Deep learning - miniproject I

Hilsberg Hendrik, Pillet Maxime, Savioz Baptiste
27th May 2022

1 Introduction

The goal of this project is to build a network that denoises images, using two training sets of 50'000 noisy images. Once the network looks performant enough, we can test it using a validation set that contains two different sets : the first one is composed of noisy images and the second one of clean images.

2 Implementation of the network

We first started by implementing a network as simple as possible and then changed it to get better results. Its structure should take tensors of Nx3xHxW as input and output a tensor of the same dimensions.

From this, we can start implementing it and analysing the effect of the different layers. The most basic one is the convolution layer, followed by an activation function (e.g ReLU(x)). Using a network containing only these two operations obviously led to poor results, mainly due to blur. We tried to use Sigmoid as activation function but obtained worse results than with ReLU. Thus, we kept ReLU for all the layers.

Then, we added *maxpool* and *upsample* layers but most of the time, we get worse result than before. Indeed, we noticed that it blurred the images too much because of the maxpooling layer which reduces the image's size. Indeed, since the maxpooling operation looses pixel information, it's not suitable for our application. Then, to reduce this blurring effect, we thought about adding skip connections. This allows to work only on a part of the image, without changing the initial one too much. Finally, adding batch normalization, we observed that the training time decreases, since it shifts and re-scales the data according to the layer's input.

With the network presented on figure 1, the image's size is always the same but we increase and decrease the number of channels thanks to convolutions. Starting with 3 channels (RGB), we increase them to 32 using a kernel size of 3 followed by a ReLU activation and a batch normalization. Then, again, the number of channels is increased to 64 and passed to a ReLU function. At this point, a skip connection is done and the number of channels is augmented to 96. This is the maximum "depth" that we have with our network. Again, convolutions are done, followed by a ReLU activation and then, the skip connection is finished. The process of convolution, ReLU and sometimes batch norm is repeated many times until having 3 channels again. At the end, we added a clamping between 0 and 1 so that the output's pixels are in the good range of values. This architecture is finally composed of 10 convolutions, 10 activations and 4 batch normalizations.

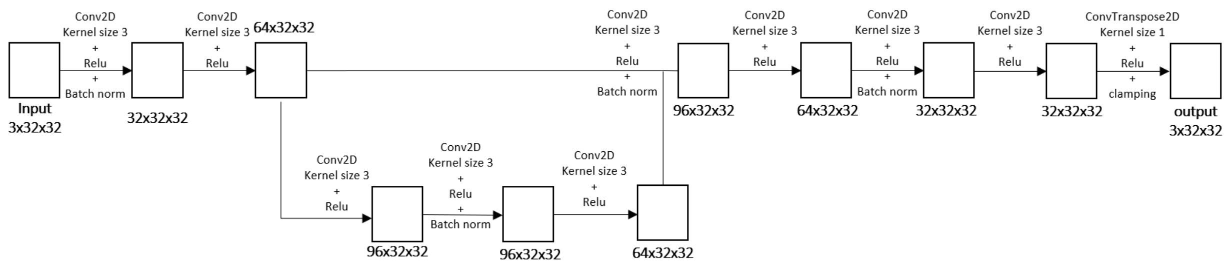

FIGURE 1 – architecture of our network

3 Results

Once we decided the structure of the network, we can play with different parameters to improve the global performance of the denoiser, based on the PSNR metric. The two following parameters have more influence than the others concerning the quality of the result and the computation speed : *nb_ epochs* and *mini_ batch_ size*. For the first one, if we increase the value too much, the computation time becomes too long and the performance doesn't improve that much. Indeed, as we can see in figure 2, from 10, a kind of plateau appears, so increasing the number of epochs doesn't change a lot the final performance. If we chose a too big value, the computation time becomes too long and the performance/computation time trade-off is not worth anymore. Based on the plot 2, we decided to train our network with 20 epochs since we get satisfying results.

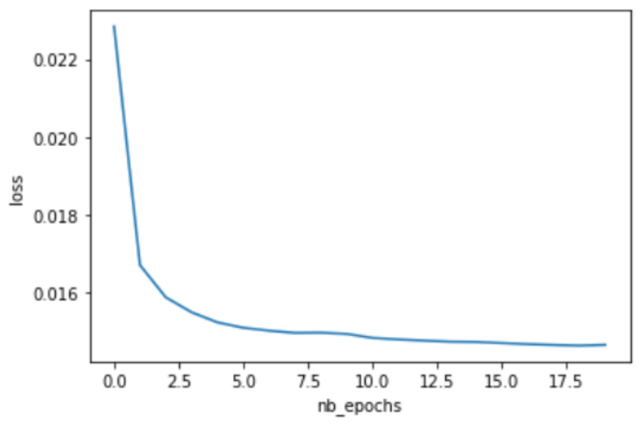

FIGURE 2 – loss in function of the number of epochs

Then, the batch size is an important parameter for the computation time. Indeed, this value affects the number of images that we train at the same time. With a too small value (e.g 50) the training time is too long since we need to repeat the process a lot. With a too big value, we are limited with the memory size since we can't stock everything at the same time. On the performance side (PSNR score), it doesn't change anything. We ended up with a batch size of 200.

To evaluate our network, we chose the mean square error. Indeed, we decided to keep the most basic one, used in the exercises since the results with other losses were almost the same. For the optimizer, we tried two different one : Adam and SGD. We kept Adam since it outperformed SGD in term of PSNR score.

With this network and these parameters, we obtained a PSNR score of 24.81 dB on the testing set with *test.py* file. Since we need at least 24 dB, our result is good enough.

4 Conclusion

Starting with a really basic network, we managed to create a model that can denoise an image pretty quickly based on two sets of noisy images. This is done using only 5 different operations : convolution, ReLU, batch normalization, transposed convolution and skip connection.