 cases were studied. 1) Increasing the number of layers to 200 from 101. 2) Going wider by increasing the bottleneck width. 3) Increasing cardinality by doubling C. They have observed that increasing the C gave better performance improvements. below are the results.

	setting	top-1 err (%)	top-5 err (%)
1× complexity references:			
ResNet-101	1 × 64d	22.0	6.0
ResNeXt-101	32 × 4d	21.2	5.6
2× complexity models follow:			
ResNet-**200** [15]	1 × 64d	21.7	5.8
ResNet-101, wider	1 × **100**d	21.3	5.7
ResNeXt-101	**2** × 64d	20.7	5.5
ResNeXt-101	**64** × 4d	**20.4**	**5.3**

Table 4. Comparisons on ImageNet-1K when the number of FLOPs is increased to 2× of ResNet-101's. The error rate is evaluated on the single crop of 224×224 pixels. The highlighted factors are the factors that increase complexity.

7. DenseNet

7.1. Introduction

Densely Connected Convolutional Networks, or DenseNets, are the next step on the way to keep increasing the depth of deep convolutional networks.
The problems arise with CNNs when they go deeper. This is because the path for information from the input layer until the output layer (and for the gradient in the opposite direction) becomes so big, that they can get vanished before reaching the other side.

DenseNets simplify the connectivity pattern between layers introduced in other architectures:

- Highway Networks
- Residual Networks
- Fractal Networks

Regardless of the architectural designs of these networks, they all try to create channels for information to flow between the initial layers and the final layers. DenseNets, with the same objective, create paths between the layers of the network.

The authors solve the problem ensuring maximum information (and gradient) flow. To do it, they simply connect every layer directly with each other.
Instead of drawing representational power from extremely deep or wide architectures, DenseNets exploit the potential of the network through feature reuse.

Key contributions of the DenseNet architecture

- Alleviates vanishing gradient problem
- Stronger feature propagation
- Feature reuse
- Reduced parameter count

7.2. What problem DenseNets solve?

Counter-intuitively, by connecting this way DenseNets require fewer parameters than an equivalent traditional CNN, as there is no need to learn redundant feature maps.

Furthermore, some variations of ResNets have proven that many layers are barely contributing and can be dropped. In fact, the number of parameters of ResNets are big because every layer has its weights to learn. Instead, DenseNets layers are very narrow (e.g. 12 filters), and they just add a small set of new feature-maps.

Another problem with very deep networks was the problems to train, because of the mentioned flow of information and gradients. DenseNets solve this issue since each layer has direct access to the gradients from the loss function and the original input image.

7.3. Structure

Traditional feed-forward neural networks connect the output of the layer to the next layer after applying a composite of operations.

We have already seen that normally this composite includes a convolution operation or pooling layers, a batch normalization and an activation function.
The equation for this would be:

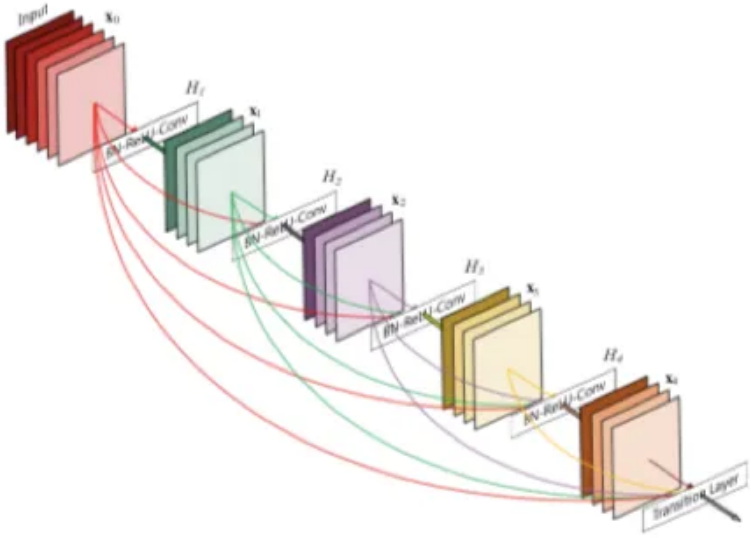
$$x_l = H_l(x_{l-1})$$

ResNets extended this behavior including the skip connection, reformulating this equation into:

$$x_l = H_l(x_{l-1}) + x_{l-1}$$

DenseNets make the first difference with ResNets right here. **DenseNets do not sum the output feature maps of the layer with the incoming feature maps but concatenate them.**

The same problem we faced on our work on ResNets, this grouping of feature maps cannot be done when the sizes of them are different. Regardless if the grouping is an addition or a concatenation. Therefore, and the same way we used for ResNets, DenseNets are divided into **DenseBlocks, where the dimensions of the feature maps remains constant within a block, but the number of filters changes between them**. These layers between them are called ***Transition Layers*** and take care of the downsampling applying a batch normalization, a 1x1 convolution and a 2x2 pooling layers.



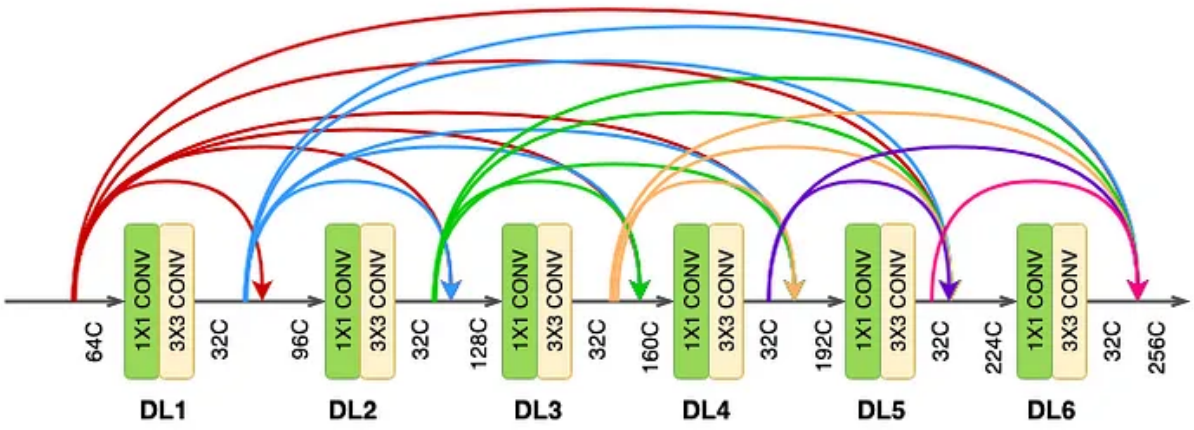
Growth rate (k)

Now we are ready to talk about the growth rate. Since we are concatenating feature maps, this channel dimension is increasing at every layer. If we make H_l to produce k feature maps every time, then we can generalize for the l-th layer:

$$k_l = k_0 + k * (l - 1)$$

This hyperparameter k is the growth rate. The growth rate regulates how much information is added to the network each layer. How so?

We could see the feature maps as the information of the network. Every layer has access to its preceding feature maps, and therefore, to the collective knowledge. Each layer is then adding a new information to this collective knowledge, in concrete k feature maps of information



Dense block with channel count (C) of features entering and exiting the layers

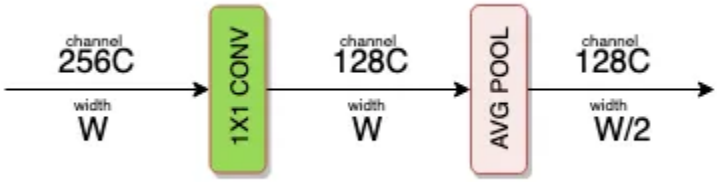
It is basically the number of channels output by a dense-layer (1x1 conv → 3x3 conv). The authors have used a value of k = 32 for the experiments. This means that the number of features received by a dense layer (l) from it's preceding dense layer (l-1) is 32. This is referred to as the growth rate because after each layer, 32 channel features are concatenated and fed as input to the next layer.

Transition layer

At the end of each dense block, the number of feature-maps accumulates to a value of — input features + (number of dense layers x growth rate). So for 64 channel features entering a dense block of 6 dense-layers of growth rate 32, the number of channels accumulated at the end of the block will be — 64 + (6 x 32) = 256. To bring down this channel count, a transition layer (or block) is added between two dense blocks. The transition layer consists of -

- 1 X 1 CONV operation
- 2 X 2 AVG POOL operation

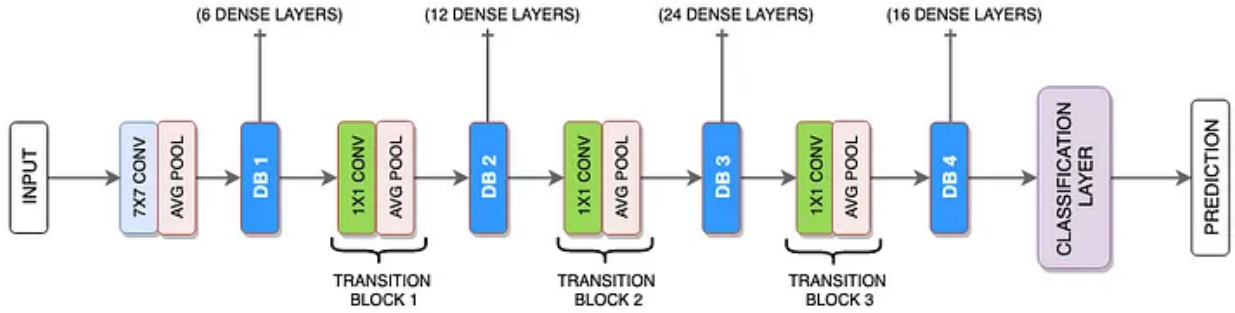
The 1 X 1 CONV operation reduces the channel count to half.
The 2 X 2 AVG POOL layer is responsible for downsampling the features in terms of the width and height.



Transition layer/block

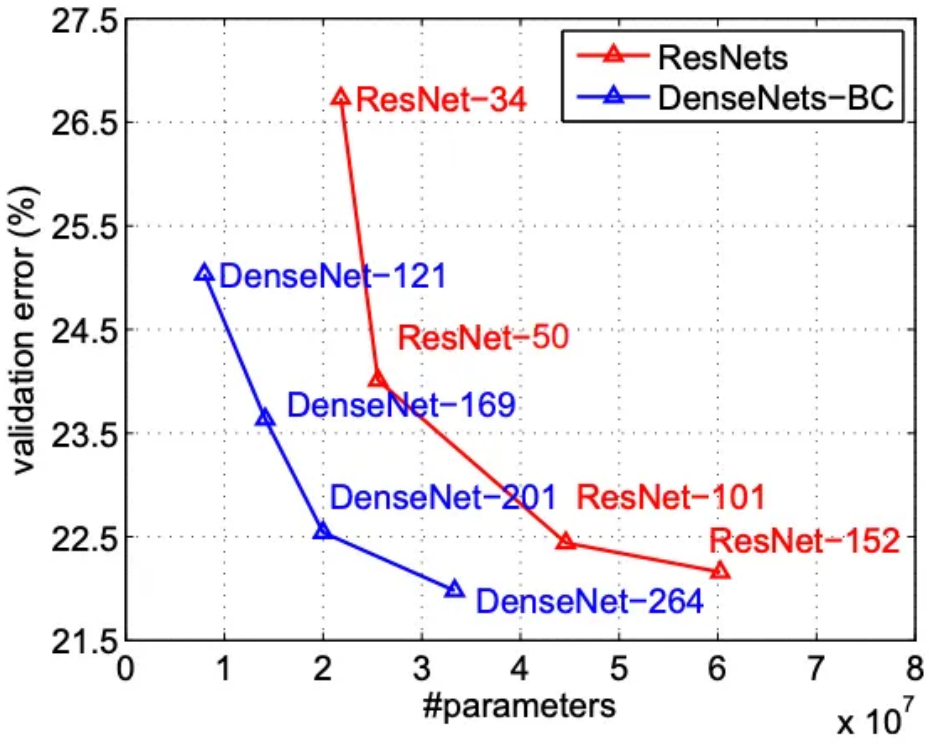
Full network

As can be seen in the diagram below, the authors have chosen different number of dense layers for each of the three dense block.



7.4. Comparison with DenseNet

We can see that even with a reduced parameter count (figure below), the DenseNet model has a significantly lower validation error for the ResNet model with the same number of parameters. These experiments were carried out on both the models with hyper-parameters that suited ResNet better. The authors claim that DenseNet would perform better after extensive hyper-parameter searches.

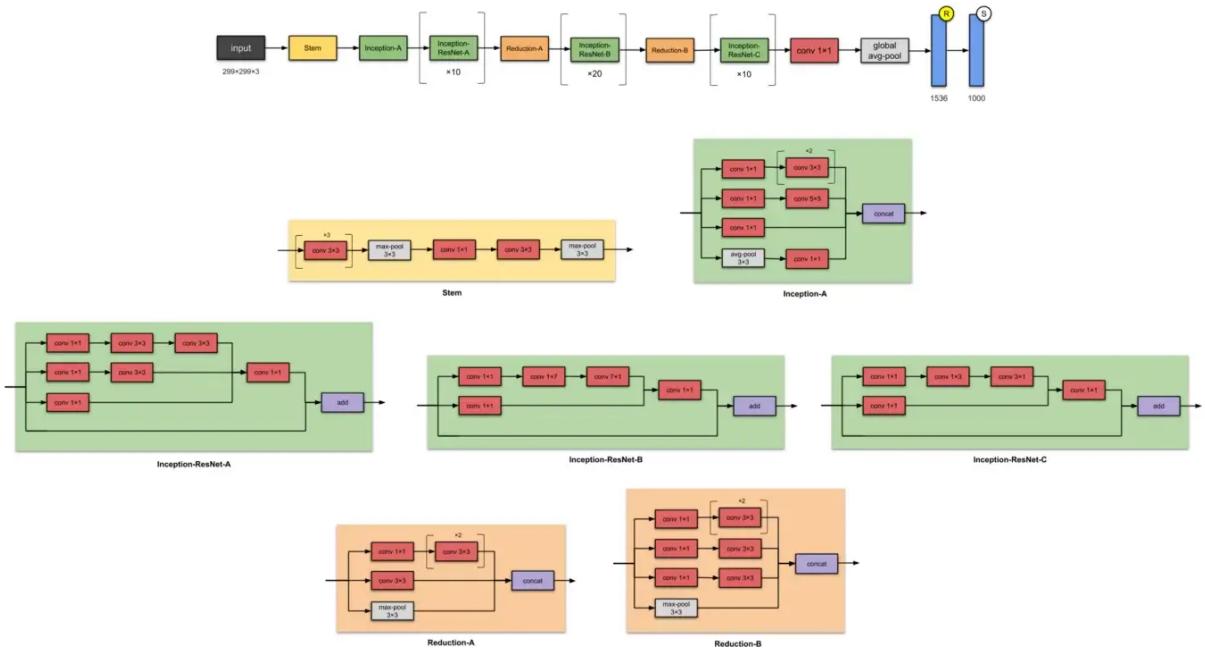


ResNet DenseNet comparison

DenseNet-201 with 20M parameters model yields similar validation error as a 101-layer ResNet with more than 40M parameters.

8. Modern network architectures

8.1 . Inception-ResNet-V2 (2016)



In the same paper as Inception-v4, the same authors also introduced Inception-ResNets — a family of Inception-ResNet-v1 and Inception-ResNet-v2. The latter member of the family has 56M parameters.

✨**What's improved from the previous version, Inception-v3?**

1. Converting Inception modules to Residual Inception blocks.
2. Adding more Inception modules.
3. Adding a new type of Inception module (Inception-A) after the Stem module.

📝**Publication**

Paper: Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning
Authors: Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi. Google
Published in: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence

8.2. Big Transfer (BiT): General Visual Representation Learning (2020)

Even though many variants of ResNet have been proposed, the most recent and famous one is BiT. Big Transfer (BiT) is a scalable ResNet-based model for effective image pre-training [5].

They developed 3 BiT models (small, medium and large) based on ResNet152. For the large variation of BiT they used ResNet152x4, which means that each layer has 4 times more channels. They pretrained that model once in far more bigger datasets than imagenet. The largest model was trained on the insanely large JFT dataset, which consists of 300M labeled images.

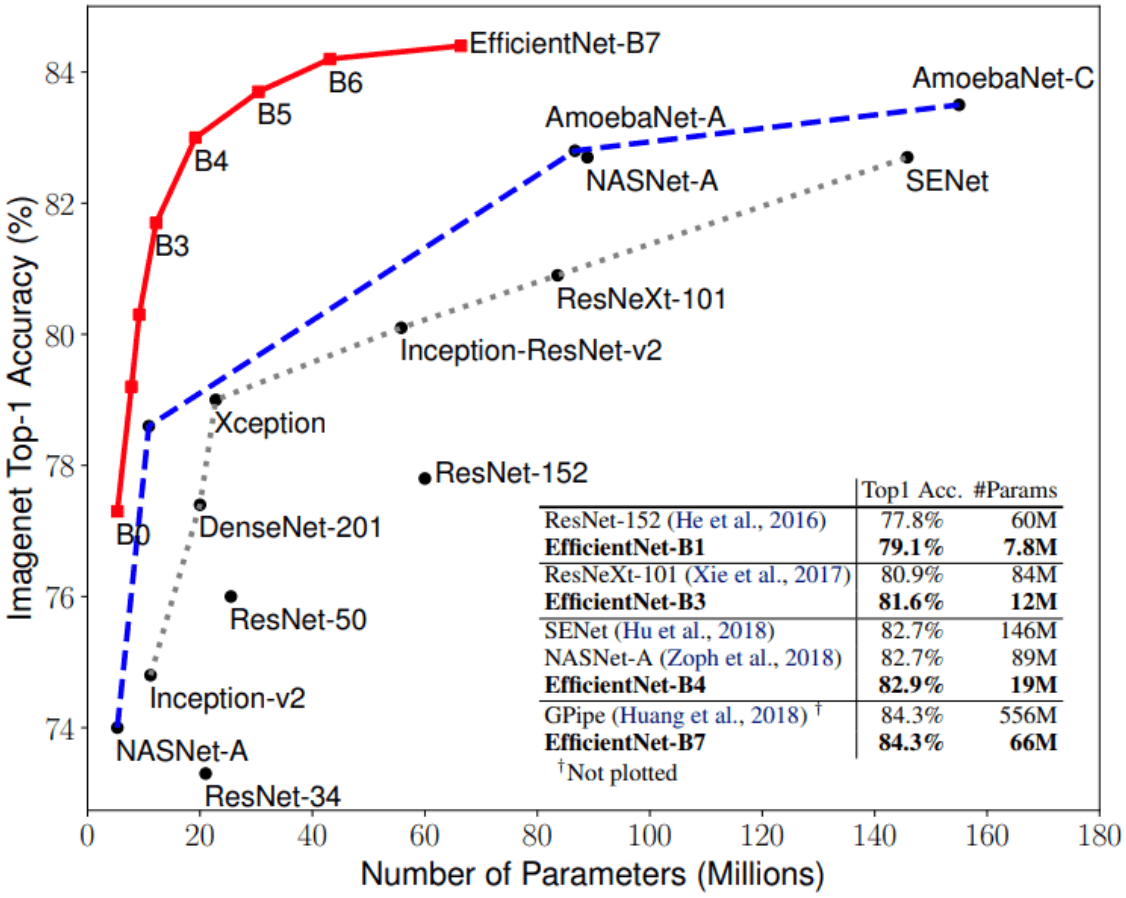
The major contribution in the architecture is the choice of normalization layers. To this end, the authors replaced batch normalization (BN) with group normalization (GN) and weight standardization (WS).

Why? Because first BN's parameters (means and variances) need adjustment between pre-training and transfer. On the other hand, GN doesn't depend on any parameter states. Another reason is that BN uses batch-level statistics, which become unreliable for distributed training in small devices like TPU's. A 4K batch distributed across 500 TPU's means 8 batches per worker, which does not give a good estimation of the statistics. By changing the normalization technique to GN+WS they avoid synchronization across workers.

Obviously, scaling to larger datasets come hand in hand with the model size.

8.3. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks (2019)

EfficientNet is all about engineering and scale. It proves that if you carefully design your architecture you can achieve top results with reasonable parameters.



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

Individual upscaling

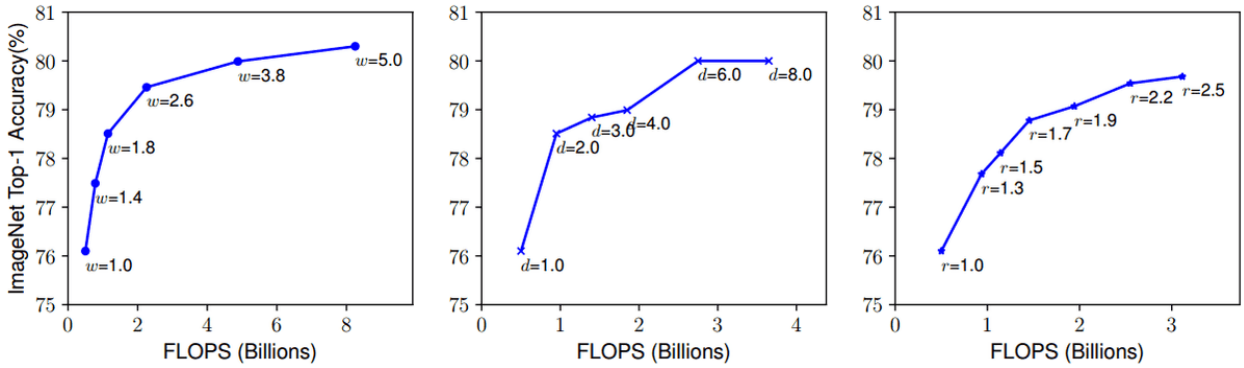
Let's understand how this is possible.

With more layers (depth) one can capture richer and more complex features, but such models are hard to train (due to the vanishing gradients)

- Wider networks are much easier to train. They tend to be able to capture more fine-grained features but saturate quickly.
- By training with higher resolution images, convnets are in theory able to capture more fine-grained details. Again, the accuracy gain diminishes for quite high resolutions

- Instead of finding the best architecture, the authors proposed to start with a relatively small baseline model F and gradually scale it.

That narrows down the design space. To constrain the design space even further, the authors restrict all the layers to uniform scaling with a constant ratio. This way, we have a more tractable optimization problem. And finally, one has to respect the maximum number of memory and FLOPs of our infrastructure.



References

1. LeNet - Wikipedia
2. Key Deep Learning Architectures: LeNet-5 | by Max Pechyonkin | Medium
3. Alexnet Architecture | Introduction to Architecture of Alexnet
4. AlexNet: The Architecture that Challenged CNNs | by Jerry Wei | Towards Data Science
5. Concept of AlexNet:- Convolutional Neural Network | by Abhijeet Pujara | Analytics Vidhya | Medium
6. Vgg 16 Architecture, Implementation and Practical Use | by Abhay Parashar | The Pythoneers | Medium
7. Everything you need to know about VGG16 | by Great Learning | Medium
8. What is VGG16 - Convolutional Network for Classification and Detection (mygreatlearning.com)
9. Understanding Residual Networks (ResNets) Intuitively | by Abhishek Verma | Towards Data Science
10. Understanding Residual Networks (ResNets) Intuitively | by Abhishek Verma | Towards Data Science
11. Deep Residual Networks (ResNet, ResNet50) 2023 Guide - viso.ai
12. Residual Networks (ResNet) - Deep Learning - GeeksforGeeks
13. Inception Network | Implementation Of GoogleNet In Keras (analyticsvidhya.com)
14. Introduction to Inception Networks | Baeldung on Computer Science
15. Understanding and visualizing DenseNets | by Pablo Ruiz | Towards Data Science
16. Paper review: DenseNet -Densely Connected Convolutional Networks | by Mukul Khanna | Towards Data Science
17. ResNeXt architecture (opengenus.org)
18. Understanding and Implementing Architectures of ResNet and ResNeXt for state-of-the-art Image Classification: From Microsoft to Facebook [Part 2] | by Prakash Jay | Medium
19. Illustrated: 10 CNN Architectures | by Raimi Karim | Towards Data Science