Noise2Noise on Pytorch framework : Miniproject 1 for Deep Learning (EE-559)

Elia Fantini, Kaan Okumuş, Kieran Vaudaux
EPFL, Switzerland
May 27, 2022

***Abstract*—Noise2Noise paper [1] from 2018 applied a Convolutional Neural Network to learn how to restore images and remove noise by training it using only noisy pictures. Applying basic statistical reasoning to signal reconstruction, they managed to achieve performances from similar to better of previous methods, which used clean targets for the supervised training. In this project we applied the same principles on a dataset of 50000 pairs of 32x32 images, corrupted by two different noises. By reducing the size of the original network thanks to the introduction of weight sharing and less convolutional filters, we achieved a PSNR score of 25.55 dB in less than 8 minutes of training and a fast convergence speed that reaches close to minimum test loss and over 25 dB of PSNR after just 2 epochs.**

I. Introduction

The procedure of removing noise to improve the quality of an image, also called denoising, has been applied since many years. With the rise in popularity of deep networks and in particular with Convolutional Neural Networks (CNN), results on denoised images improved significantly. On the other hand, CNN require a lot of data to be trained and obtaining pairs of noisy images with the corresponding clean groundtruth is not so easy in all applications. Noise2Noise paper [1] solved this problem by using noisy images as the targets of training. Their intuition was to use a property of L2 loss minimization: on expectation, the estimate remains unchanged if we replace the targets with random numbers whose expectations match the targets. In other words, corrupting the training targets of a neural network with zero-mean noise will not change what the network learns. A similar reasoning can be applied with other losses, such as L1 loss. In our project, we were given a dataset of 50000 pairs of 32x32 images. In every pair, the first picture is corrupted by the same but unknown noise, while the second picture is corrupted as well, but with a different noise. We were also given additional 1000 pairs of noisy images with the corresponding clean targets to validate our training. Performance was measured using the peak signal-to-noise ratio (PSNR), a ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. It was calculated with the following equation:

$$PSNR = 20 \log_{10} MAX_I - 10 \log_{10} MSE$$

MAX_I is the maximum possible pixel value of the image, whereas MSE is the mean square loss (L2). In our case MAX_I was 1, since images were scaled in the range [0,1] before being used for training.This is because neural networks process inputs using small weight values, thereby inputs with large integer values can derange or slow down the learning process. A model with large weight values is often unstable, meaning that it may suffer from poor performance during learning and sensitivity to input values resulting in higher generalization error.

II. Data preprocessing

A. Data augmentation

The network used by the Noise2Noise paper is an adaptation of the UNet [2] architecture, which became successful thanks to the fact that it does not require a huge amount of data but it relies on a strong use of data augmentation. The number of images we were given was big, but the size of them was not, hence the information provided was less. We decided to augment our data mainly to improve the generalization power, since overfitting on a small dataset is otherwise really easy. The augmentation doubled the number of final pictures, dividing the original dataset into five parts and applying to each of them a different random permutation of vertical and horizontal flips, as well as ± 90 and 180 rotations.

B. Data normalization

Normalizing the data generally speeds up learning and leads to faster convergence, hence we performed standardization on every channel of the image, subtracting the channel-wise pixels' mean and dividing everything by the standard deviation.

C. Results

We measured the difference between the performance when using and not using the preprocessing methods during 15 epochs training. This way we could asses the effectiveness of a particular method. Both data augmentation and normalization have shown a slight increase in the stability and speed of convergence. Analyzing mean and standard deviation of training and test data, we noticed that they are very similar, so the improved generalization property of data augmentation is not so noticeable on this dataset. More details on resulting metrics can be found in table I.

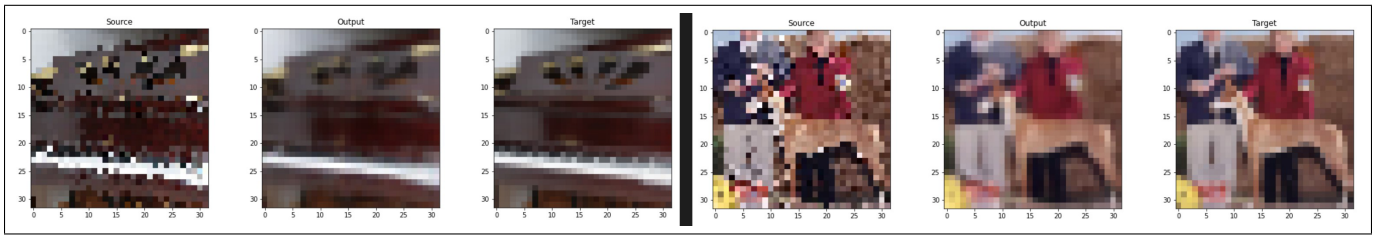

Fig. 1. Final model: example of noisy image (left), model's output (center), and clean target (right).

III. Models and Methods

TABLE I
BEST PSNR IN 15 EPOCHS AND RELATED TEST AND TRAIN LOSSES

	PSNR (dB)	Train L.	Test L.
Batch 400, channels depth 48			
Noise2Noise UNet	24.86	0.0151	0.0040
Noise2Noise + BatchNorm2D	24.89	0.0151	0.0039
Our UNet (L2 loss)	25.37	0.0146	0.0035
Our UNet with L1 loss	24.68	0.0721	0.0389
OurUNet + normalization	25.28	0.0146	0.0037
OurUNet + norm + augm	25.36	0.0145	0.0036
Batch 100, channels depth 24			
OurUNet + norm + augm	**25.65**	0.0145	**0.0033**
OurUNet+norm+augm+sched	25.60	**0.0144**	0.0034

A. Noise2Noise UNet

As a first model, we reproduced the modified version of UNet [2] as it was implemented in the original Noise2Noise paper [1]. This architecture contains 5 blocks that act as an encoder and 6 constituting the decoder. Every encoding sub-part is made up of a convolutional layer, followed by a LeakyReLU activation function and a max pooling layer, with an initial channels' depth of 48. The deconding part is made of convolutional and transposed convolutional layers. The output of each upsampling layer is concatenated with the output of the corresponding downsampling layer, as typical in every UNet architecture. The last layer gets back to 3 channels and a 32x32 image, and the last activation function is a rectified linear unit (ReLU). A big benefit of this function is that it doesn't activate all nodes at the same time and is simple and fast to compute since it converts all negative inputs to zero and the node does not get activated. If the input is positive, it's not modified in any way. As a drawback, when a node isn't activated, the gradient is equal to zero and during backpropagation, all the weights will not be updated. LeakyReLU solves this problem, and for this reason all hidden layers have that instead of the simpler ReLU. With a batch size of 400 and 15 epochs of training, the network reached a PSNR of 24.86 dB and started to converge faster after 7 epochs (see figure 2). As a first modification to the original architecture, we tried to put a BatchNorm2D layer after every convolution: the convergence speed improved a lot (see figure 2) and the network reached a PSNR of 24.89 dB. See I for further details.

B. Our UNet

Noise2Noise architecture was made for much bigger datasets and images than the one we trained our network with, so we created a lighter and less flexible version of the architecture described above, so that the training and convergence speed would have been much faster and it would have avoided an excessive overfitting. To do so, we introduced weight sharing by removing many downsampling and upsampling blocks and applying the same block over and over again, consecutively. We also changed the initial channels' depth to 24, and switching LeakyReLU with the simpler ReLU stabilized the convergence. By keeping every other parameter the same (batch, epochs, etc.) but introducing weight sharing, we reached a PSNR of 25.3 dB and the model started to converge fast from the very beginning of the training (see figure 2). Subsequently,decreasing the channels' depth helped to reach 25.5 dB. Finally, we decreased the decreased the batch size to 100 to get an even higher PSNR of 25.65 dB, as shown in table I.

C. Optimizations

Once the architecture of the neural network was chosen, we had to choose other components. Good choice of the loss function and optimizer are crucial for good performance.

1) Loss function: As explained in the introduction, L2 loss has the important property at the core of the training with noisy targets: the estimate remains unchanged if we replace the targets with random numbers whose expectations match the targets. If the noise does not have a zero mean distribution though, using L1 loss can lead to better performance, since it recovers the median of the targets, meaning that neural networks can be trained to repair images with significant outlier content, again only requiring access to pairs of such corrupted images. Since we didn't know the distribution of the noise our dataset had been corrupted with, we tried with both criterions. Using the lighter architecture to minimize the L2 loss lead us to the 25.3 dB PSNR score, whereas minimizing L1 loss reached a lower maximum of 24.7 dB PSNR (see table I and figure 2). Given this results, we suppose that the applied noise has a mean which is close to zero, without a lot of outliers.

2) Optimizer: Optimizers are another key element of the training process. Among many different algorithms, the most popular is by far Adam, which usually requires little tuning

on the learning rate hyperparameter. The original Noise2Noise implementation used a learning rate of 0.001, beta 1 equal to 0.9, beta 2 equal to 0.999, and epsilon equal to 10^{-8}. Making the betas lower caused a very low performance, whereas increasing the learning rate made convergence much faster but also unstable. A learning rate of 0.006 reached a PSNR over 25 dB in just 3 epochs, but it then stopped converging, whereas higher values diverged. The best solution in the long term would have been the original value of 0.001, but we added a scheduler that divided by half the learning rate every 3 consecutive epochs that the test loss kept getting worse. With an initial learning rate of 0.006 and the use of the scheduler, the model reaches a PSNR over 25 dB in just 3 epochs and a final one of 25.64 dB, as shown in table I.

IV. Results and Conclusions

The main focus of our experiments was to achieve the fastest convergence possible in order to get the best performance in less than 10 minutes of training. The key to achieve this result was to reduce the size of the original architecture thanks to weight sharing, and another improvement was given by reducing the channels' depth. The best result was obtained with a learning rate for Adam of 0.001, but since we aimed at a fast convergence, we have chosen as final model the solution starting with a learning rate of 0.006, reduced by a scheduler over time. This way our model reaches a PSNR higher than 25 dB after the 2nd epoch, 25.6 dB at the 12th epoch and 25.61 at the 20th, after only 455 seconds. This results are shown in the last plot of figure 2. Longer training did not result into significantly better results, hence the fastest solution is also the best we could achieve. Every test loss was computed on a random subset of the validation data to avoid overfitting, since the scheduler computations were dependant of the test loss values. The final PSNR on the whole set is 25.55 dB, an example of the resulting output is shown in figure 1.

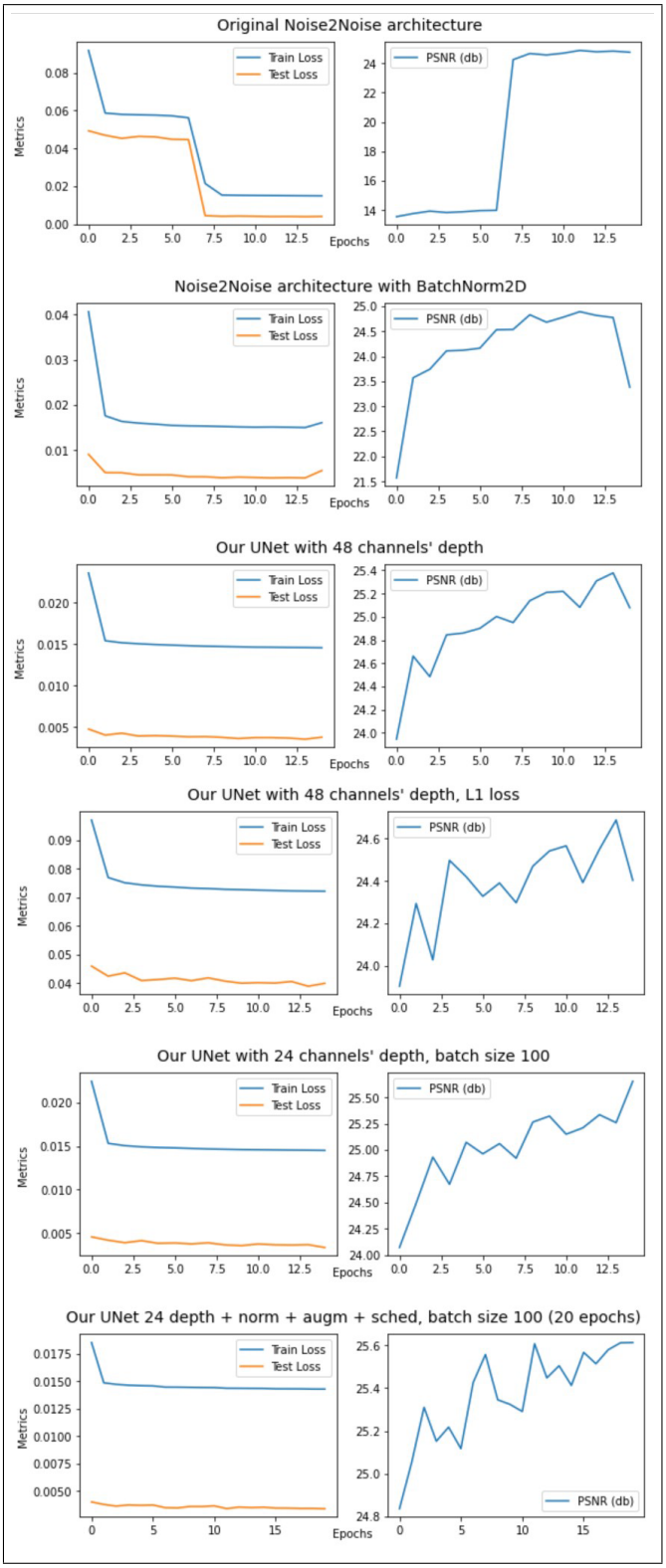

Fig. 2. Evolution of losses (left) and PSNR (right) over epochs

References

[1] Jaakko Lehtinen et al. "Noise2Noise: Learning Image Restoration without Clean Data". In: *CoRR* abs/1803.04189 (2018). arXiv: 1803.04189. URL: http://arxiv.org/abs/1803.04189.

[2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: (2015). Ed. by Nassir Navab et al., pp. 234–241.