K_05 Understanding of ResNet - EN

ResNet is one of the most powerful deep neural networks which has achieved fantabulous performance results in the ILSVRC 2015 classification challenge. ResNet has achieved excellent generalization performance on other recognition tasks and won the first place on ImageNet detection, ImageNet localization, COCO detection and COCO segmentation in ILSVRC and COCO 2015 competitions. There are many variants of ResNet architecture i.e. same concept but with a different number of layers. We have ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-110, ResNet-152, ResNet-164, ResNet-1202 etc. The name ResNet followed by a two or more digit number simply implies the ResNet architecture with a certain number of neural network layers. In this post, we are going to cover ResNet-50 in detail which is one of the most vibrant networks on its own.

Although the object classification problem is a very old problem, people are still solving it to make the model more robust. LeNet was the first Deep Neural Network that came into existence in 1998 to solve the digit recognition problem. It has 7 layers which are stacked up one over the other to recognize the digits written in the Bank Cheques. Despite the introduction of LeNet, more advanced data such as high-resolution images can’t be used to train LeNet. Moreover, the computation power of computer systems during 1998 was very less.

OLD TRICK NEW TWIST:

The Deep Learning community had achieved groundbreaking results during the year 2012 when AlexNet was introduced to solve the ImageNet classification challenge. AlexNet has a total of 8 layers which are further subdivided into 5 convolution layers and 3 fully connected layers. Unlike LeNet, AlexNet has more filters to perform the convolution operation in each convolutional layer. In addition to the number of filters, the size of filters used in AlexNet was $11\times11, 5\times5$ and 3×3. The number of parameters present in the AlexNet is around 62 million. The training of AlexNet

was done in a parallel manner i.e. two Nvidia GPUs were used to train the network on the ImageNet dataset. AlexNet achieved 57% and 80.3% as its top-1 and top-5 accuracy respectively. Furthermore, the idea of Dropout was introduced to protect the model from overfitting. Consequently, a few million parameters were reduced from 60 million parameters of AlexNet due to the introduction of Dropout.

LET’S GET DEEPER:

Addition of layers to make the network deep.

After the AlexNet, the next champion of ImageNet (ILSVRC-2014) classification challenge was VGG-16. There are a lot of differences between AlexNet and VGG-16. Firstly, VGG-16 has more convolution layers which imply that deep learning researchers started focusing to increase the depth of the network. Secondly, VGG-16 only uses 3×3 kernels in every convolution layer to perform the convolution operation. Unlike AlexNet, the small kernels of VGG-16 can extract fine features present in images. The architecture of VGG-16 has an overall 5 blocks. The first two blocks of the network have 2 convolution layers and 1 max-pooling layer in each block. The remaining three blocks of the network have 3 convolution layers and 1 max-pooling layer. Thirdly, three fully connected layers are added after block 5 of the network: the first two layers have 4096 neurons and the third one has 1000 neurons to do the classification task in ImageNet. Therefore, the deep learning community also refers to VGG-16 as one the widest network ever built. Moreover, the number of parameters in the first two fully-connected layers of VGG-16 has

around a contribution of 100 million out of 138 million parameters of the network. The final layer is the Soft-max layer. The top-1 and top-5 accuracy of VGG-16 was 71.3% and 90.1% respectively.

GOOGLENET: NETWORK IN NETWORK:

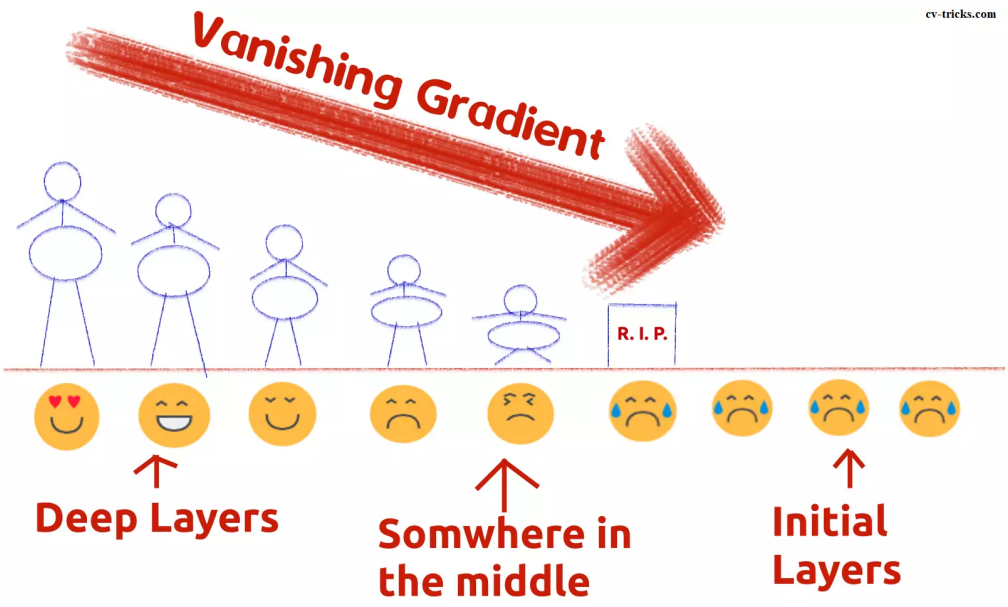
After the VGG-16 show, Google gave birth to the GoogleNet (Inception-V1): the other champion of ILSVRC-2014 with higher accuracy value than its predecessors. Unlike the prior networks, GoogleNet has a little strange architecture. Firstly, the networks such as VGG-16 have convolution layers stacked one over the other but GoogleNet arranges the convolution and pooling layers in a parallel manner to extract features using different kernel sizes. The overall intention was to increase the depth of the network and to gain a higher performance level as compared to previous winners of the ImageNet classification challenge. Secondly, the network uses 1×1 convolution operation to control the size of the volume passed for further processing in each inception module. The inception module is the collection of convolution and pooling operation performed in a parallel manner so that features can be extracted using different scales. Thirdly, the number of parameters present in the network is 24 million which makes GoogleNet a less compute-intensive model as compared to AlexNet and VGG-16. Fourthly, the network uses a Global Average Pooling layer in place of fully-connected layers. Ultimately, GoogleNet had achieved the lowest top-5 error of 6.67% in ILSVRC-2014.

STILL GRADIENTS

NOW, IT’S TIME TO TALK ABOUT THE MAIN SUBJECT OF THIS POST: THE WINNER OF ILSVRC 2015: THE DEEP RESIDUAL NETWORK

The winner of the ImageNet competition in 2015 was ResNet152 i.e. Residual Network having 152 layers variant. In this post, we will cover the concept of ResNet50 which can be generalized to any other variant of ResNet. Prior to the explanation of the deep residual network, I would like to talk about simple deep networks (networks having more number of convolution, pooling and activation layers stacked one over the other). Since 2013, the Deep Learning community started to build deeper networks because they were able to achieve high accuracy values. Furthermore, deeper networks can represent more complex features, therefore the model robustness and performance can be increased. However, stacking up more layers didn’t work for the researchers. While training deeper networks, the problem of accuracy degradation was

observed. In other words, adding more layers to the network either made the accuracy value to saturate or it abruptly started to decrease. The culprit for accuracy degradation was vanishing gradient effect which can only be observed in deeper networks.

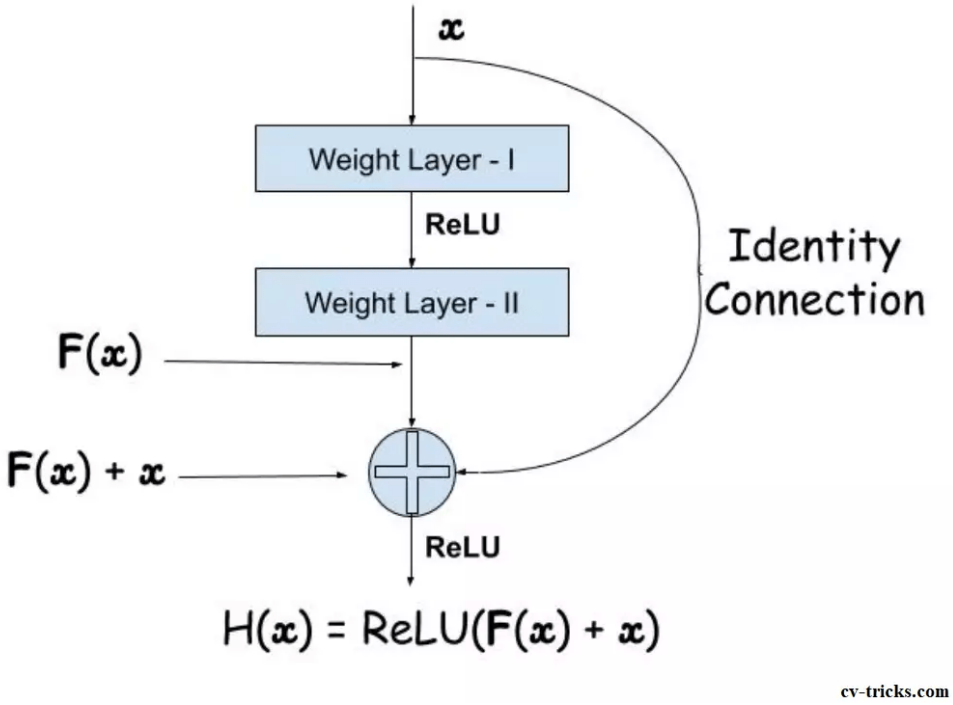


Vanishing Gradient Intuition

During the backpropagation stage, the error is calculated and gradient values are determined. The gradients are sent back to hidden layers and the weights are updated accordingly. The process of gradient determination and sending it back to the next hidden layer is continued until the input layer is reached. The gradient becomes smaller and smaller as it reaches the bottom of the network. Therefore, the weights of the initial layers will either update very slowly or remains the same. In other words, the initial layers of the network won't learn effectively. Hence, deep network training will not converge and accuracy will either starts to degrade or saturate at a particular value. Although vanishing gradient problem was addressed using the normalized initialization of weights, deeper network accuracy was still not increasing.

What is Deep Residual Network?

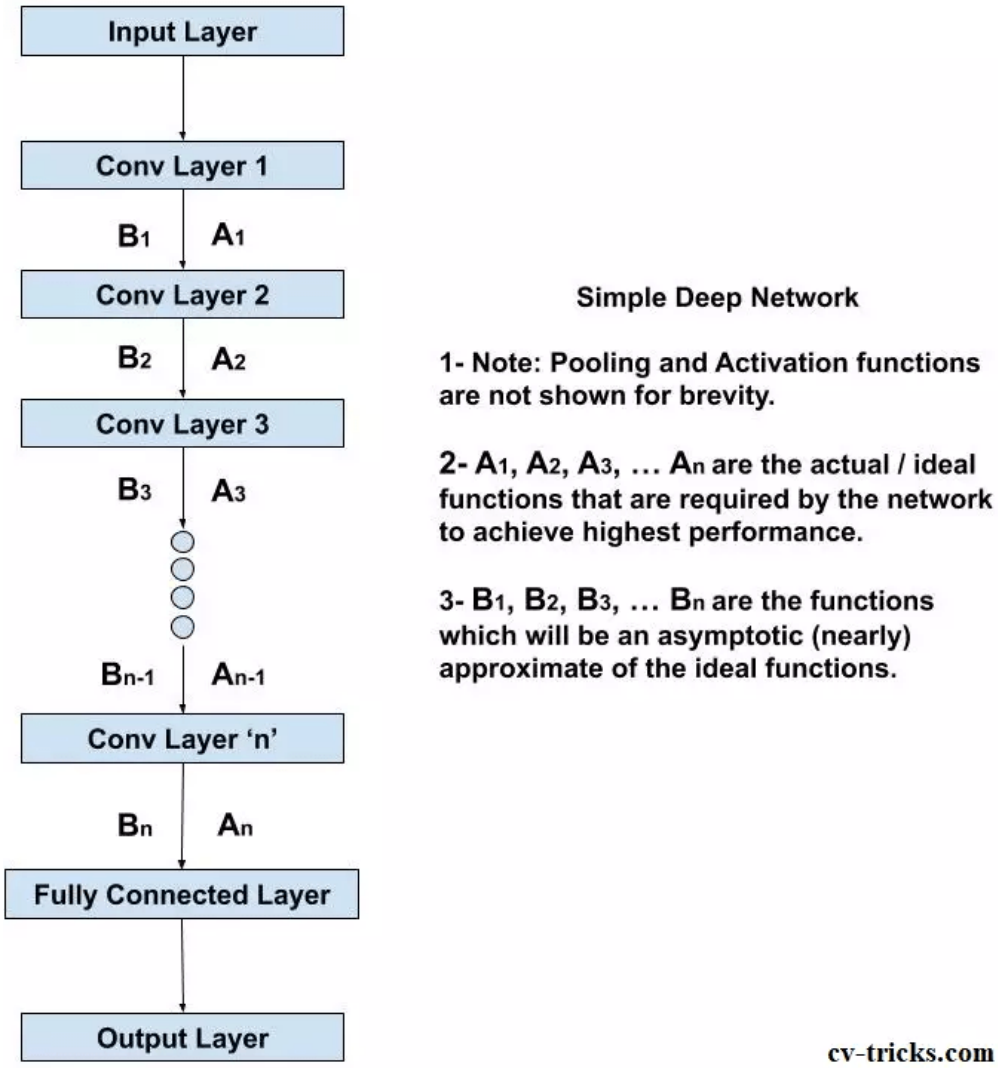
Deep Residual Network is almost similar to the networks which have convolution, pooling, activation and fully-connected layers stacked one over the other. The only construction to the simple network to make it a residual network is the identity connection between the layers. The screenshot below shows the residual block used in the network. You can see the identity connection as the curved arrow originating from the input and sinking to the end of the residual block.



A Residual Block of Deep Residual Network

What is the intuition behind the residual block?

As we have learned earlier that increasing the number of layers in the network abruptly degrades the accuracy. The deep learning community wanted a deeper network architecture that can either perform well or at least the same as the shallower networks. Now, try to imagine a deep network with convolution, pooling, etc layers stacked one over the other. Let us assume that, the actual function that we are trying to learn after every layer is given by $A_i(x)$ where A is the output function of the i-th layer for the given input x. You can refer to the next screenshot to understand the context. You can see that the output functions after every layer are $A_1, A_2, A_3, \ldots A_n$.



Assumed Deep Convolutional Neural Network

In this way of learning, the network is directly trying to learn these output functions i.e. without any extra support. Practically, it is not possible for the network to learn these ideal functions $(A_1, A_2, A_3, \ldots . A_n)$. The network can only learn the functions say $B_1, B_2, B_3, \ldots . B_n$ that are closer to $A_1, A_2, A_3, \ldots . A_n$. However, our assumed deep network is so much worse that even it can't learn $B_1, B_2, B_3, \ldots . B_n$ which will be closer to $A_1, A_2, A_3, \ldots . A_n$ because of the vanishing gradient effect and also due to the unsupported way of training.

The support for the training will be given by the identity mapping addition to the residual output. Firstly, let us see what is the meaning of identity mapping? In a nutshell, applying identity mapping to the input will give you the output which is the same as the input ($AI = A$: where A is input matrix and I is Identity Mapping).

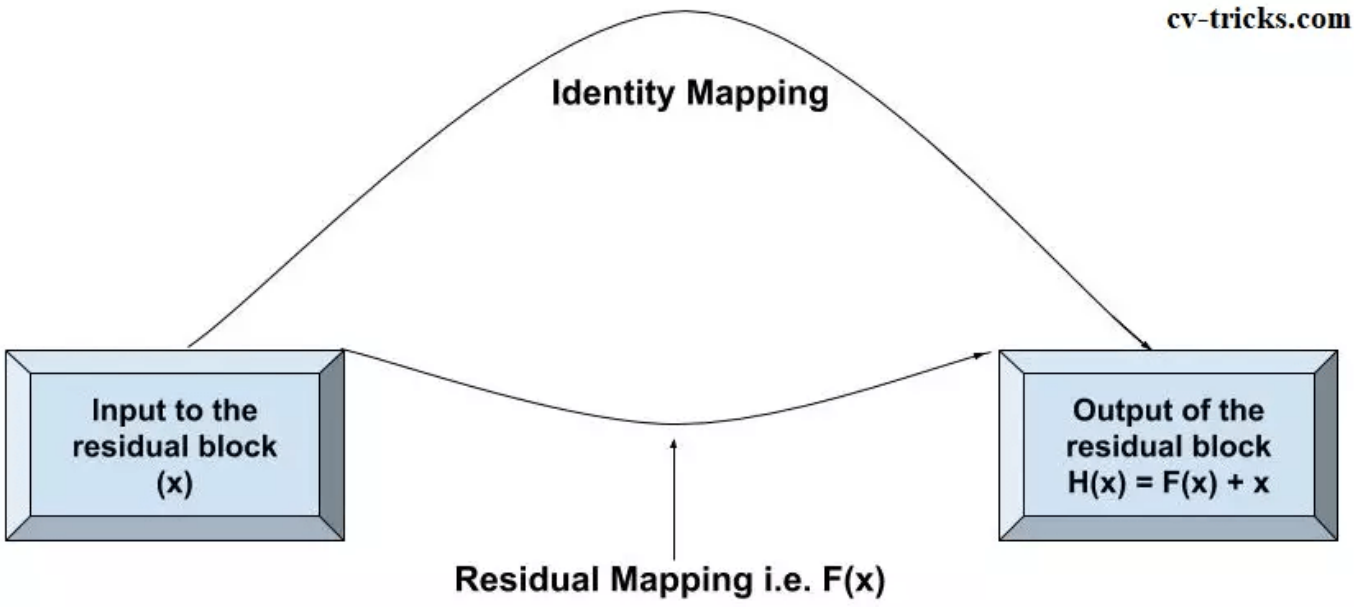
Traditional networks such as AlexNet, VGG-16, etc try to learn the $A_1, A_2, A_3, \ldots . A_n$ directly as shown in the diagram of the simple deep network. In the forward pass, the input (image) is passed to the network to get the output. The error is calculated and gradients are determined and backpropagation helps the network to approximate the functions $A_1, A_2, A_3, \ldots . A_n$ in the form of $B_1, B_2, B_3, \ldots . B_n$. The creators of ResNet thought that:

“IF THE MULTIPLE NON-LINEAR LAYERS CAN ASYMPTOTICALLY APPROXIMATE COMPLICATED FUNCTIONS, THEN IT IS EQUIVALENT TO HYPOTHESIZE THAT THEY CAN ASYMPTOTICALLY APPROXIMATE THE RESIDUAL FUNCTION”.

After reading the statement, the first question that strikes our mind is as follows:

What is a Residual Function?

The simple answer to this question is that the residual function (also known as residual mapping) is the difference between the input and output of the residual block under question. In other words, residual mapping is the value that will be added to the input to approximate the final function ($A_1, A_2, A_3, \ldots . A_n$) of the block. You can also assume that the residual mapping is the amount of error which can be added to input so as to reach the final destination i.e. to approximate the final function. You can visualize the Residual Mapping as shown in the next figure. You can see that the Residual Mapping is acting as a bridge between the input and the output of the block. Note that the weight layers and activation function are not shown in the diagram but they are actually present in the network.



Intuitive representation of Residual Function

Let us change our naming conventions to make this post compatible with ResNet paper. Fig-1, shows the residual block. The function which should be learned as a final result of the block is represented as $H(x)$. The input to the block is x and the residual mapping.

Why the Residual Function will work?

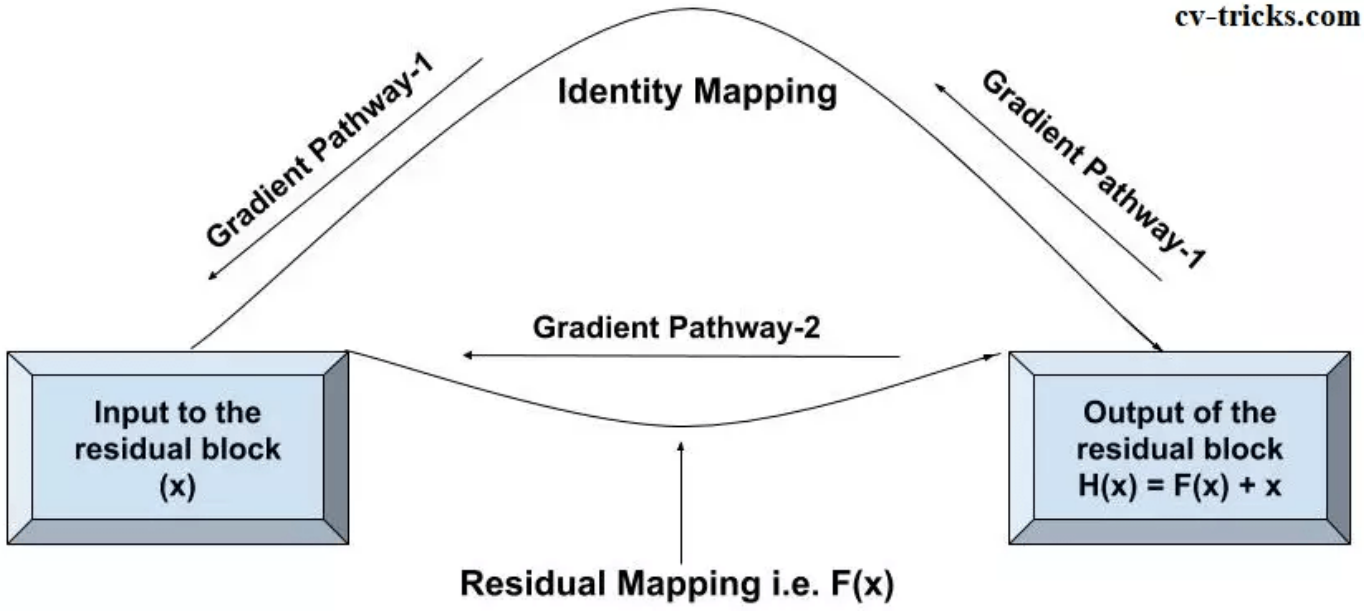
The creators of ResNet again thought as the statement goes:

" RATHER THAN EXPECTING STACKED LAYERS TO LEARN TO APPROXIMATE $H(x)$, THE AUTHORS ARE LETTING THE LAYERS TO APPROXIMATE A RESIDUAL FUNCTION i.e. $F(x) = H(x) - x$."

The above statement is explaining that during training the deep residual network, the main focus is to learn the residual function i.e. $F(x)$. So, if the network will somehow learn the difference $(F(x))$ between the input and output, then the overall accuracy can be increased. In other words, the residual value should be learned in a way such that it approaches zero, therefore making the identity mapping optimal. In this way, all the layers in the network will always produce the optimal feature maps i.e. the best case feature map after the convolution, pooling and activation operations. The optimal feature map contains all the pertinent features which can perfectly classify the image to its ground-truth class.

Why identity mapping will work? How does the identity connection affect the performance of the network?

During the time of backpropagation, there are two pathways for the gradients to transit back to the input layer while traversing a residual block. In the next diagram, you can see that there are two pathways: pathway-1 is the identity mapping way and pathway-2 is the residual mapping way.



Intuitive representation of Residual Function

We have already discussed the vanishing gradients problem in the simple deep network. Now we will try to explain the solution for the same keeping the identity connection in mind. Firstly, we will see how to represent the residual block $F(x)$ mathematically?

$$y = F(x, W_i) + x$$

where y is the output function, x is the input to the residual block and $F(x, W_i)$ is the residual block. Note that the residual block contains weight layers which are represented as W_i where $1 \leq i \leq \text{number of layers in a residual block}$. Also, the term $F(x, W_i)$ for 2 weight layers in a residual block can be simplified and can be written as follows:

$$F(x, W_i) = W_2\boldsymbol{\sigma}(W_1x)$$

where $\boldsymbol{\sigma}$ is the ReLU activation function and the second non-linearity is added after the addition with identity mapping i.e. $H(x) = \boldsymbol{\sigma}(y)$.

When the computed gradients pass from the **Gradient Pathway-2**, two weight layers are encountered which are W_1 and W_2 in our residual function $F(x)$. The weights or the kernels in the weight layers W_1 and W_2 are updated and new gradient values are calculated. In the case of initial layers, the newly computed values will either become small or eventually vanish. To save the gradient values from vanishing, the shortcut connection (identity mapping) will come into the picture. The gradients can directly pass through the **Gradient Pathway-1** shown in the previous diagram. In **Gradient Pathway-1**, the gradients don't have to encounter any weight layer, hence, there won't be any change in the value of computed gradients. The residual block will be skipped at once and the gradients can reach the initial layers which will help them to learn the correct weights. Also, ResNet version 1 has ReLU function after the addition operation, therefore, gradient values will be changed as soon as they are getting inside the residual block.

Basic properties and assumptions regarding the identity connection:

The addition of the identity connection does not introduce extra parameters. Therefore, the computation complexity for simple deep networks and deep residual networks is almost the same.

The dimensions of x and F must be the same for performing the addition operation. The dimensions can be matched by using one of the following ways:

- Extra Zero entries should be padded for increasing dimensions. This is not going to introduce any extra parameter.

- Projection shortcut can be used to match the dimensions (1×1 convolutions).

$$y = F(x, W_i) + W_s x$$

Key Features of ResNet:

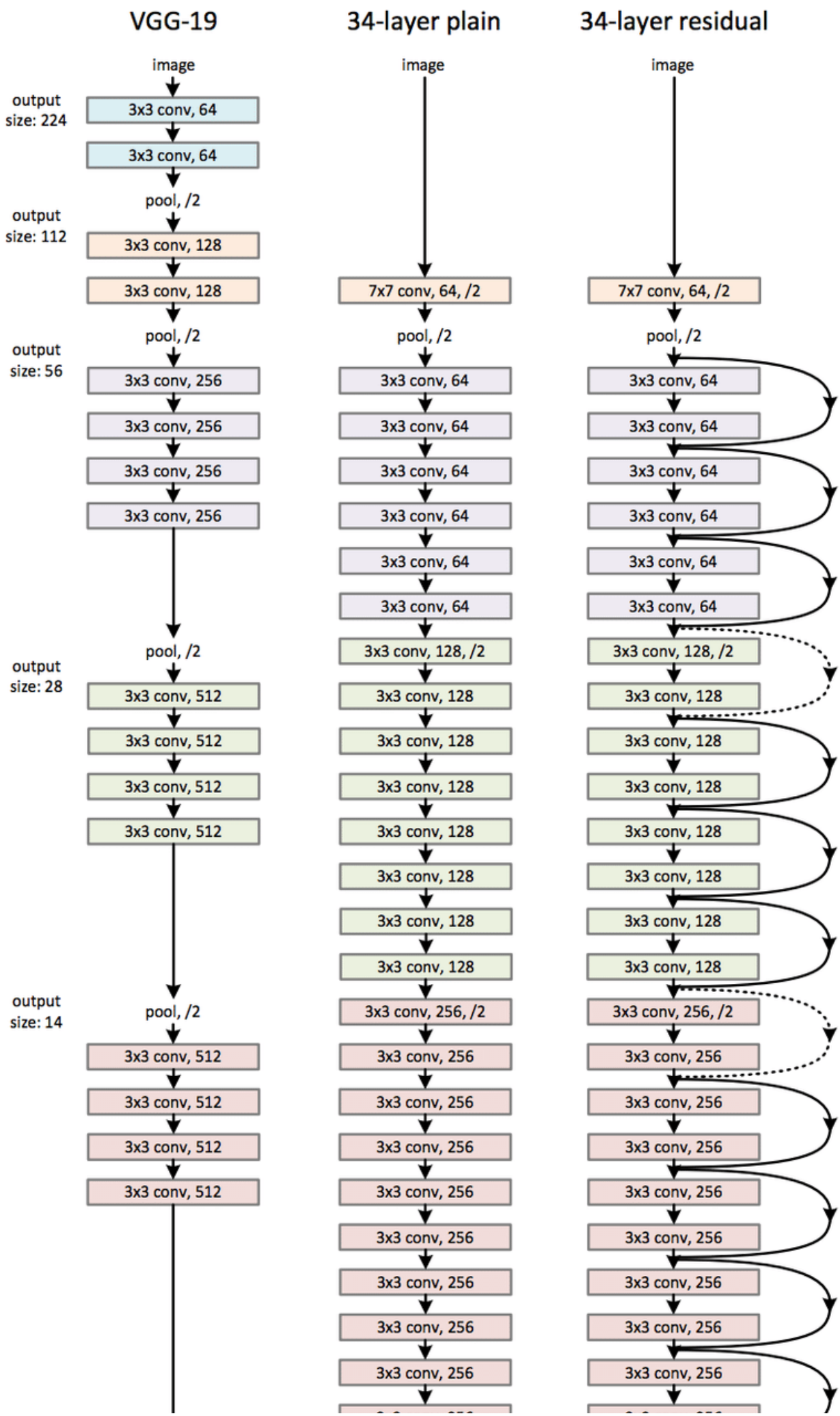
ResNet uses Batch Normalization at its core. The Batch Normalization adjusts the input layer to increase the performance of the network. The problem of covariate shift is mitigated. ResNet makes use of the Identity Connection, which helps to protect the network from vanishing gradient problem. Deep Residual Network uses bottleneck residual block design to increase the performance of the network.

Architecture of ResNet-34

The first ResNet architecture was the Resnet-34 (find the research paper here (https://arxiv.org/pdf/1512.03385.pdf)), which involved the insertion of shortcut connections in turning a plain network into its residual network counterpart. In this case, the plain network was inspired by VGG neural networks (VGG-16, VGG-19), with the convolutional networks having 3×3 filters. However, compared to VGGNets, ResNets have fewer filters and lower complexity. The 34-layer ResNet achieves a performance of 3.6 bn FLOPs, compared to 1.8bn FLOPs of smaller 18-layer ResNets.

It also followed two simple design rules – the layers had the same number of filters for the same output feature map size, and the number of filters doubled in case the feature map size was halved in order to preserve the time complexity per layer. It consisted of 34 weighted layers.

The shortcut connections were added to this plain network. While the input and output dimensions were the same, the identity shortcuts were directly used. With an increase in the dimensions, there were two options to be considered. The first was that the shortcut would still perform identity mapping while extra zero entries would be padded for increasing dimensions. The other option was to use the projection shortcut to match dimensions.


VGG-19
34-layer plain
34-layer residual
image
output size: 224
3x3 conv, 64
3x3 conv, 64
pool, /2
output size: 112
3x3 conv, 128
3x3 conv, 128
pool, /2
output size: 56
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool, /2
output size: 28
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 14
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256

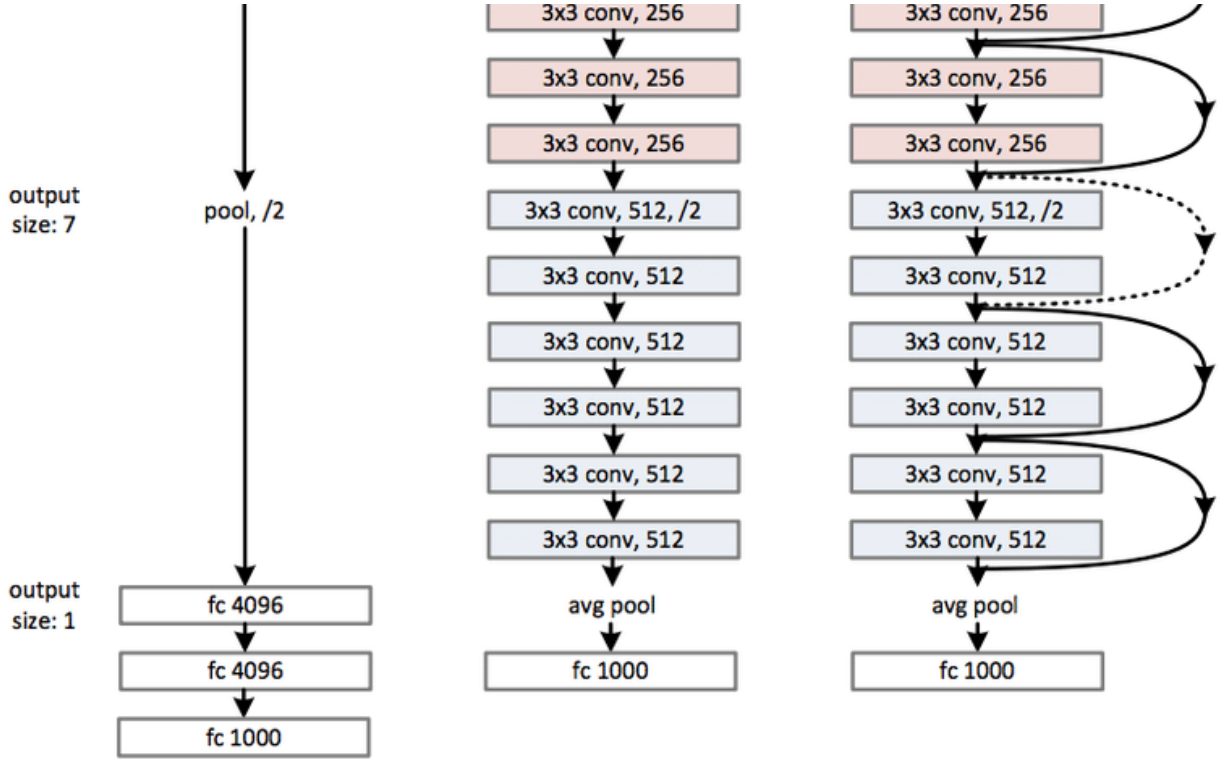


Figure 1. ResNet 34 from original paper

Architecture in detail

Since ResNets can have variable sizes, depending on how big each of the layers of the model are, and how many layers it has, we will follow the described by the authors in the paper — ResNet 34 — in order to explain the structure after these networks.

If you have taken a look at the paper, you will have probably seen some figures and tables like the following ones, that you are struggling to follow. Lets depict those figures by going into the detail of every step.

In here we can see that the ResNet (the one on the right) consists on one convolution and pooling step (on orange) followed by 4 layers of similar behavior.

Each of the layers follow the same pattern. They perform 3x3 convolution with a fixed feature map dimension (F) [64, 128, 256, 512] respectively, bypassing the input every 2 convolutions. Furthermore, the width (W) and height (H) dimensions remain constant during the entire layer.

The dotted line is there, precisely because there has been a change in the dimension of the input volume (of course a reduction because of the convolution). Note that this reduction between layers is achieved by an increase on the stride, from 1 to 2, at the first convolution of each layer; instead of by a pooling operation, which we are used to see as down samplers.

In the table, there is a summary of the output size at every layer and the dimension of the convolutional kernels at every point in the structure.

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				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 2. Sizes of outputs and convolutional kernels for ResNet 34

But this is not visible. We want images! An image is worth a thousand words!

The Figure 3 is the way I prefer to see convolutional models, and from which I will explain every layer. I prefer to observe how actually the volumes that are going through the model are changing their sizes. This way is easier to understand the mechanism of a particular model, to be able to adjust it to our particular needs — we will see how just changing the dataset forces to change the architecture of the entire model. Also, I will try to follow the notation close to the PyTorch official implementation to make it easier to later implement it on PyTorch.

For instance, ResNet on the paper is mainly explained for ImageNet dataset. But the first time I wanted to make an experiment with ensembles of ResNets, I had to do it on CIFAR10. Obviously, since CIFAR10 input images are (32x32) instead of (224x224), the structure of the ResNets need to be modify. If you want to have a control on the modifications to apply to your ResNet, you need to understand the details. This other tutorial is a simplified of the current one applied to CIFAR10. So, let’s go layer by layer!

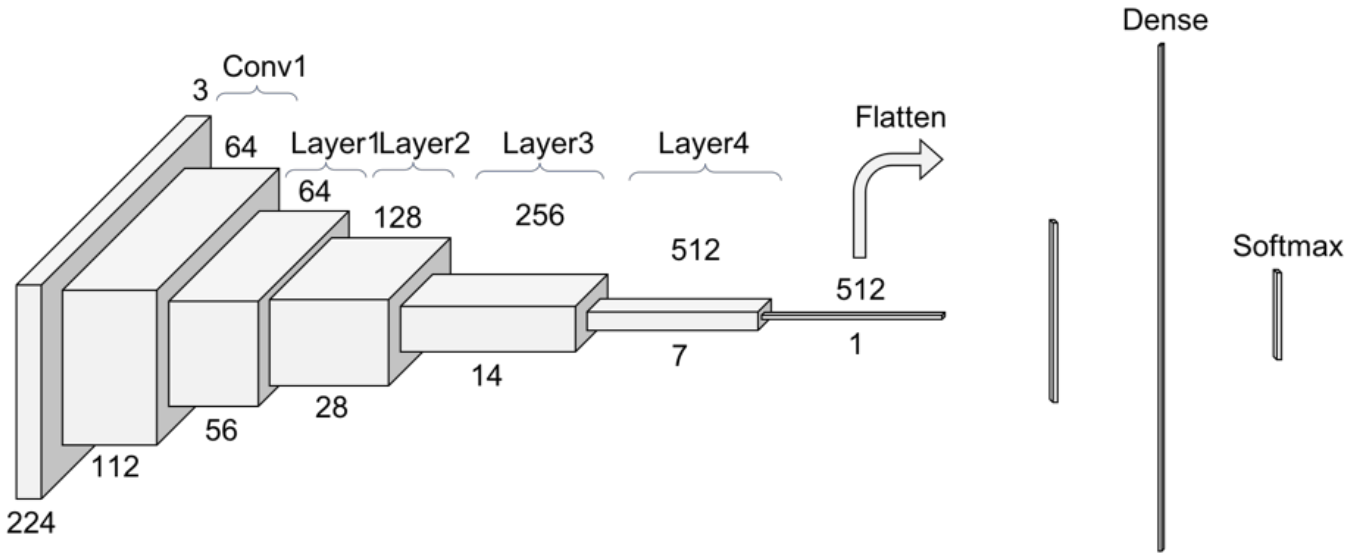


Figure 3. Another look at ResNet 34

Convolution 1

The first step on the ResNet before entering the common layer behavior is a block — called here Conv1 — consisting on a convolution + batch normalization + max pooling operation.

If you don’t remember how convolutions and pooling operations where performed, take a quick look at this draws I made to explain them, since I reused part of them here.

So, first there is a convolution operation. In the Figure 1 we can see that they use a kernel size of 7, and a feature map size of 64. You need to infer that they have padded with zeros 3 times on each dimension — and check it on the PyTorch documentation. Taken this into account, it can be seen in Figure 4 that the output size of that operation will be a (112x122) volume. Since each convolution filter (of the 64) is providing one channel in the output volume, we end up with a $(112 \times 112 \times 64)$ output volume — note this is free of the batch dimension to simplify the explanation.

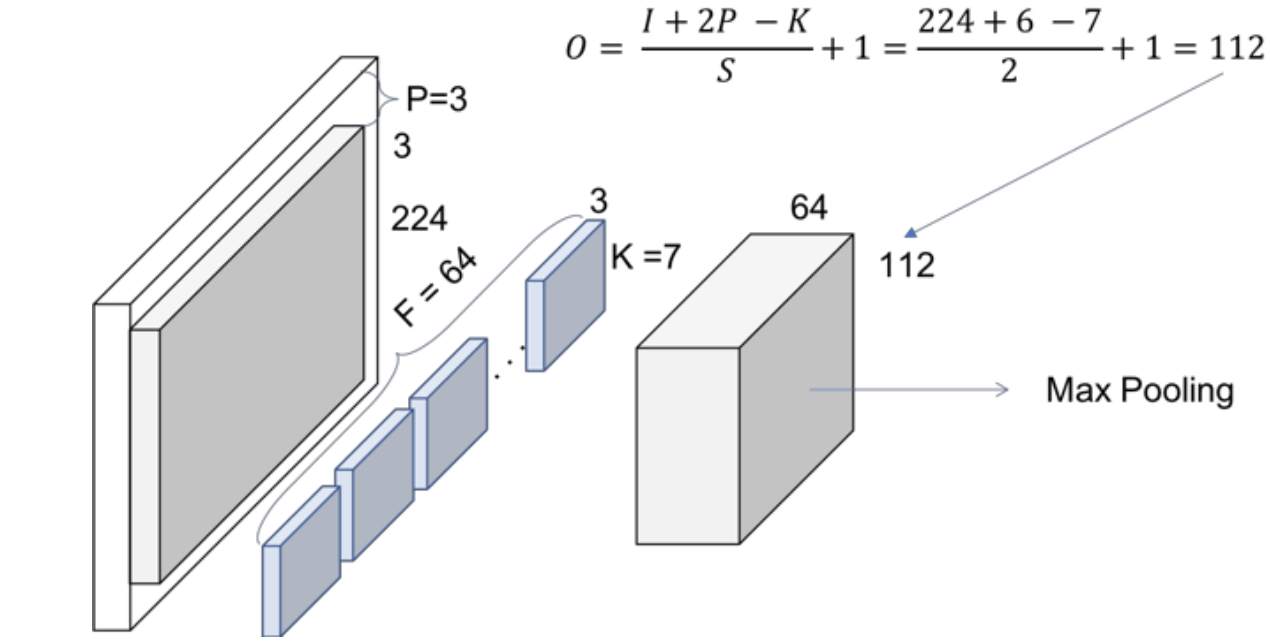


Figure 4. Conv1 — Convolution

The next step is the batch normalization, which is an element-wise operation and therefore, it does not change the size of our volume. Finally, we have the (3x3) Max Pooling operation with a stride of 2. We can also infer that they first pad the input volume, so the final volume has the desired dimensions.

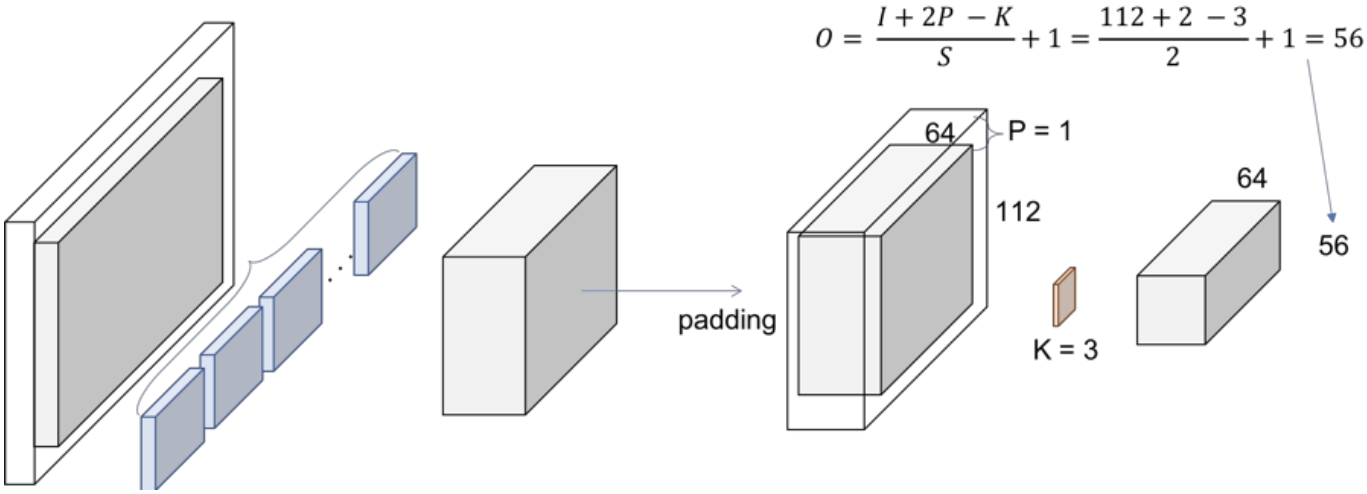


Figure 5. Conv1 — Max Pooling

ResNet Layers

So, let’s explain this repeating name, block. Every layer of a ResNet is composed of several blocks. This is because when ResNets go deeper, they normally do it by increasing the number of operations within a block, but the number of total layers remains the same — 4. An operation here refers to a convolution a batch normalization and a ReLU activation to an input, except the last operation of a block, that does not have the ReLU.

Therefore, in the PyTorch implementation they distinguish between the blocks that includes 2 operations — Basic Block — and the blocks that include 3 operations — Bottleneck Block. Note that normally each of these operations is called layer, but we are using layer already for a group of blocks.

We are facing a Basic one now. The input volume is the last output volume from Conv1. Let's see Figure 6 to figure out what is happening inside this block.

Block 1

1 convolution

We are replicating the simplified operation for every layer on the paper.

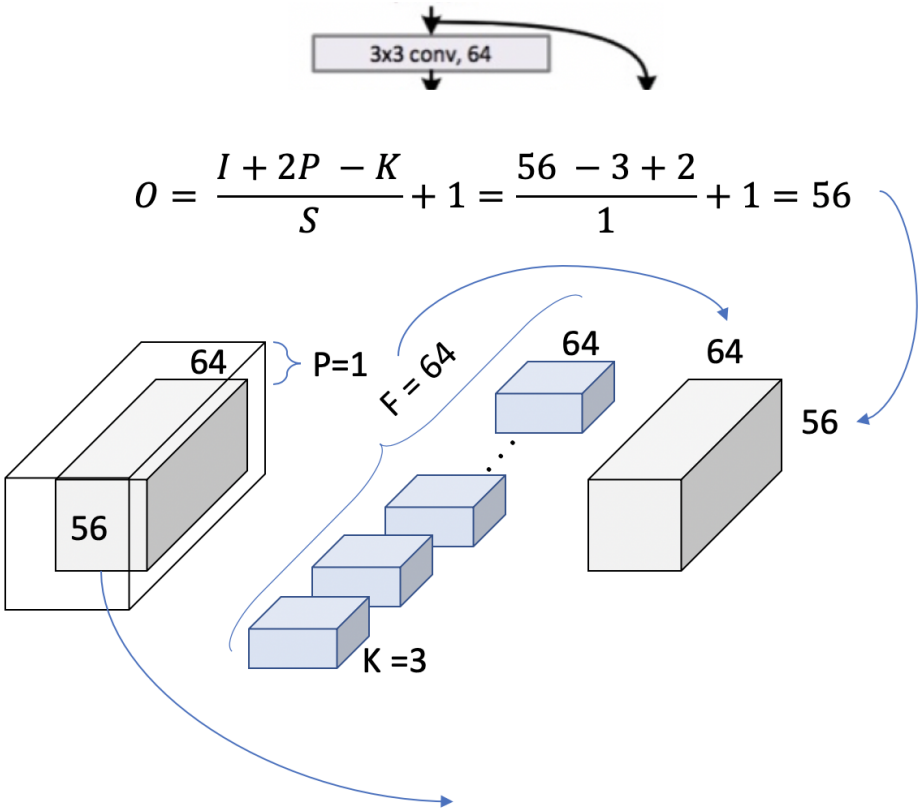


Figure 6. Layer 1, block 1, operation 1

We can double check now in the table from the paper we are using $[3 \times 3, 64]$ kernel and the output size is $[56 \times 56]$. We can see how, as we mentioned previously, the size of the volume does not change within a block. This is because a padding = 1 is used and a stride of also 1. Let's see how this extends to an entire block, to cover the 2 $[3 \times 3, 64]$ that appears in the table.

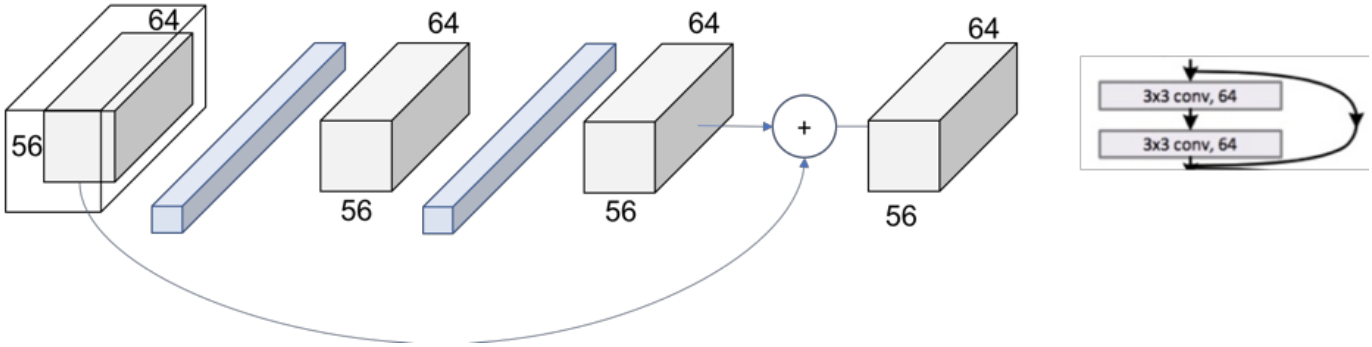


Figure 7. Layer 1, block 1

The same procedure can be expanded to the entire layer then as in Figure 8. Now, we can completely read the whole cell of the table (just recap we are in the 34 layers ResNet at Conv2_x layer.

We can see how we have the $[3 \times 3, 64] \times 3$ times within the layer.

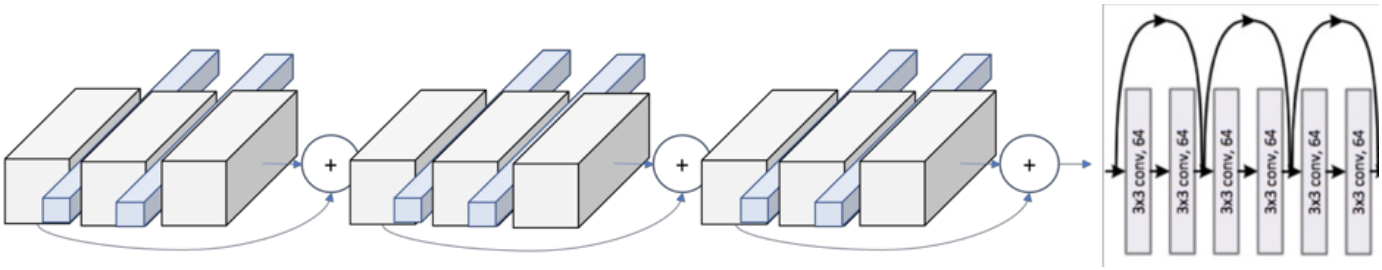


Figure 8. Layer 1

Patterns

The next step is to escalate from the entire block to the entire layer. In the Figure 1 we can see how the layers are differentiable by colors. However, if we look at the first operation of each layer, we see that the stride used at that first one is 2, instead of 1 like for the rest of them.

This means that the down sampling of the volume though the network is achieved by increasing the stride instead of a pooling operation like normally CNNs do. In fact, only one max pooling operation is performed in our Conv1 layer, and one average pooling layer at the end of the ResNet, right before the fully connected dense layer in Figure 1.

We can also see another repeating pattern over the layers of the ResNet, the dot layer representing the change of the dimensionality. This agrees with what we just said. The first operation of each layer is reducing the dimension, so we also need to resize the volume that goes through the skip connection, so we could add them like we did in Figure 7.

This difference on the skip connections are the so called in the paper as Identity Shortcut and Projection Shortcut. The identity shortcut is the one we have already discussed, simply bypassing the input volume to the addition operator. The projection shortcut performs a convolution operation to ensure the volumes at this addition operation are the same size. From the paper we can see that there are 2 options for matching the output size. Either padding the input volume or perform 1×1 convolutions. Here, this second option is shown.

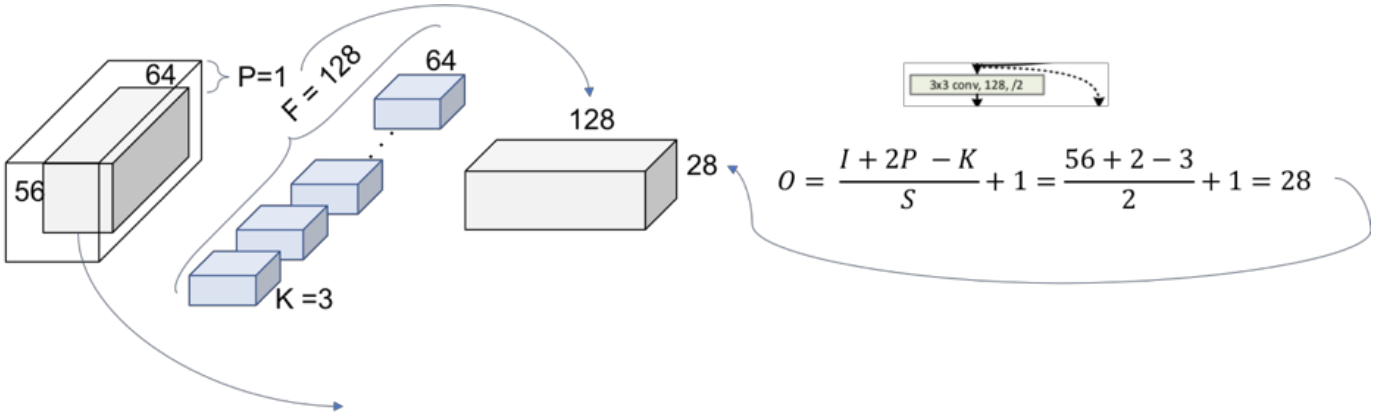


Figure 9. Layer2, Block 1, operation 1

Figure 9 represents this down sampling performed by increasing the stride to 2. The number of filters is duplicated in an attempt to preserve the time complexity for every operation $(56 \times 64 = 28 \times 128)$. Also, note that now the addition operation cannot be performed since the volume got modified. In the shortcut we need to apply one of our down sampling strategies. The 1×1 convolution approach is shown in Figure 10.

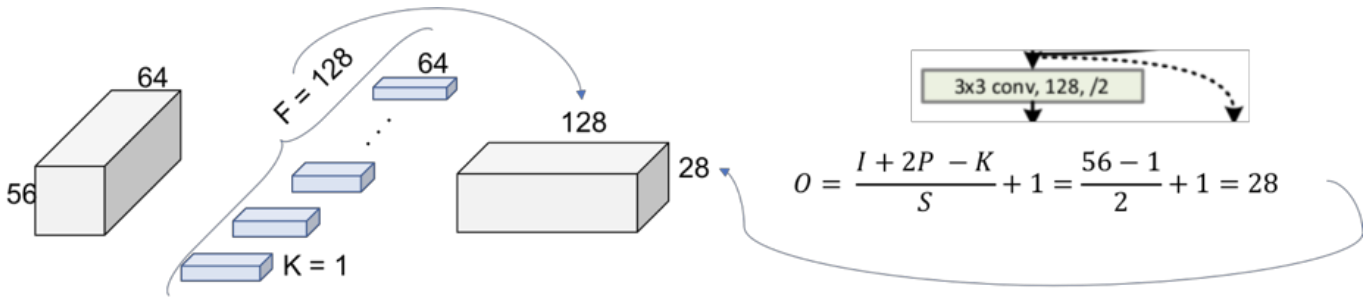


Figure 10. Projection Shortcut

The final picture looks then like in Figure 11 where now the 2 output volumes of each thread has the same size and can be added.

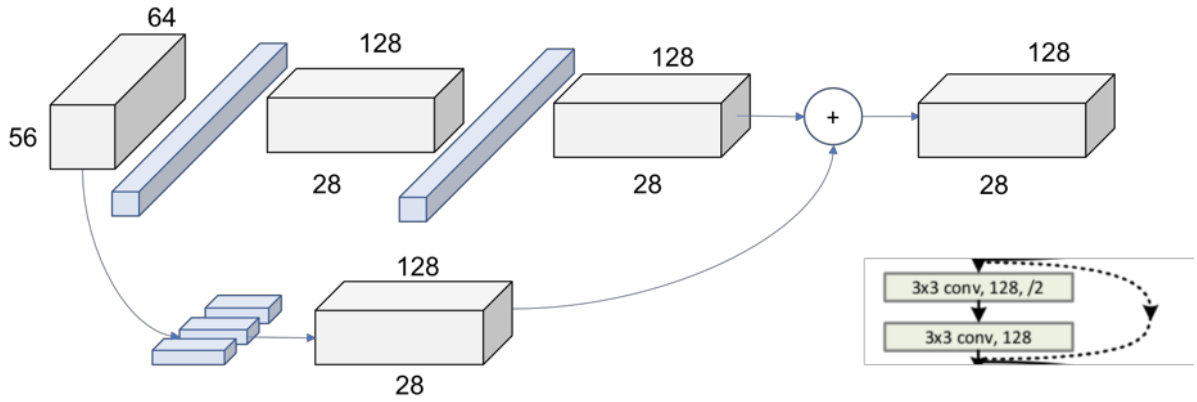


Figure 11. Layer 2, Block 1

In Figure 12 we can see the global picture of the entire second layer. The behavior is exactly the same for the following layers 3 and 4, changing only the dimensions of the incoming volumes.

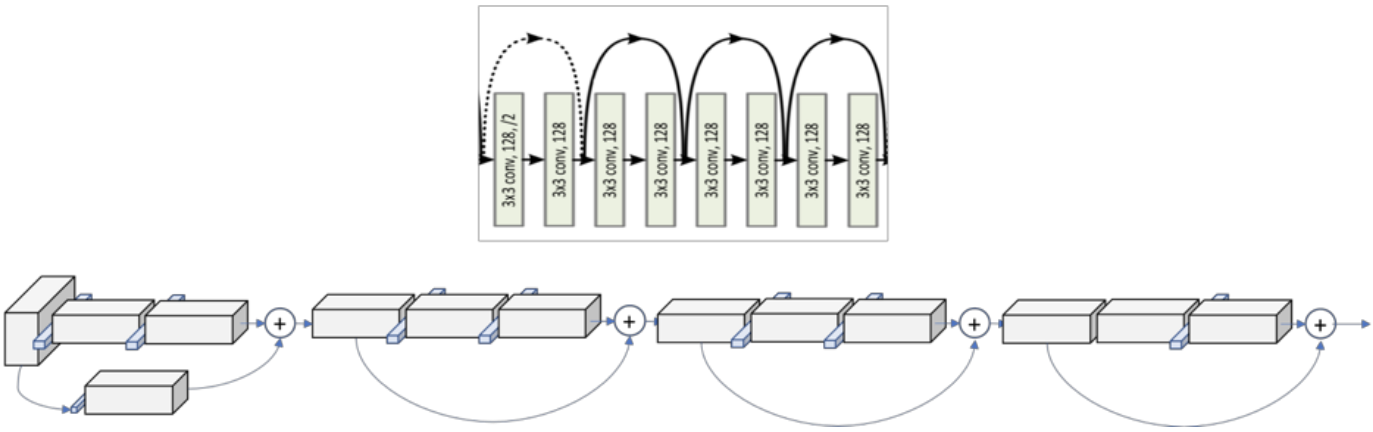


Figure 12.Layer 2

Architecture of ResNet-50

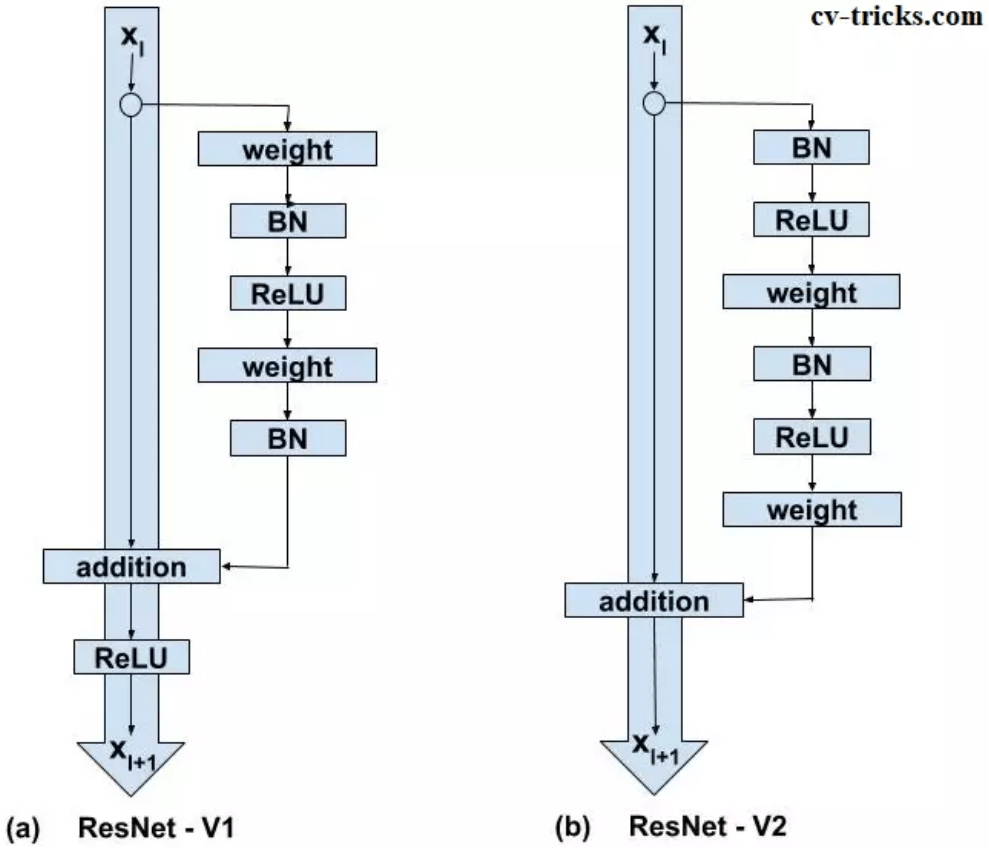
Now we'll talk about the architecture of ResNet50. The architecture of ResNet50 has 4 stages as shown in the diagram below. The network can take the input image having height, width as multiples of 32 and 3 as channel width. For the sake of explanation, we will consider the input size as $224 \times 224 \times 3$. Every ResNet architecture performs the initial convolution and max-pooling using 7×7 and 3×3 kernel sizes respectively. Afterward, Stage 1 of the network starts and it has 3 Residual blocks containing 3 layers each. The size of kernels used to perform the convolution operation in all 3 layers of the block of stage 1 are 64, 64 and 128 respectively. The curved arrows refer to the identity connection. The dashed connected arrow represents that the convolution operation in the Residual Block is performed with stride 2, hence, the size of input will be 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.

For deeper networks like ResNet50, ResNet152, etc, bottleneck design is used. For each residual function F, 3 layers are stacked one over the other. The three layers are $1\times1, 3\times3, 1\times1$ convolutions. The 1×1 convolution layers are responsible for reducing and then restoring the dimensions. The 3×3 layer is left as a bottleneck with smaller input/output dimensions.

Finally, the network has an Average Pooling layer followed by a fully connected layer having 1000 neurons (ImageNet class output).

ResNet – V2

Till now we have discussed the ResNet50 version 1. Now, we will discuss the ResNet50 version 2 which is all about using the pre-activation of weight layers instead of post-activation. The figure below shows the basic architecture of the post-activation (original version 1) and the pre-activation (version 2) of versions of ResNet.



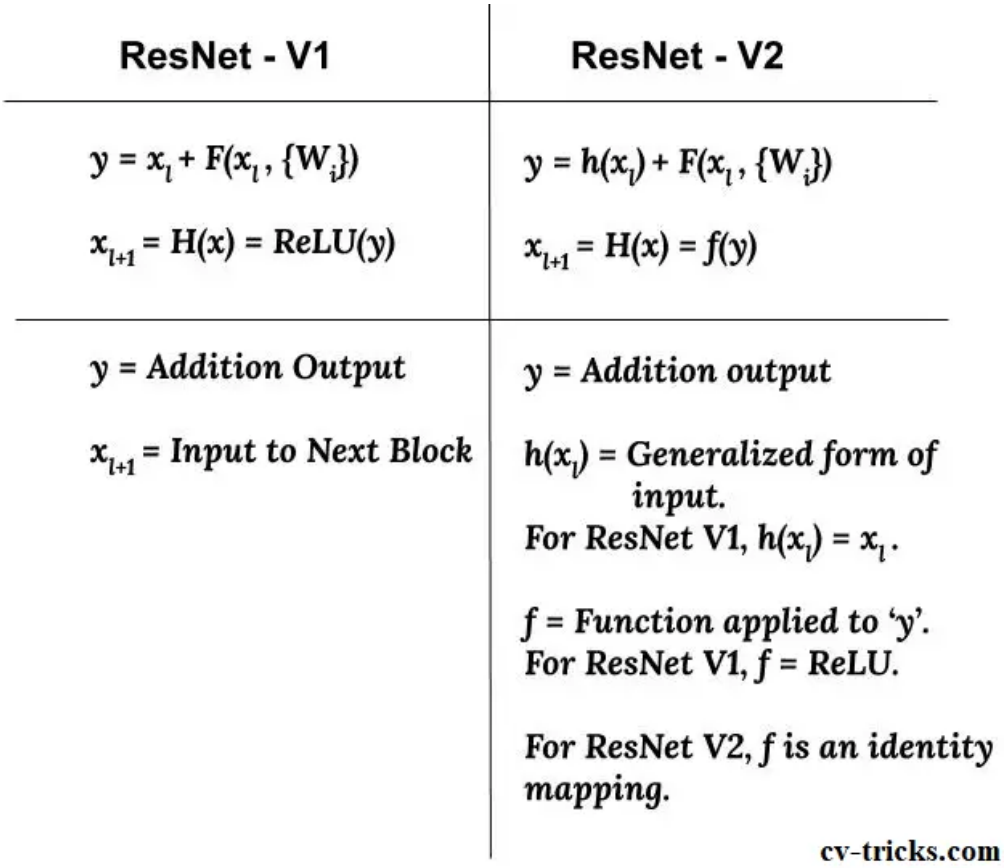
ResNet V1 and ResNet V2

The major differences between ResNet – V1 and ResNet – V2 are as follows:

- ResNet V1 adds the second non-linearity after the addition operation is performed in between the x and $F(x)$. ResNet V2 has removed the last non-linearity, therefore, clearing

the path of the input to output in the form of identity connection.

- ResNet V2 applies Batch Normalization and ReLU activation to the input before the multiplication with the weight matrix (convolution operation). ResNet V1 performs the convolution followed by Batch Normalization and ReLU activation.



Difference between ResNet V1 and ResNet V2

The ResNet V2 mainly focuses on making the second non-linearity as an identity mapping i.e. the output of addition operation between the identity mapping and the residual mapping should be passed as it is to the next block for further processing. However, the output of the addition operation in ResNet V1 passes from ReLU activation and then transferred to the next block as the input.

When the function 'f' is an identity function the signal can be directly propagated between any two units. Also, the gradient value calculated at the output layer can easily reach the initial layer without any change in signal.

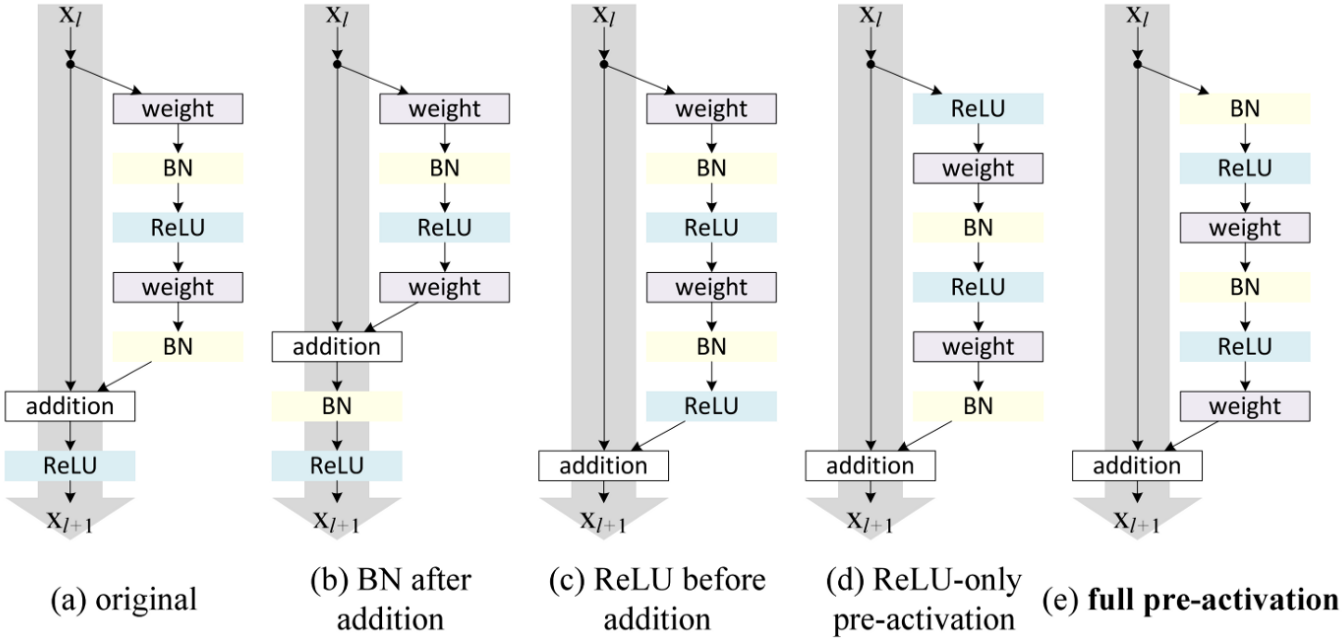
Variants of residual blocks

As a matter of fact, ResNet was not the first to make use of shortcut connections, Highway Network introduced gated shortcut connections. These parameterized gates control how much information is allowed to flow across the shortcut. Similar idea can be found in the Long Term

Short Memory (LSTM) cell, in which there is a parameterized forget gate that controls how much information will flow to the next time step. Therefore, ResNet can be thought of as a special case of Highway Network.

However, experiments show that Highway Network performs no better than ResNet, which is kind of strange because the solution space of Highway Network contains ResNet, therefore it should perform at least as good as ResNet. This suggests that it is more important to keep these “gradient highways” clear than to go for larger solution space.

Following this intuition, the authors refined the residual block and proposed a pre-activation variant of residual block, in which the gradients can flow through the shortcut connections to any other earlier layer unimpededly. In fact, using the original residual block, training a 1202-layer ResNet resulted in worse performance than its 110-layer counterpart.



Variants of residual blocks

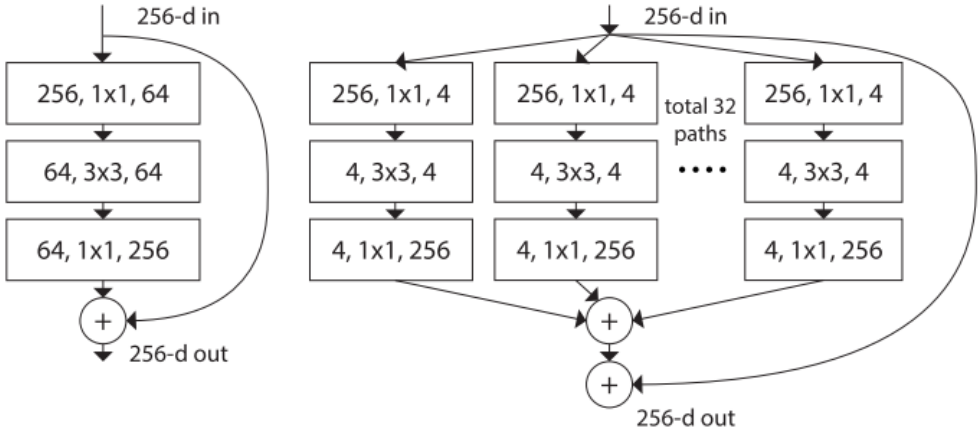
The authors demonstrated with experiments that they can now train a 1001-layer deep ResNet to outperform its shallower counterparts. Because of its compelling results, ResNet quickly became one of the most popular architectures in various computer vision tasks.

Recent Variants and Interpretations of ResNet

As ResNet gains more and more popularity in the research community, its architecture is getting studied heavily. In this section, I will first introduce several new architectures based on ResNet, then introduce a paper that provides an interpretation of treating ResNet as an ensemble of many smaller networks.

ResNeXt

Xie et al. proposed a variant of ResNet that is codenamed ResNeXt with the following building block:

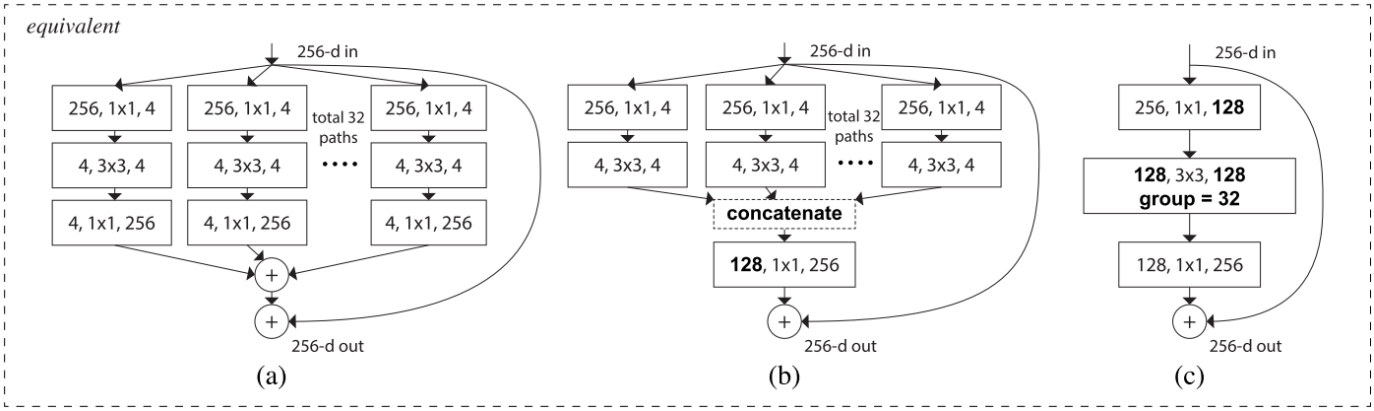


left: a building block of resnet, right: a building block of ResNeXt with cardinality = 32

This may look familiar to you as it is very similar to the Inception module, they both follow the split-transform-merge paradigm, except in this variant, the outputs of different paths are merged by adding them together, while in Inception they are depth-concatenated. Another difference is that in Inception, each path is different ($1 \times 1, 3 \times 3$ and 5×5 convolution) from each other, while in this architecture, all paths share the same topology.

The authors introduced a hyper-parameter called cardinality — the number of independent paths, to provide a new way of adjusting the model capacity. Experiments show that accuracy can be gained more efficiently by increasing the cardinality than by going deeper or wider. The authors state that compared to Inception, this novel architecture is easier to adapt to new datasets/tasks, as it has a simple paradigm and only one hyper-parameter to be adjusted, while Inception has many hyper-parameters (like the kernel size of the convolutional layer of each path) to tune.

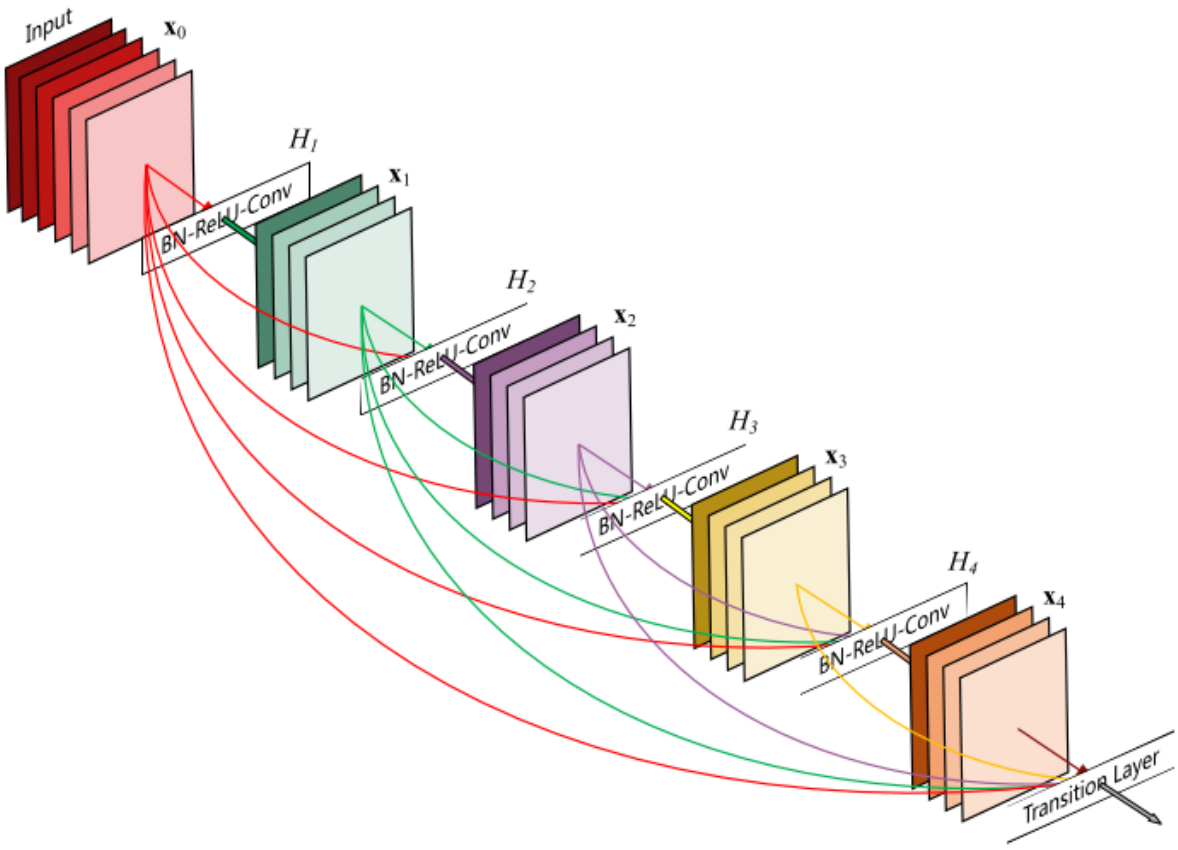
This novel building block has three equivalent form as follows:



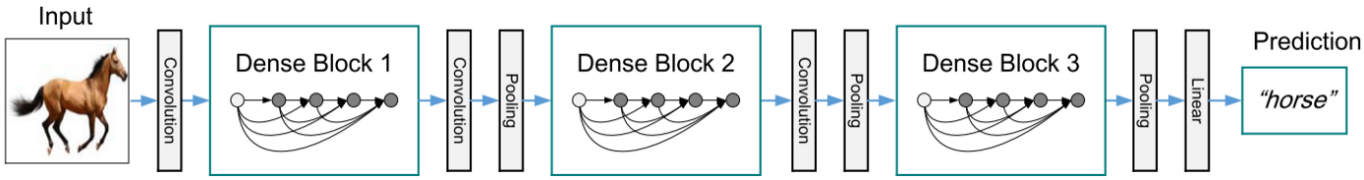
In practice, the “split-transform-merge” is usually done by pointwise grouped convolutional layer, which divides its input into groups of feature maps and perform normal convolution respectively, their outputs are depth-concatenated and then fed to a 1×1 convolutional layer.

Densely Connected CNN

Huang et al. proposed a novel architecture called DenseNet that further exploits the effects of shortcut connections — it connects all layers directly with each other. In this novel architecture, the input of each layer consists of the feature maps of all earlier layer, and its output is passed to each subsequent layer. The feature maps are aggregated with depth-concatenation.



Other than tackling the vanishing gradients problem, S. Xie et al. argue that this architecture also encourages feature reuse, making the network highly parameter-efficient. One simple interpretation of this is that, the output of the identity mapping was added to the next block, which might impede information flow if the feature maps of two layers have very different distributions. Therefore, concatenating feature maps can preserve them all and increase the variance of the outputs, encouraging feature reuse.



Following this paradigm, we know that the $l-th$ layer will have $k*(l-1)+k_0$ input feature maps, where k_0 is the number of channels in the input image. The authors used a hyper-parameter called growth rate (k) to prevent the network from growing too wide, they also used a 1×1 convolutional bottleneck layer to reduce the number of feature maps before the expensive 3×3 convolution. The overall archiecture is shown in the below table:

Layers	Output Size	DenseNet-121($k=32$)	DenseNet-169($k=32$)	DenseNet-201($k=32$)	DenseNet-161($k=48$)
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix}1\times 1\text{ conv}\\3\times 3\text{ conv}\end{bmatrix}\times 6$	$\begin{bmatrix}1\times 1\text{ conv}\\3\times 3\text{ conv}\end{bmatrix}\times 6$	$\begin{bmatrix}1\times 1\text{ conv}\\3\times 3\text{ conv}\end{bmatrix}\times 6$	$\begin{bmatrix}1\times 1\text{ conv}\\3\times 3\text{ conv}\end{bmatrix}\times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix}1\times 1\text{ conv}\\3\times 3\text{ conv}\end{bmatrix}\times 12$	$\begin{bmatrix}1\times 1\text{ conv}\\3\times 3\text{ conv}\end{bmatrix}\times 12$	$\begin{bmatrix}1\times 1\text{ conv}\\3\times 3\text{ conv}\end{bmatrix}\times 12$	$\begin{bmatrix}1\times 1\text{ conv}\\3\times 3\text{ conv}\end{bmatrix}\times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix}1\times 1\text{ conv}\\3\times 3\text{ conv}\end{bmatrix}\times 24$	$\begin{bmatrix}1\times 1\text{ conv}\\3\times 3\text{ conv}\end{bmatrix}\times 32$	$\begin{bmatrix}1\times 1\text{ conv}\\3\times 3\text{ conv}\end{bmatrix}\times 48$	$\begin{bmatrix}1\times 1\text{ conv}\\3\times 3\text{ conv}\end{bmatrix}\times 36$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix}1\times 1\text{ conv}\\3\times 3\text{ conv}\end{bmatrix}\times 16$	$\begin{bmatrix}1\times 1\text{ conv}\\3\times 3\text{ conv}\end{bmatrix}\times 32$	$\begin{bmatrix}1\times 1\text{ conv}\\3\times 3\text{ conv}\end{bmatrix}\times 32$	$\begin{bmatrix}1\times 1\text{ conv}\\3\times 3\text{ conv}\end{bmatrix}\times 24$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

DenseNet architectures for ImageNet

Deep Network with Stochastic Depth

Although ResNet has proven powerful in many applications, one major drawback is that deeper network usually requires weeks for training, making it practically infeasible in real-world applications. To tackle this issue, Huang et al. introduced a counter-intuitive method of randomly dropping layers during training, and using the full network in testing.

The authors used the residual block as their network's building block, therefore, during training, when a particular residual block is enable, its input flows through both the identity shortcut and the weight layers, otherwise the input only flows only through the identity shortcut. In training time, each layer has a "survival probability" and is randomly dropped. In testing time, all blocks are kept active and re-calibrated according to its survival probability during training.

Formally, let H_l be the output of the $l-th$ residual block, f_l be the mapping defined by the $l-th$ block's weighted mapping, b_l be a Bernoulli random variable that be only 1 or 0 (indicating whether a block is active), during training:

$$H_l = ReLU(b_l \times f_l(H_{l-1}) + id(H_{l-1}))$$

When $b_l = 1$, this block becomes a normal residual block, and when $b_l = 0$, the above formula becomes:

$$H_l = ReLU(id(H_{l-1}))$$

Since we know that H_{l-1} is the output of a ReLU, which is already non-negative, the above equation reduces to a identity layer that only passes the input through to the next layer:

$$H_l = id(H_{l-1})$$

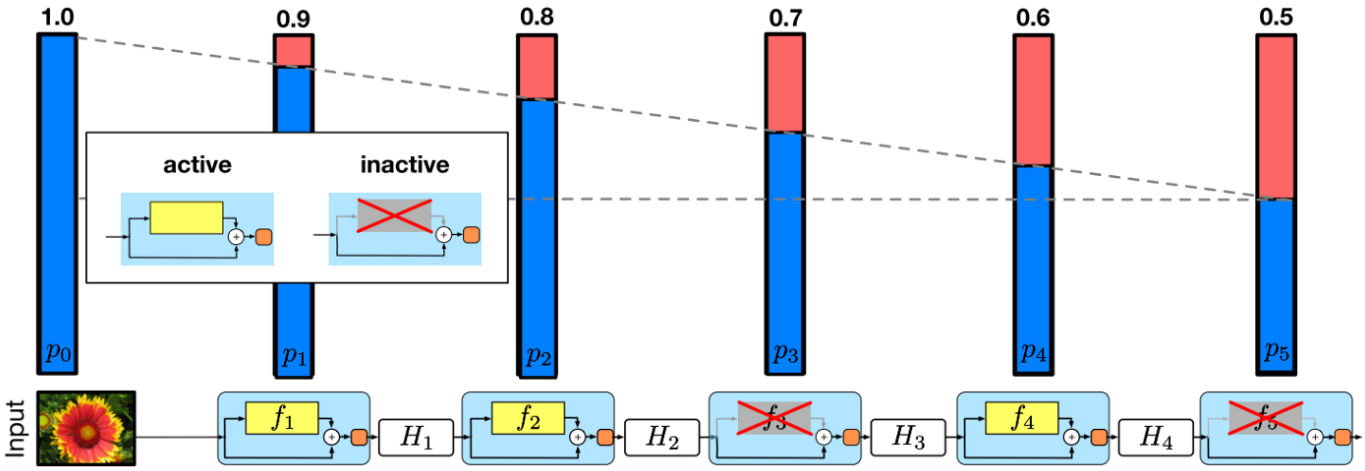
Let p_l be the survival probability of layer l during training, during test time, we have:

$$H_l = ReLU(p_l \times f_l(H_{l-1}) + id(H_{l-1}))$$

The authors applied a linear decay rule to the survival probability of each layer, they argue that since earlier layers extract low-level features that will be used by later ones, they should not be dropped too frequently, the resulting rule therefore becomes:

$$p_l = 1 - \frac{l}{L}(1 - p_L)$$

Where L denotes the total number of blocks, thus p_L is the survival probability of the last residual block and is fixed to 0.5 throughout experiments. Also note that in this setting, the input is treated as the first layer $(l = 0)$ and thus never dropped. The overall framework over stochastic depth training is demonstrated in the figure below.



During training, each layer has a probability of being disabled

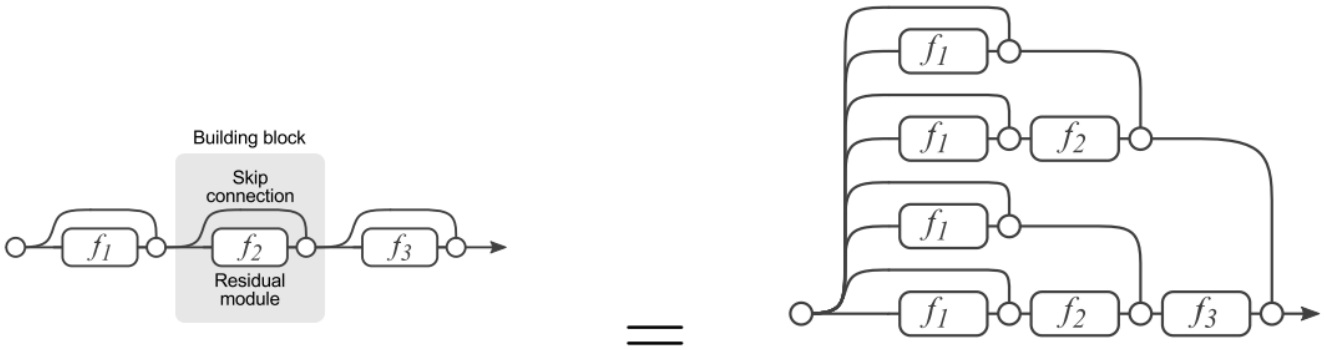
Similar to Dropout, training a deep network with stochastic depth can be viewed as training an ensemble of many smaller ResNets. The difference is that this method randomly drops an entire layer while Dropout only drops part of the hidden units in one layer during training.

Experiments show that training a 110-layer ResNet with stochastic depth results in better performance than training a constant-depth 110-layer ResNet, while reduces the training time dramatically. This suggests that some of the layers (paths) in ResNet might be redundant.

ResNet as an Ensemble of Smaller Networks

G. Huang et al. proposed a counter-intuitive way of training a very deep network by randomly dropping its layers during training and using the full network in testing time. Veit et al. had an even more counter-intuitive finding: we can actually drop some of the layers of a trained ResNet and still have comparable performance. This makes the ResNet architecture even more interesting, also dropped layers of a VGG network and degraded its performance dramatically.

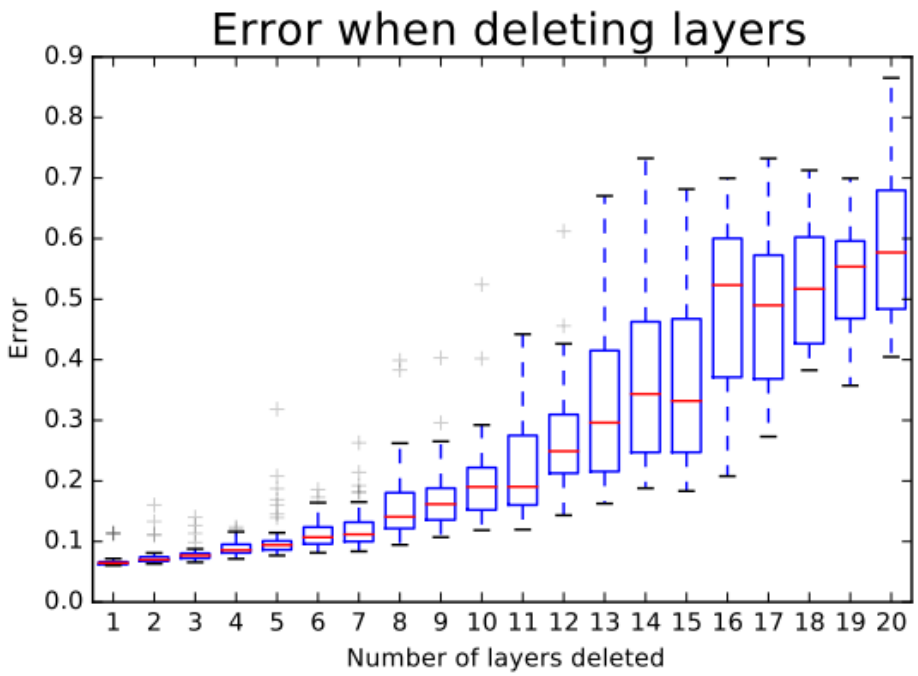
Veit et al. first provides an unraveled view of ResNet to make things clearer. After we unroll the network architecture, it is quite clear that a ResNet architecture with i residual blocks has 2 ** i different paths (because each residual block provides two independent paths).



(a) Conventional 3-block residual network (b) Unraveled view of (a)

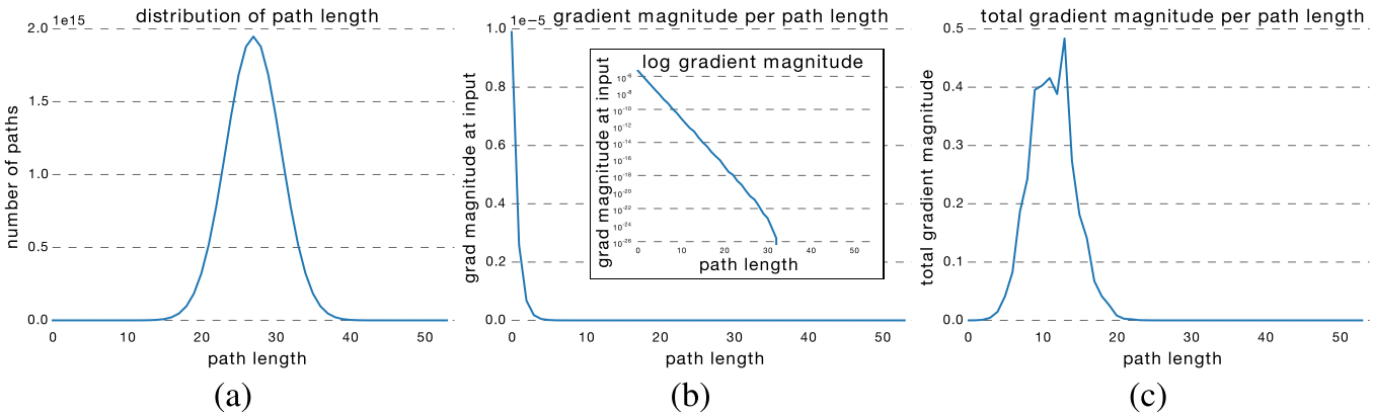
Given the above finding, it is quite clear why removing a couple of layers in a ResNet architecture doesn't compromise its performance too much — the architecture has many independent effective paths and the majority of them remain intact after we remove a couple of layers. On the contrary, the VGG network has only one effective path, so removing a single layer compromises this one the only path.

The authors also conducted experiments to show that the collection of paths in ResNet have ensemble-like behaviour. They do so by deleting different number of layers at test time, and see if the performance of the network smoothly correlates with the number of deleted layers. The results suggest that the network indeed behaves like ensemble, as shown in the below figure:



error increases smoothly as the the number of deleted layers increases

Finally the authors looked into the characteristics of the paths in ResNet: It is apparent that the distribution of all possible path lengths follows a Binomial distribution, as shown in (a) of the blow figure. The majority of paths go through 19 to 35 residual blocks.



To investigate the relationship between path length and the magnitude of the gradients flowing through it. To get the magnitude of gradients in the path of length k, the authors first fed a batch of data to the network, and randomly sample k residual blocks. When back propagating the gradients, they propagated through the weight layer only for the sampled residual blocks. (b) shows that the magnitude of gradients decreases rapidly as the path becomes longer.

We can now multiply the frequency of each path length with its expected magnitude of gradients to have a feel of how much paths of each length contribute to training, as in (c). Surprisingly, most contributions come from paths of length 9 to 18, but they constitute only a tiny portion of the

total paths, as in (a). This is a very interesting finding, as it suggests that ResNet did not solve the vanishing gradients problem for very long paths, and that ResNet actually enables training very deep network by shortening its effective paths.

Last edited by : May 18, 2023, 9:59 a.m.

- **Prev** : K_04 Understanding of MobileNet - EN
- **Next** : K_06 Understanding of XCeption - EN

TOP