Deep Learning, Mini-projects I

# Noise2Noise model: image denoising network trained without clean references

## Using Pytorch modules

*Authors:*
Steven Brown,
Guillaume Briand and
Paulin de Schoulepnikoff

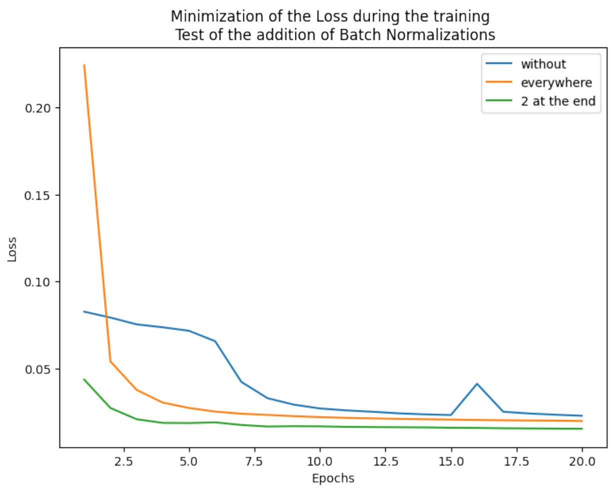
*Professor:*
Prof. François Fleuret

May 26, 2022

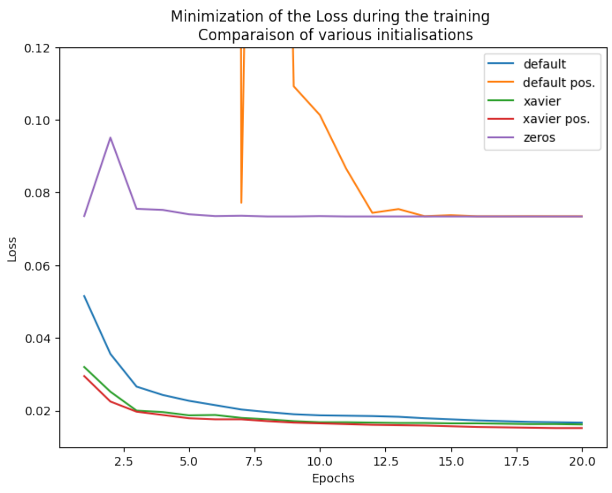
## 1 Implementation

### 1.1 Model

The goal of this first mini-project is the implementation of a denoising autoencoder using the PyTorch framework. A deep autoencoder combines an encoder composed of convolutional layers, with a decoder composed of transposed convolutions. After performing several tests on simple models, A model with 5 pairs of convolutional/transposed convolutional layers has been used. The latter is presented in Appendix A. As the train images have a fairly low resolution (see section 1.2), a kernel size of 2 and a stride of 1 was used each time. Tests for adding MaxPooling have been carried out. However, as the tests were not conclusive, the final model does not include MaxPooling. A padding of 1 was found to improve performance, thus this was added for each convolution. Given its popularity in the Deep Learning community, the ReLU activation function was used. It was also noted that the addition of BatchNorm before the last two convolutional layers (respectively after the first two transpose convolutional layers), made training easier. Indeed, several tests were carried out and, as illustrated in Figure 1(a), the choice of two BatchNorms at the end made training easier.



(a)



(b)

Figure 1: (a) Testing the effect of adding Batch Normalization to the model. Three configurations are compared: without BatchNorm, with BatchNorms everywhere and with two BatchNorms at the end. The last choice is optimal. (b) Testing of different types of initialization of the model parameters: default pytorch, default positive, Xavier, Xavier positive and just zeros. The positive Xavier is optimal. The simulations were performed on a train set of 1'000 samples, with a batch size of 100, during 20 epochs and with momAGD optimizer.

For the initialization of the model parameters, several possibilities have been taken into account and compared: the default pytorch initialization, the Xavier initialization and just setting everything to zeros. Since the inputs represent images with positive values, it also makes sense to initialise the weights with positive values. The "positive" versions of pytorch default and Xavier have therefore also been considered. They correspond to, after the standard initialization, setting all negative values to zeros. The results of these tests are shown in Figure 1(b). The Xavier positive version is found to be optimal.

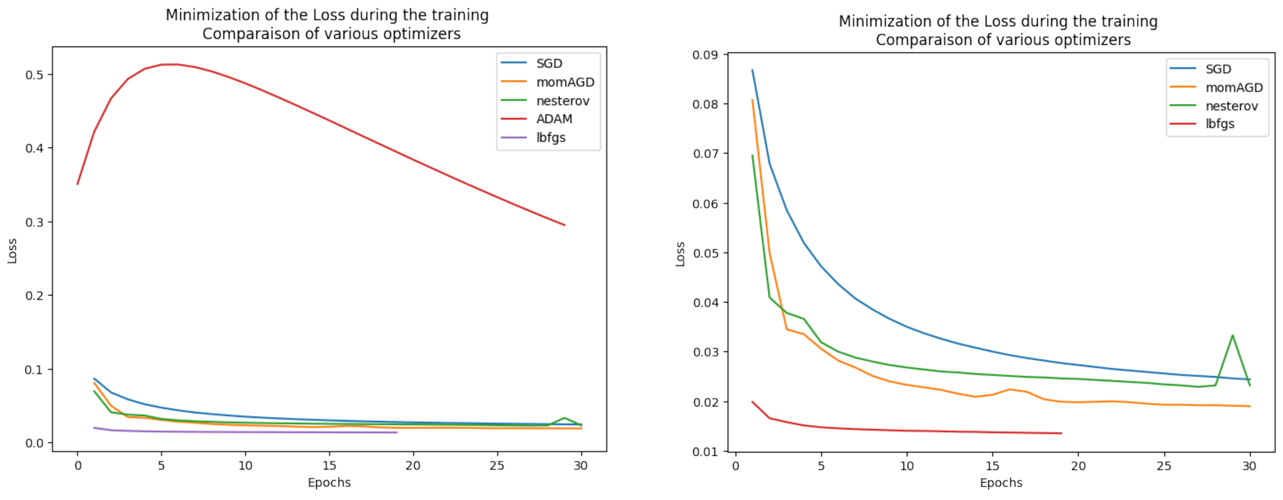
## 1.2 Training set

The model that is implemented, called Noise2Noise, has the particularity to train on two sets of noisy images. They are composed of the same images and uncorrected noises have been applied to them. One of the images from these sets is shown in Figure 4(a) and 4(b) in Appendix B.

In order to increase the size of the training sets, it is possible to train the model also on vertical and horizontal flip images, see Figure 4(b) in Appendix B.

## 1.3 Training

In order to train the model, a loss has been defined. In this project, the MSE loss has been used. Its minimization is performed by using an optimizer. Several options are available. The simplest choice is the Stochastic Gradient Descent (SGD). The latter can also be improved by taking into account the momentum, leading to momentum assisted Gradient Descent (momAGD) and Nesterov. Another optimizer, very popular in the deep learning community is ADAM. These optimizers are based on a Taylor series of the loss function of the first order. It is therefore possible to improve these optimizers by pushing the development to the second order, which leads to the Newton optimizer. However, the latter is very costly as it requires the computation and inversion of the Hessian matrix. It is however possible to approximate the latter with so-called quasi-Newton methods. An optimizer employing this strategy is LBFGS.



(a) (b)

Figure 2: (a) Test of the different optimizers with a zoom (b). Surprisingly, poor results are obtained with ADAM. As expected, the best performance are obtain by LBFGS. The simulations were performed on a train set of 1'000 samples, with a batch size of 200, during 30 epochs and with a learning rate of 0.05.

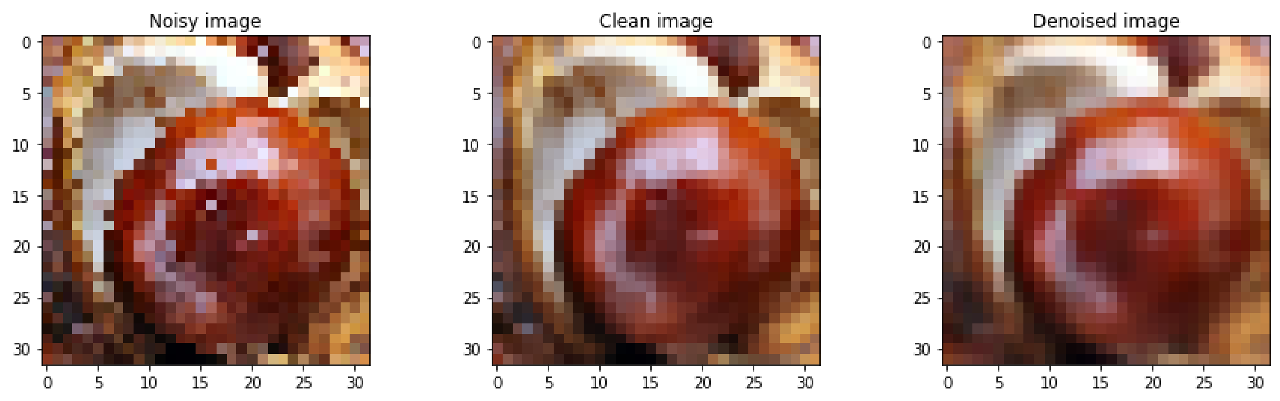
All these optimizers have been tested. The results are shown in Figure 2(a) and (b). Surprisingly, poor results are obtained with ADAM. Apart from that, as expected, SGD gets the worst performance. A faster convergence is obtained by adding momentum (with momentum coefficient of 0.9) and finally, the best performance is obtained by LBFGS.
It was therefore decided to use a hybrid method. The latter consists in performing the first 5 iterations with LBFGS and then performing the rest of the training with momAGD. This idea is motivated by the fact that, since momAGD accumulates the gradients through $\nu$, starting the accumulation at the strategic location of the parameter space would allow to have better performance. LBFGS being computationally very expensive, it is not possible to use it throughout

the training. However, given its accuracy, it is an ideal choice for these first iterations. This method can be used by setting *hybrid=True* in the *train()* function parameters.
With this method, the two main hyperparameters to optimize are the learning rate $lr$ and the momentum coefficient $\mu$. Several tests were performed and the optimal choice was found to be $lr = 0.05$ and $\mu = 0.9$.

## 2 Results

As a consistency check, the model was first trained on a single simple. This allowed the model to be tested for its ability to crimp the identity map. The results are presented in Figure 5 in Appendix B. With a training of 20 epochs which lasted 11sec, a psnr of 74.5 was obtained.

Then, a more consequent training was performed on the training data set. Since the training can only take about 10 minutes on a GPU and the basic train set is already large, only a fraction of the vertically flipped images were added. The training was done on an epoch with a minibatch size of 100. Using Collab's GPUs, it lasted 8min and 22sec. In order to test the performance of the model, it was used to denoise a new set of noisy images. As the clean images were also provided, it was possible to evaluate the model with a method called psnr. After this training, a psnr of 25.08 dB has been obtained. An example of a denoised image is shown in Figure 3.



(a) (b) (c)

Figure 3: Example of a noisy image (a), the clean version (b) and the one denoised by the model (c).

The denoised image is satisfying but a bit blurry, which was predictable given that MSE loss was used. Indeed, due to the uncertainty of the location of a given object, the best one can do with MSE loss is to average over all the location of the object and this generate a blurry signal. In order to improve the result, a possible solution would be to sample according to $\mu_{X|Y=y}$, which can be achieved by implementing a conditional GAN.

## References

[1] Sutskever, Ilya and Martens, James and Dahl, George and Hinton, Geoffrey, *On the importance of initialization and momentum in deep learning*, PMLR, 2013, https://proceedings.mlr.press/v28/sutskever13.html

[2] Fangyu Zou and Li Shen and Zequn Jie and Ju Sun and Wei Liu, *Weighted AdaGrad with Unified Momentum*, 2019, arXiv:1808.03408

[3] Diederik P. Kingma, University of Amsterdam, OpenAI , Jimmy Lei Ba, University of Toronto, *adam: a method for stochastic optimization*, Published as a conference paper at ICLR 2015, 30 Jan 2017 , arXiv:1412.6980v9

[4] Glorot, A. Bordes, and Y. Bengio, *Deep sparse rectifier neural networks*, In International Conference on Artificial Intelligence and Statistics (AISTATS), 2011.

[5] Krizhevsky, I. Sutskever, and G. Hinton, *Imagenet classification with deep convolutional neural networks*, Neural Information Processing Systems (NIPS), 2012.

[6] L. Maas, A. Y. Hannun, and A. Y. Ng, *Rectifier nonlinearities improve neural network acoustic models*, ICML Workshop on Deep Learning for Audio, Speech and Language Processing, 2013.

[7] He, X. Zhang, S. Ren, and J. Sun, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, CoRR, abs/1502.01852, 2015.

[8] Xu, N. Wang, T. Chen, and M. Li, *Empirical evaluation of rectified activations in convolutional network*, CoRR, abs/1505.00853, 2015.

[9] Clevert, T. Unterthiner, and S. Hochreiter, *Fast and accurate deep network learning by exponential linear units (elus)*, CoRR, abs/1511.07289, 2015.

[10] simonyan and zisser 2014

[11] Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, International Conference on Artificial Intelligence and Statistics (AISTATS), 2010.

[12] Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, *Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies*, pages 237–243. IEEE Press, 2001.

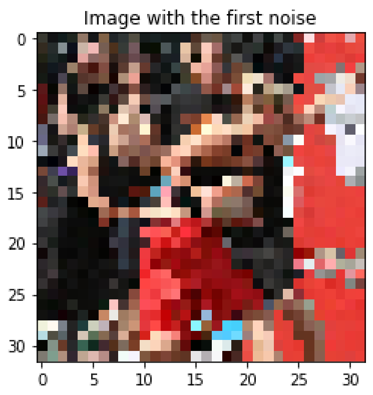
[13] Balduzzi, M. Frean, L. Leary, J. Lewis, K. Wan-Duo Ma, and B. McWilliams, *The shattered gradients problem: If resnets are the answer, then what is the question?*, CoRR, abs/1702.08591, 2017

## A Appendix : Additional code

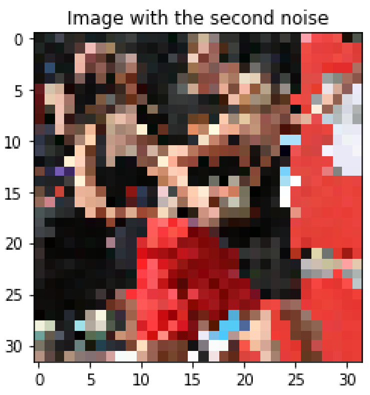
### Deep Auto-encoder

```
    Sequential(
   #encoder
  (0): Conv2d(3, 64, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (1): ReLU(inplace=True)
  (2): Conv2d(64, 64, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (3): ReLU(inplace=True)
  (4): Conv2d(64, 128, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (5): BatchNorm2d(128)
  (6): ReLU(inplace=True)
  (7): Conv2d(128, 256, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
  (8): BatchNorm2d(256)
  (9): ReLU()
  (10): Conv2d(256, 512, kernel_size=(2, 2), stride=(1, 1), padding=(1, 1))
 #decoder
  (11): ConvTranspose2d(512, 256, kernel_size=(2, 2),
        stride=(1, 1), padding=(1, 1))
  (12): BatchNorm2d(256)
  (13): ReLU(inplace=True)
  (14): ConvTranspose2d(256, 128, kernel_size=(2, 2),
        stride=(1, 1), padding=(1, 1))
  (15): BatchNorm2d(128)
  (16): ReLU(inplace=True)
  (17): ConvTranspose2d(128, 64, kernel_size=(2, 2),
        stride=(1, 1), padding=(1, 1))
  (18): ReLU(inplace=True)
  (19): ConvTranspose2d(64, 64, kernel_size=(2, 2),
        stride=(1, 1), padding=(1, 1))
  (20): ReLU(inplace=True)
  (21): ConvTranspose2d(64, 3, kernel_size=(2, 2),
        stride=(1, 1), padding=(1, 1))
)
```

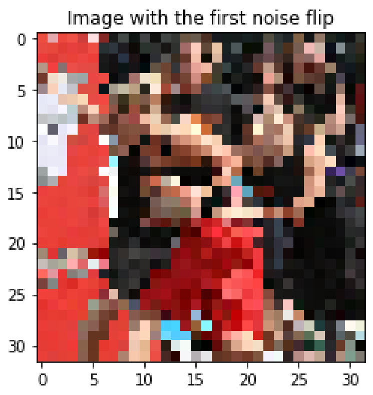
## B Appendix : supplementary figures



(a)

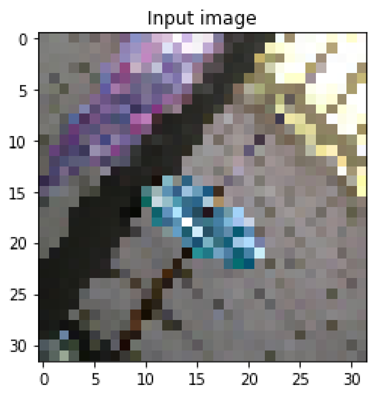


(b)

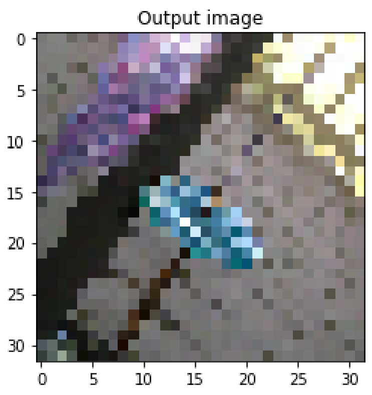


(c)

Figure 4: Image of train set 1 in (a) and train set 2 in (b). In order to increase the number of training samples, it was also possible to flip the images vertically (c).



(a)



(b)

Figure 5: Consistency check: training on a single input, (a), to check that the model can reconstruct the same image, (b). This result, gives a psnr of 74.5.