Understanding and visualizing ResNets

Pablo Ruiz Oct 8, 2018 · 9 min read

This post can be downloaded in PDF here.

It is part of a series of tutorials on CNN architectures.

The main purpose is to give insight to understand ResNets and go deep into ResNet34 for ImageNet dataset.

- For ResNets applied to CIFAR10, there is another tutorial here.
- There is also a PyTorch implementation detailed tutorial here.

Index

comes to convolutional neural networks. This makes sense, since the models should be more capable (their flexibility to adapt to any space increase because they have a bigger parameter space to explore). However, it has been noticed that after some depth, the performance degrades.

This was one of the bottlenecks of VGG. They couldn't go as deep as wanted, because they started to lose generalization capability.

Motivation

Since neural networks are good function approximators, they should be able to easily solve the identify function, where the output of a function becomes the input itself.

$$f(x) = x$$

Following the same logic, if we bypass the input to the first layer of the model to be the output of the last layer of the model, the network should be able to predict whatever function it was learning before with the input added to it.

$$f(x) + x = h(x)$$

The intuition is that learning f(x) = 0 has to be easy for the network.

What problems ResNets solve?

One of the problems ResNets solve is the famous known **vanishing gradient**. This is because when the network is too deep, the gradients from where the loss function is calculated easily shrink to zero after several applications of the chain rule. This result on the weights never updating its values and therefore, no learning is being performed.

With ResNets, the **gradients can flow directly through the skip connections backwards from later layers to initial filters**.

Architecture

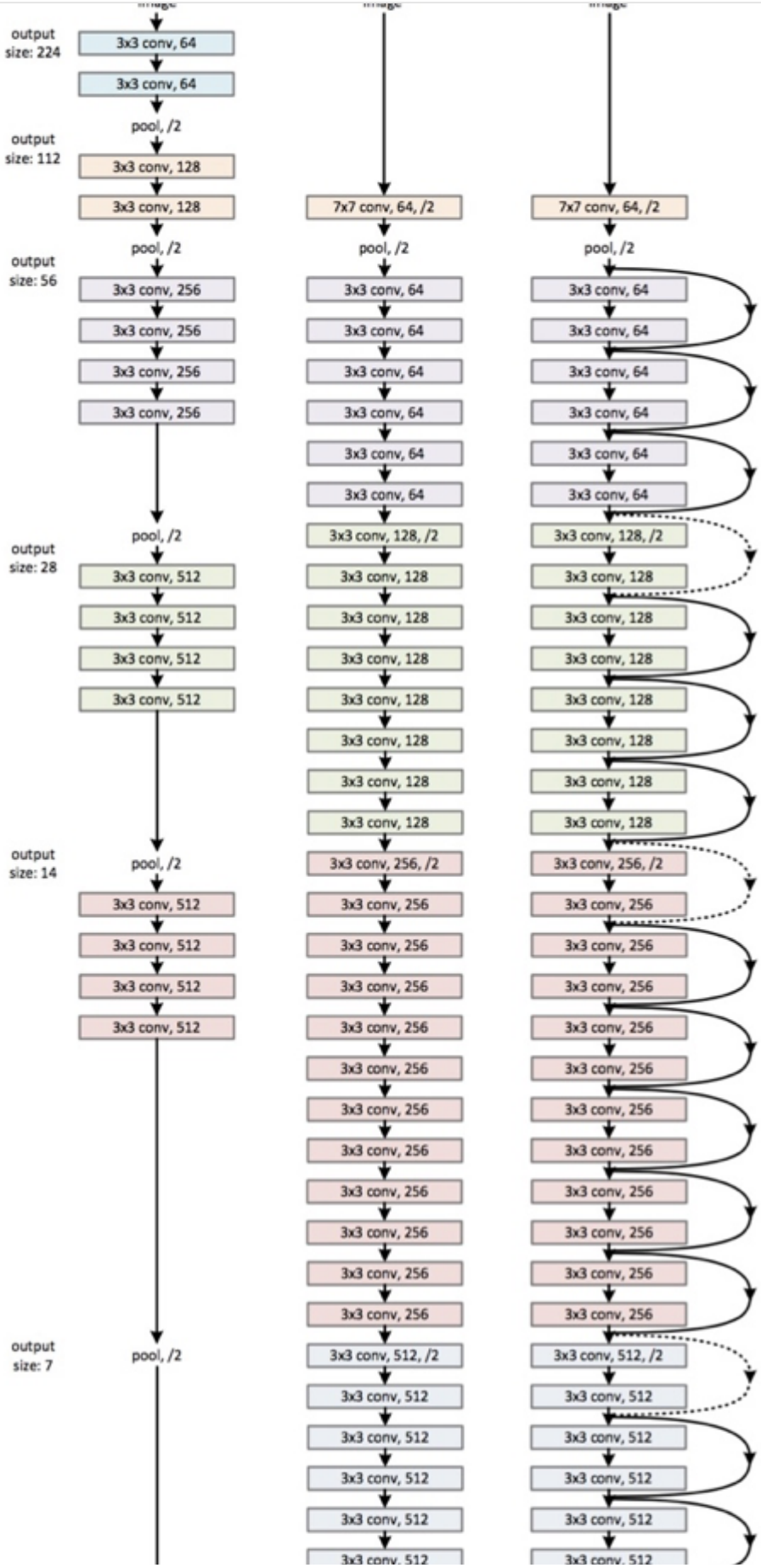


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

Figure 1. ResNet 34 from original paper [1]

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 [1] — 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.

conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
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!

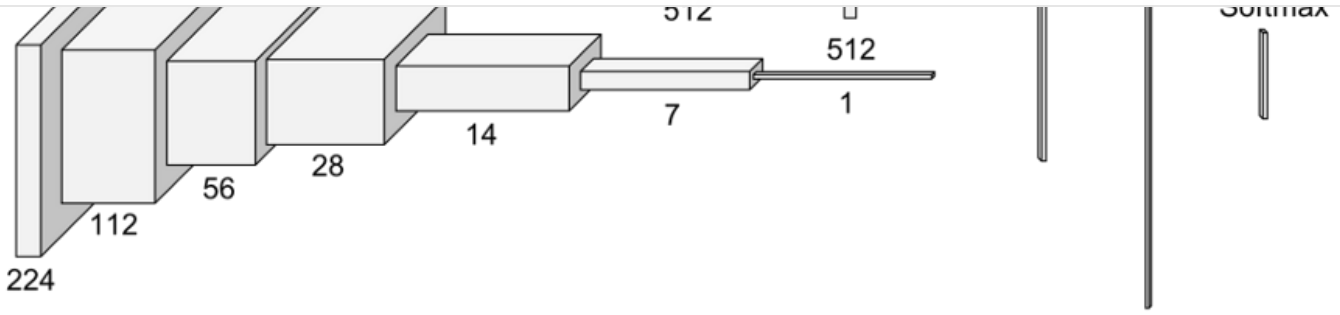


Figure 3. Another look at ResNet 34

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 (112x112x64) output volume — note this is free of the batch dimension to simplify the explanation.

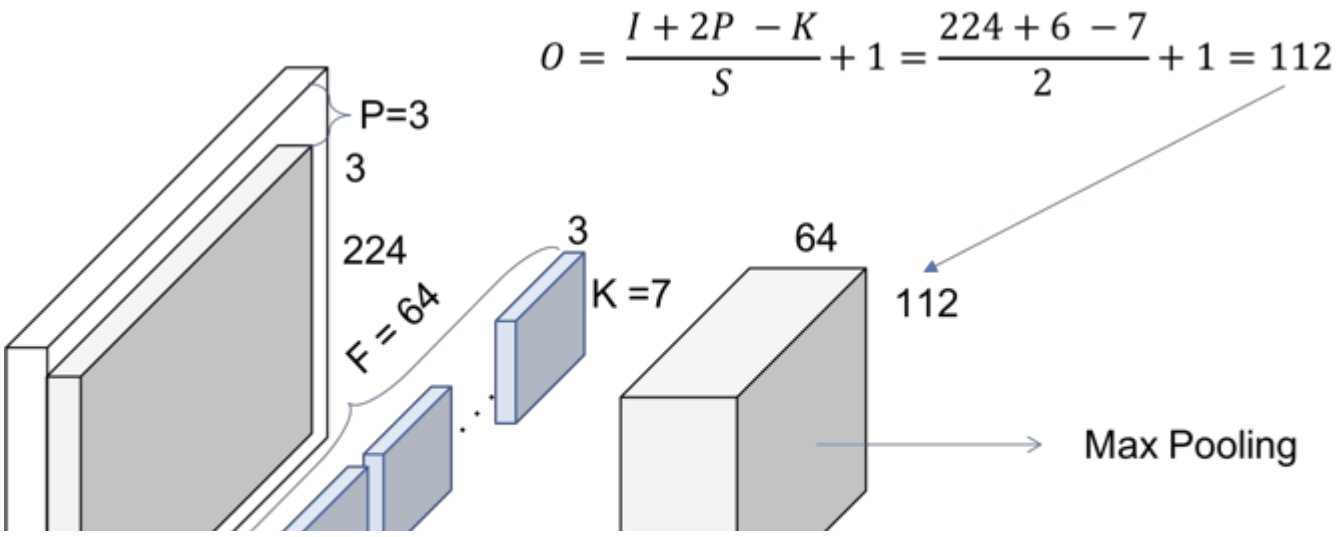


Figure 4. Conv1 — Convolution

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.

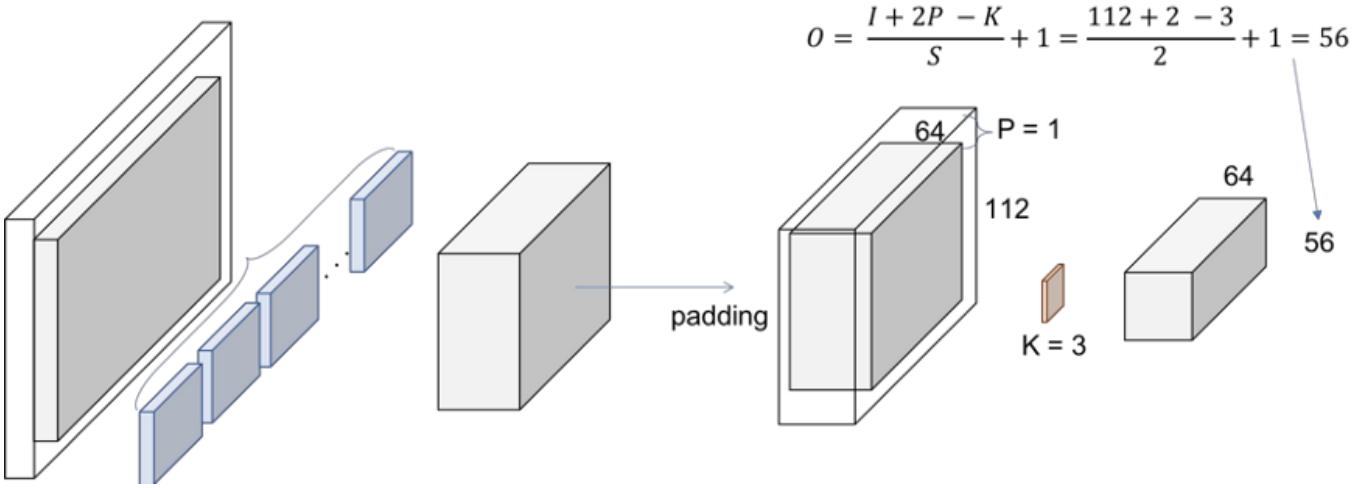


Figure 5. Conv1 — Max Pooling

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.

Block 1

1 convolution

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

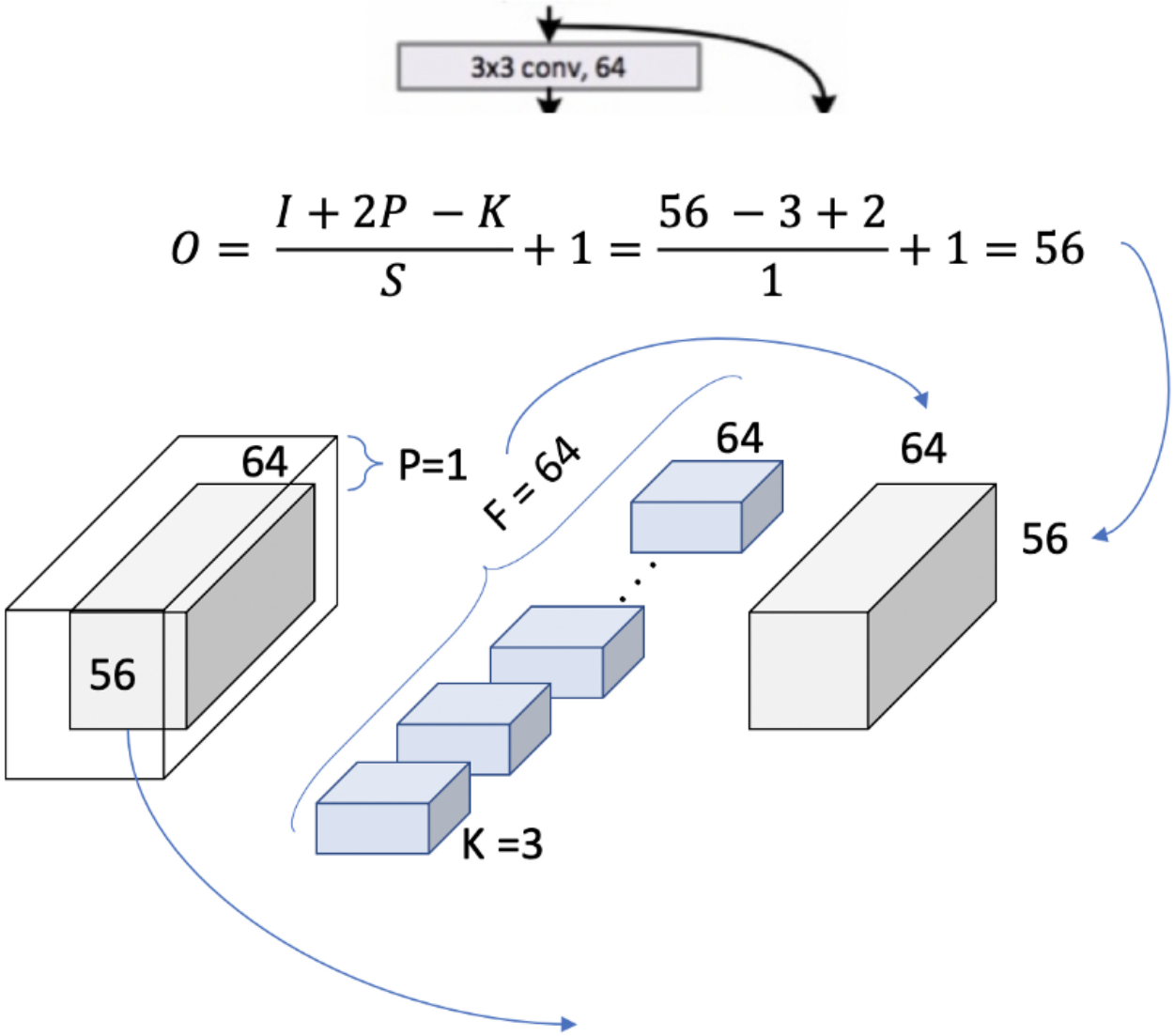


Figure 6. Layer 1, block 1, operation 1

We can double check now in the table from the paper we are using [3x3, 64] kernel and the output size is [56x56]. 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

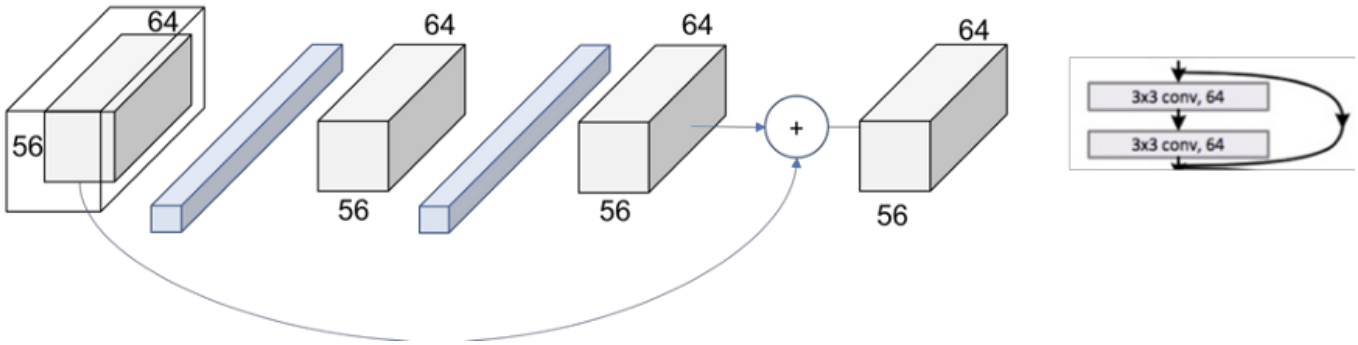


Figure 7. Layer 1, block 1

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 ***[3x3, 64] x 3 times within the layer***.

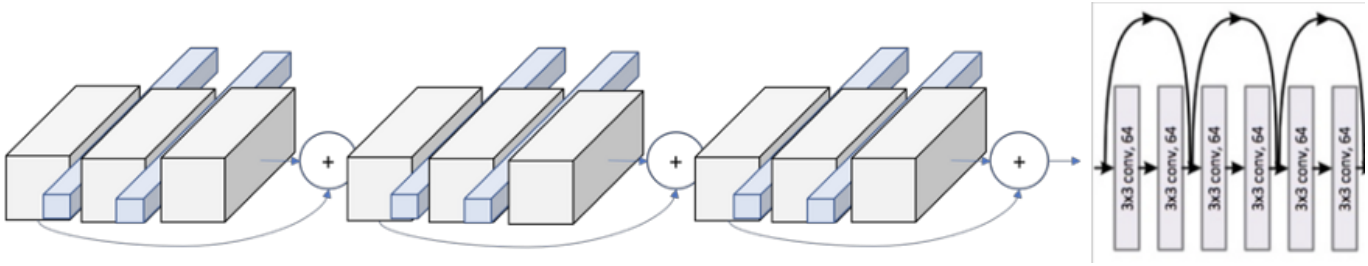


Figure 8. Layer 1

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.

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 ***1x1 convolutions***. Here, this second option is shown.

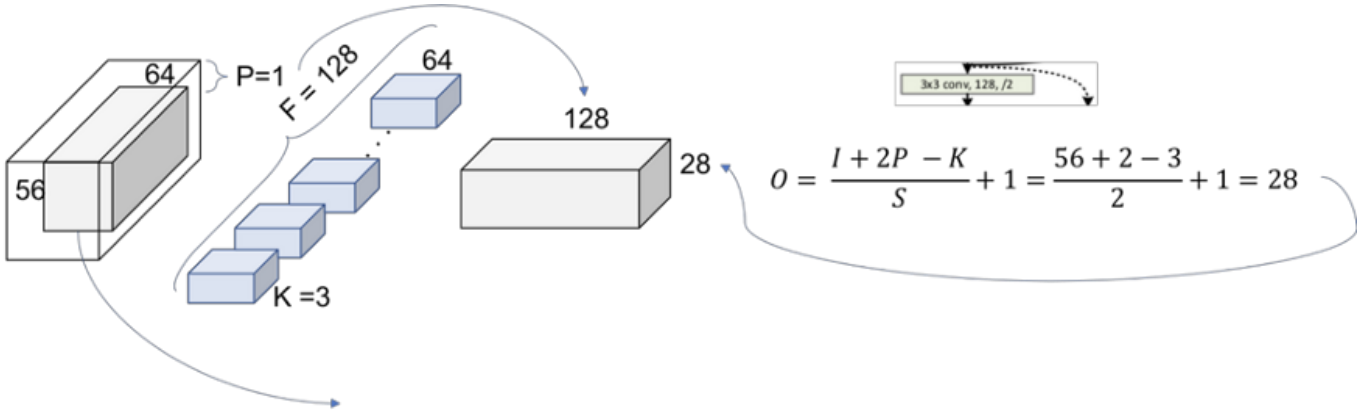
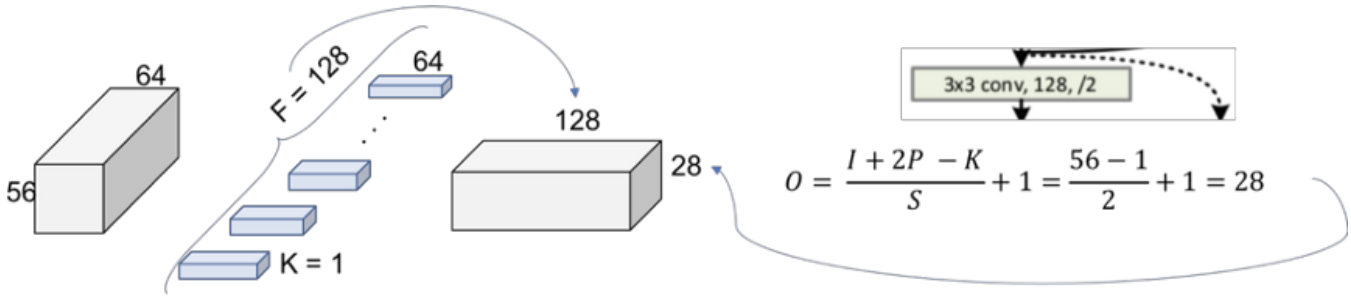


Figure 9. Layer2, Block 1, operation 1

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*64 = 28*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 1x1 convolution approach is shown in Figure 10.



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.

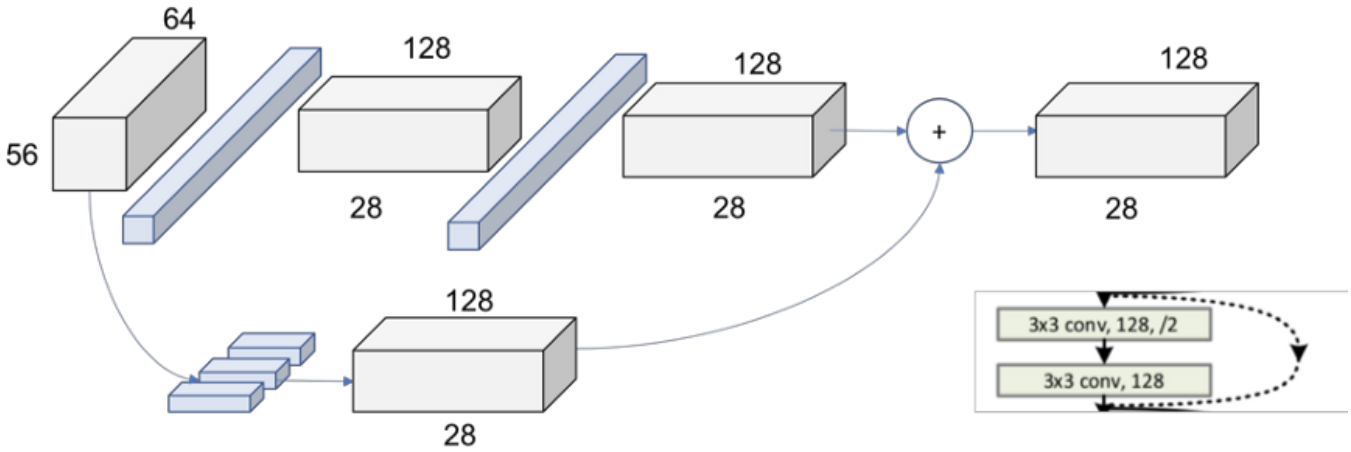


Figure 11. Layer 2, Block 1

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.

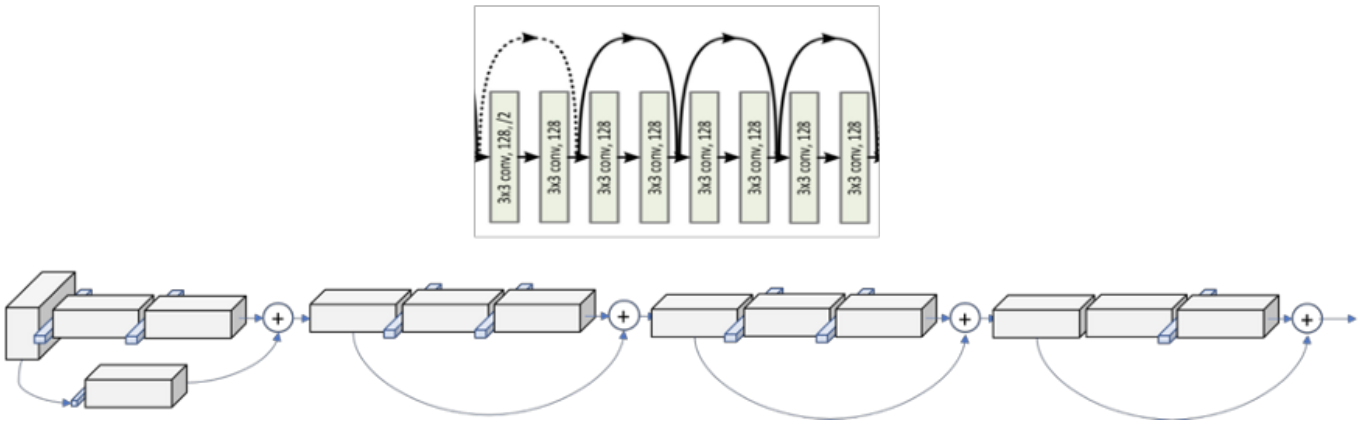


Figure 12.Layer 2

Summary

The ResNets following the explained rules built by the authors yield to the following structures as shown in Figure 2:

Number of Layers	Number of Parameters

ResNet 50	23.521M
ResNet 101	42.513M
ResNet 152	58.157M

Table 1. ResNets architectures for ImageNet

Bibliography

[1] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," in *CVPR*, 2016.