Residual Networks

Deep Residual Learning for Image Recognition

Github of the project : here

Biéchy Lucas
Cavalier Pierre

Introduction

Residual Learning is a concept born in 2015, in a context where new algortihmes try to develop the concept of "deep" networks. Indeed, the first algorithm, initiated by LeCun [1] in 1998, named LeNet is greatly outdone by AlexNet [2] who wins *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)* in 2012, then by VGG [3] and finally by GoogleNet [4] which wins this same contest in 2014 with a network of 22 layers. In this period, it is highlighted [3, 4] that the deeper a network is, the more it is performing.

One year later, Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun show in their article [5] that the accuracy of deep networks can be further improved with the concept of Residual Learning. This one pushes back, without increasing the complexity of the algorithms, the known limit of very deep learning (i.e. above 20 layers) : the exploding and vanishing [6, 7] gradient which results in an attribution of 0 and ∞ values to the gradient during the backpropagation, blocking the improvement of the network.

In this report, we will first summarize the theoretical part, then we will approach the experimental part where we will comment our results.

1 Theory

The origin of the problem comes from the following observation, using the CIFAR-10 data [8], it was noticed that increasing the depth of the neural network did not increase its accuracy and could even decrease it.

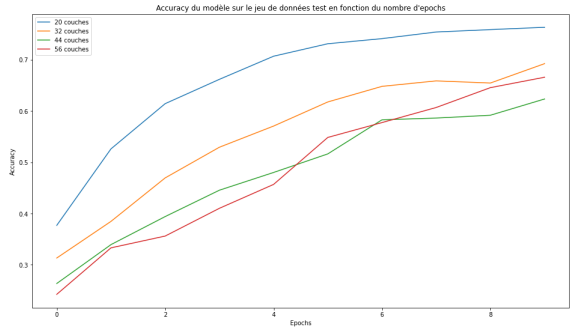

Figure 1.1 – Accuracy according to the number of epochs

For example, on this graph 1.1, we can see that going from 34 to 56 layers is not efficient, and the goal of Residual Learning is to make this complexity increase profitable. To do this, the goal is to add shortcuts in the following way :

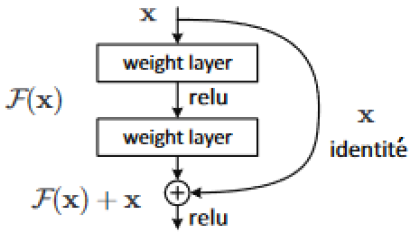

Figure 1.2 – Shortcut

The "shortcuts" are performed in the following way, after passing through a fixed number of layers and activation functions, the input value is added to the output, this process does not add any computational complexity, the number of parameters remains the same with option of zero-padding or increases by a few parameters depending on the depth of the architecture with option projection which is therefore largely negligible. The interest of Residual learning is that it acts as a kind of reset of the network depth and allows to refresh it regularly, which avoids the problem of "vanishing gradient" by allowing the signal to bypass a weight system.

In the context of the article we place a shortcut after two convolutions in two dimensions and a ReLU activation function. The networks being rather large, twenty layers at least, this gives a rather large number of shortcuts, a little less than half the number of layers.

The network described in the article is the ResNet34 defined in this way by 34 layers, hence its name.

Figure 1.3 – Architecture CIFAR-10

It is composed of several successive layers of convolutions of the same size which makes short cuts easy, the inputs being of the same dimensions as the outputs. However, the convolutions go from size 64 to 512 by powers of two, which forces a modification of the input, not being at the right dimension. To resize the inputs several choices are offered to us, either we proceed to a zero-padding to change the dimensions of the input, or we proceed to a projection. The method that will be chosen is the first one to increase the dimensions.

The model, in the article, is trained on the ImageNet database of 2012 with more than ten million images with ten thousand possible labels and obtains conclusive results, the network with thirty four layers with residual learning obtains better results than the one without residual learning but also than the one with twenty layers whether with or without residual learning, as shown in this graph :

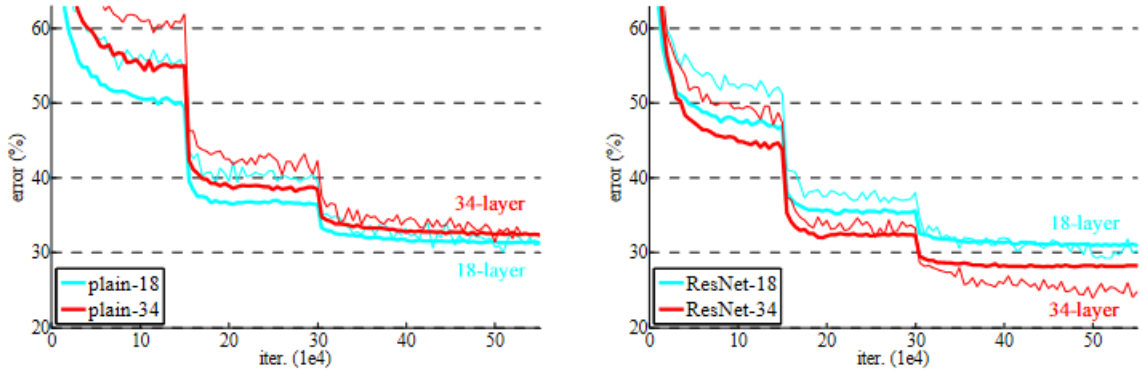

Figure 1.4 – Comparison with and without Residual Learning

To go to even deeper networks than thirty four layers, we proceed to a division of the layers before the shortcuts, passing, for example, from two 3x3 convolution layers by two 1x1 convolution layers and one 3x3 layer which allows to go from 34 to 50 layers. Other models have been created in the same way to obtain networks with 101, 152 and even 1202 layers.

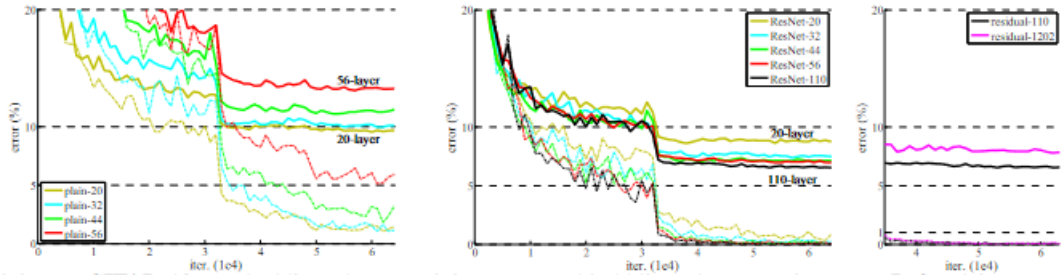

Figure 1.5 – Comparison of all created networks

Comparing their accuracy on this graph, we notice that the deeper the network is, the better its performance up to a certain number of layers. Indeed, the 1202 layer network is less accurate, although it has the same training error as the 152 layer network, this is due to overfitting and to the small data set.

2 Implementation

2.1 General

Our implementation follows the one given in the initial article. We use the batch normalization (BN) after each convolution and before each ReLU activation [9] as indicated in [10]. The drop out [11] will not be used. The network ends with an average pooling. Of course, we train the model only after initializing the weights (here with the default method of PyTorch, i.e. each weight follows a uniform law). Contrary to the paper and in order to decrease the computational resources, we use the Adam optimization algorithm [12] with its recommended parameters already implemented in PyTorch as well as a mini batch of size 32, a number of epochs of 10 and the images are neither cropped nor resized.

2.2 Case Study

As we have no technical possibility to use the ImageNet dataset [13] (containing 1.28 million training images), we will use the second dataset studied in the article : CIFAR-10 [8] consisting of 50 000 training images and 10 000 test images divided into 10 features. The architecture of our network is inspired by the figure 1 where the arrows represent in dotted line the shortcuts rectifying the dimension of the input and in solid line no change is necessary. The first layer of our network is a 3x3 convolution taking as input the three RGB channels of the images and as output 16 channels. Then we have n identical blocks each composed of two 3x3 convolutions, with an input equal to the output, then a shorcut. We repeat this block two other times for each channel, here respectively $16, 32, 64$. The convolution layers of dimension transition will have as indicated in the article a stride of 2. Finally the network ends with a linear classification. We will choose $n = \{3, 5, 7, 9\}$ giving then networks of $\{20, 32, 44, 56\}$ layers. The

shortcut can be done in two ways, either option (A) by adding a null padding to artificially increase the size, in this case the article advises to put a padding of 4 around each value, or option (B) by making a projection using a 1x1 convolution. In the rest of this report, we will abuse the language to speak of the accuracy and the loss of a model instead of their respective empirical mean on 5 samples of identical models as calculated in practice.

2.3 Verification of the implementation

This part is just to check if our algorithm is well implemented.

2.3.1 Comparison with the models *models.resnet()* from PyTorch

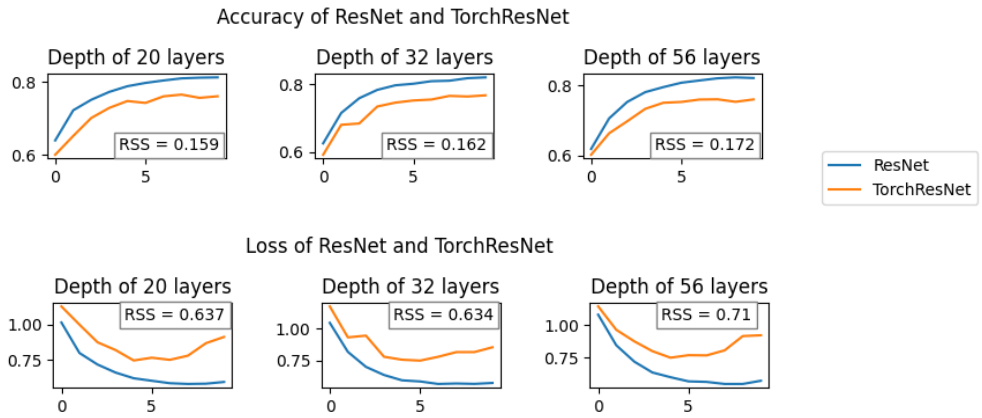

Figure 2.6 – Performance of TorchResNet and ResNet (with option (A))

Surprisingly, the implementation of the ResNet with PyTorch seems from the figure 2.6 to be much less efficient than the one we made. Therefore, we cannot directly conclude that our implementation of the ResNet is correct. However, the excellent performance of our algorithms is encouraging in terms of the positive direction our research has taken, namely the proportional improvement in performance as a function of network depth.

2.3.2 Verification of performance improvement with increasing depth

Because of the "vanishing and exploding gradients", if our network is not well implemented, the performance should decrease proportionally to the depth. Let's check graphically if this is the case.

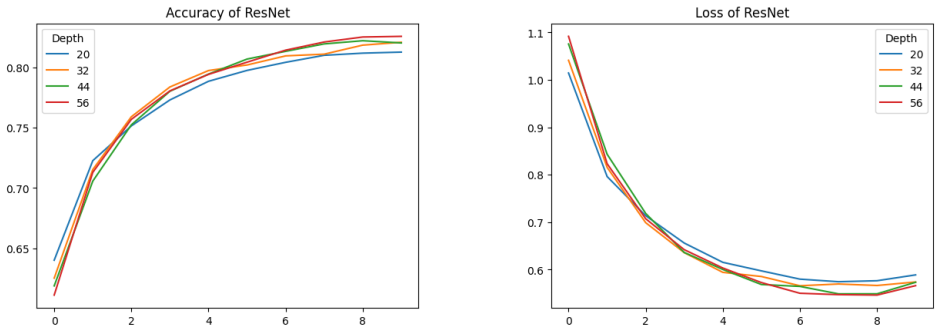

Figure 2.7 – Performance of ResNet (with option (A))

The figure 2.7 shows that the deeper our ResNet network is the better its performances are. The difference in accuracy and loss between our 20-layer and 56-layer networks are respectively 1.3% and 2.2% after 10 epochs. This reassures us on the implementation of our network which seems to work.

2.4 Comparison of options (A) and (B)

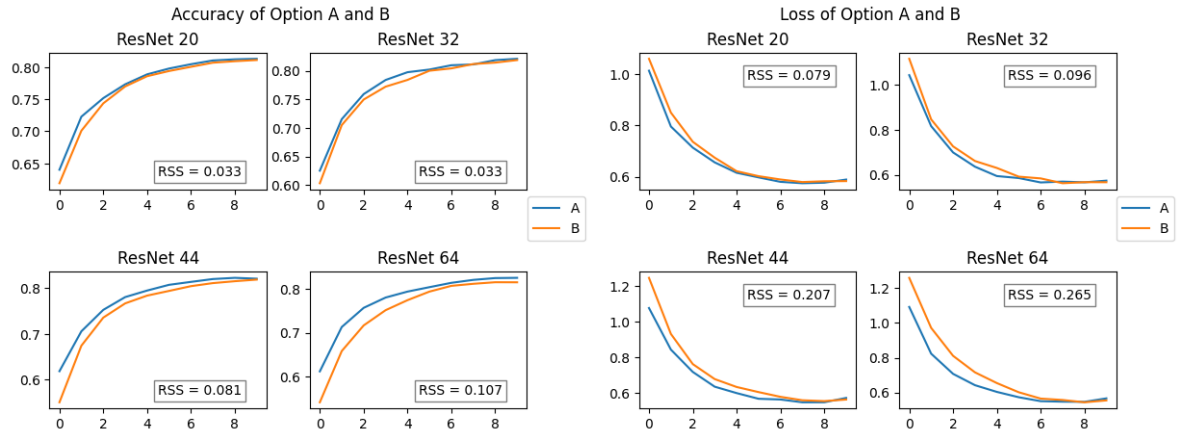

Figure 2.8 – Performance of Option (A) and (B)

It can be seen, either through the curves or the Sum of Squares Residuals (SSR) presented on the figure 2.8, that the two shortcut options are very similar (especially after some epochs). Moreover, option (A) offers slightly better overall performance than option (B). Therefore, like the original article, we will default to option (A).

2.5 Comparison with models without Residual Learning

The goal of this section is to confirm that the ResNet algorithm does indeed provide additional information to a classical Convolution Neural Network (CNN), i.e. without the shortcut. It should be noted that we have limited our experiments to 10 epochs for computational reasons, as it is generally from this point onwards that the performances of both networks decrease.

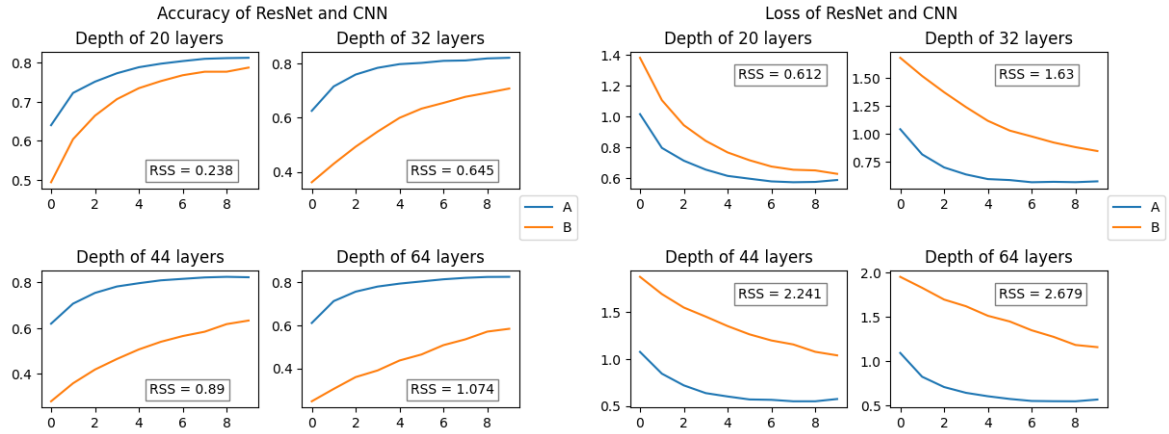

Figure 2.9 – Performance of ResNet and CNN

We can see with the figure 2.9 that ResNet performs much better than the CNN, especially when the depth is large. According to our results, ResNet improves on average the accuracy and loss of the CNN by about 28% and 43% after 10 epochs.

Conclusion

Through this report, we have seen the power of residual learning, which allows to increase the accuracy of a neural network without increasing its complexity, which explains why it is one of the major concepts of deep learning nowadays.

We have succeeded in reproducing, at our scale, conclusive results about the impact that residual learning can have on a neural network. It would have been interesting to evaluate the models on a larger number of epochs, but given our limited computing power and time, we preferred to compare the different models.

Bibliographie

[1] Yann LeCun et al. « Gradient-based learning applied to document recognition ». In : *Proceedings of the IEEE* 86.11 (1998), p. 2278-2324.

[2] Alex Krizhevsky, Ilya Sutskever et Geoffrey E Hinton. « Imagenet classification with deep convolutional neural networks ». In : *Communications of the ACM* 60.6 (2012), p. 84-90.

[3] Karen Simonyan et Andrew Zisserman. « Very deep convolutional networks for large-scale image recognition ». In : *arXiv preprint arXiv :1409.1556* (2014).

[4] Christian Szegedy et al. « Going deeper with convolutions ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, p. 1-9.

[5] Kaiming He et al. « Deep residual learning for image recognition ». In : *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, p. 770-778.

[6] Xavier Glorot et Yoshua Bengio. « Understanding the difficulty of training deep feedforward neural networks ». In : *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop et Conference Proceedings. 2010, p. 249-256.

[7] Donald F Specht et al. « A general regression neural network ». In : *IEEE transactions on neural networks* 2.6 (1991), p. 568-576.

[8] Alex Krizhevsky, Geoffrey Hinton et al. « Learning multiple layers of features from tiny images ». In : (2009).

[9] Vinod Nair et Geoffrey E Hinton. « Rectified linear units improve restricted boltzmann machines ». In : *Icml*. 2010.

[10] Sergey Ioffe et Christian Szegedy. « Batch normalization : Accelerating deep network training by reducing internal covariate shift ». In : *International conference on machine learning*. PMLR. 2015, p. 448-456.

[11] Geoffrey E Hinton et al. « Improving neural networks by preventing co-adaptation of feature detectors ». In : *arXiv preprint arXiv :1207.0580* (2012).

[12] Diederik P Kingma et Jimmy Ba. « Adam : A method for stochastic optimization ». In : *arXiv preprint arXiv :1412.6980* (2014).

[13] Olga Russakovsky et al. « Imagenet large scale visual recognition challenge ». In : *International journal of computer vision* 115.3 (2015), p. 211-252.