Noise2Noise: Denoising autoencoder

Alessio Verardo Arthur Brousse Emna Fendri

Spring 2022

Abstract

Signal or image reconstruction is today an important and popular subfield of Machine Learning. This paper explores several ways of reconstructing clean images given only pairs of noisy images for the training and assess their performance in this unsupervised Machine Learning task.

1 Introduction

With the advent of neural networks in computer vision, and in particular convolutional neural networks, treatment of large amount of images became an easier task. Image restoration is one such example. Imperfect images appear everywhere, from picture taken in low light with smartphone to issues such as corrupted packets when downloading from the web. This project has two main objectives: trying to reproduce the results obtained by the Noise2Noise network [1] as well as exploring different ways of potential improvements. As part of the project, we were given a time constraint of 10 minutes for training our network.

2 Methodology

2.1 Data

The given training data consists of 50'000 pairs of 32 × 32, RGB and noisy images (examples of such noisy inputs can be found in the left images of fig. 2). To assess the performance of our models, we were given a set of 1'000 noisy and clean image pairs (the two left images in fig. 2). We selected the first 100 of them as our validation for some tuning of the parameters and the remaining 900 pairs are used as a test set to compute the global performance of the networks.

2.2 Baseline model

As the baseline model, we use the Noise2Noise network as described in the original paper [1]. This neural network is a convolutional neural network with 5 downsampling blocks, forming the *encoder*, followed by 5 upsampling blocks, forming the *decoder*. The architecture is depicted in table 1. Let's try to understand what does the given structure represent intuitively. More specifically, let's try to understand what is an **inductive bias**. It is defined as *A central factor in the application of machine learning to a given task is the inductive bias, i.e. the choice of hypotheses space from which learned functions are taken* in [2]. In convolutional neural network, the inductive bias is that images are invariant to translation. It means that if the network learns to detect a horizontal edge in the upper corner of the images, it should use this knowledge to detect horizontal edges in any part of the images. Additionally, this network also uses max-pooling which implements the following inductive bias: *small translations should not have a large effect on the output of the neural network.* The two previous building blocks are classical components of a neural networks for images. One of the implementation choice made by the authors of the Noise2Noise paper is to use a LeakyReLU (with negative slope of $\alpha = 0.1$) as non-linearity instead of a classical ReLU. This choice helps dealing with the vanishing gradient problem as the gradient of the LeakyReLU is never 0. Finally, they use *skip-connection*[3], i.e. concatenation of a reconstruction stage of the signal with the original downsampled image which enable features reuse and also deal with the vanishing gradient problem. At downsampling or upsampling block, they use two combinations of convolution-LeakyReLU layer. We tried 3 different versions of this network, the original one (referred to as *Noise2Noise* in the results), another version with 3 convolutions at each downsampling (and upsampling) step instead of 2 (referred to as *Deeper Noise2Noise*) and a version with only 3 downsampling block (we remove layers from Pool3 to Pool5 and from UP_5 to DEC_4_CONV2 in table 1, referred to as *Truncated Noise2Noise* in the results).

2.3 Noise2Noise HLS

Noise2Noise HLS, which stands for Noise2Noise Higher Latent Space, is a small modification of the Noise2Noise network whose goal is to help with our specific dataset. In the original network [1], the input data were 256 × 256 images from the ImageNet dataset. Thus, using 5 different max-pooling stage with a stride of 2 reduce the image representation to a 8 × 8 signal, i.e. the latent representation has 64 features. As you will see in 2, the resulting images from the baseline model are a bit blurry. We thought that this might be due to the fact that our input images are 32×32 images and therefore 5 pooling stages will reduce their sizes to 1 × 1 in the latent space. We though that the network doesn't have enough features to sharpen the edges of the reconstructed objects in the image. As a try to overcome this potential issue, we modify the stride used in the MaxPooling layer. Instead of dividing the image size by 2 in every downsampling block, we only reduce the number of pixel in the pixel by 2 in every downsampling block (i.e. we use a stride of (1,1) instead of a stride of (2,2)). Then, to compensate in the reconstruction, we used TransposedConvolution instead of a nearest neighbours upsampling followed by two convolutions.

Layer Name	Output size	Function
Input	$32 \times 32 \times 3$	
ENC_1_CONV1	$32 \times 32 \times 48$	Convolution 3×3
ENC_1_CONV2	$32 \times 32 \times 48$	Convolution 3×3
POOL1	$16 \times 16 \times 48$	MaxPool 2×2
ENC_2_CONV1	$16 \times 16 \times 48$	Convolution 3×3
ENC_2_CONV2	$16 \times 16 \times 48$	Convolution 3×3
POOL2	$8 \times 8 \times 48$	MaxPool 2×2
ENC_3_CONV1	$8 \times 8 \times 48$	Convolution 3×3
ENC_3_CONV2	$8 \times 8 \times 48$	Convolution 3×3
POOL3	$4 \times 4 \times 48$	MaxPool 2×2
ENC_4_CONV1	$4 \times 4 \times \times 48$	Convolution 3×3
ENC_4_CONV2	$4 \times 4 \times \times 48$	Convolution 3×3
POOL4	$2 \times 2 \times 48$	MaxPool 2×2
ENC_5_CONV1	$2 \times 2 \times 48$	Convolution 3×3
ENC_5_CONV2	$2 \times 2 \times 48$	Convolution 3×3
POOL5	$1 \times 1 \times 48$	MaxPool 2×2
MID_CONV1	$1 \times 1 \times 48$	Convolution 1×1
UP_5	$2 \times 2 \times 48$	Upsample 2×2
CONC_5	$2 \times 2 \times 96$	Concat. of UP_5 and POOL4
DEC_5_CONV1	$2 \times 2 \times 48$	Convolution 3×3
DEC_5_CONV2	$2 \times 2 \times 48$	Convolution 3×3
UP_4	$2 \times 4 \times 48$	Upsample 2×2
CONC_4	$2 \times 4 \times 96$	Concat. of UP_4 and POOL3
DEC_4_CONV1	$4 \times 4 \times 48$	Convolution 3×3
DEC_4_CONV2	$4 \times 4 \times 48$	Convolution 3×3
UP_3	$8 \times 8 \times 48$	Upsample 2×2
CONC_3	$8 \times 8 \times 96$	Concat. of UP_3 and POOL2
DEC_3_CONV1	$8 \times 8 \times 48$	Convolution 3×3
DEC_3_CONV2	$8 \times 8 \times 48$	Convolution 3×3
UP_2	$16 \times 16 \times 48$	Upsample 2×2
CONC_2	$16 \times 16 \times 96$	Concat. of UP_2 and POOL1
DEC_2_CONV1	$16 \times 16 \times 48$	Convolution 3×3
DEC_2_CONV2	$16 \times 16 \times 48$	Convolution 3×3
UP_1	$32 \times 32 \times 48$	Upsample 2×2
CONC_1	$32 \times 32 \times 51$	Concat. of UP_1 and Input
DEC_1_CONV1	$32 \times 32 \times 64$	Convolution 3×3
DEC_1_CONV2	$32 \times 32 \times 32$	Convolution 3×3
DEC_1_CONV2	$32 \times 32 \times 3$	Convolution 3×3

Table 1: *Every convolution is followd by a LeakyReLU except for the last one. MaxPool layers have a stride of 2 and upsampling are nearest neighbors.*

2.4 Other considered architectures

Finally, we considered two others and more elaborated architectures in terms of design. The first one is simply that, at each downsampling block, we use *Inception module*. These modules were introduced by Lehtinen et al. [4]. The goal is to let the network choose, at every downsampling stage, whether it prefers to use a 5×5, 3×3, 1×1 or an average pooling layer followed by a convolution. This layer is shown in fig. 1. Unfortunately, this architecture was quite slow to train (exceeding the required time limit to achieve higher PSNR) and therefore, we did not push the fine-tuning too far and we sticked with the other implementations.

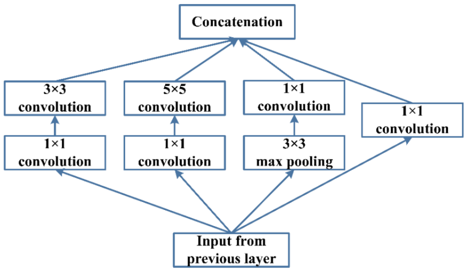

Figure 1: Inception module illustrated by Szegedy et al. [4]

Network	PSNR	Time	Epochs
Original Noise2Noise	25.5373	9min03	25
Deeper Noise2Noise	25.4773	9min22	20
Truncated Noise2Noise	25.5151	6min49	20
Noise2Noise HLS	25.3334	8min12	12
Noise2Noise FFT	25.5123	9min6	20
Noise2Noise Inception	-%	10%	5%

Table 2: *Results obtained on the the 900 last images of the validation set. Refer to section 3 for more information about the training parameters.*

The final considered architecture comes from the following idea. The noise introduced in the images could be seen as high frequency components in the images. We thought that maybe splitting the high and low frequency componenents of the image could be a good idea to help the model denoising the images. To do so, in each downsampling block, we compute the 2D Fourier Transform on each channel, split the central component (high frequency component) from the border-component (low-frequency ones). Then, we reconstruct the image using the Inverse Fast Fourier Transform on the low and high frequency channels separated. This way, the network has the possibility to handle the high and the low-frequency component as it wants.

3 Results

All the results were obtained by training models on a PC with a nVidia GTX 1080 GPU. The choice we made throughout the project was to explore a lot of different architecture rather than fine-tuning only one. The models were trained using the Adam optimizer [5] with parameter $\beta = (0.9, 0.99)$, the learning rate was implemented using a step learning rate scheduler which starts at 0.001 and decreases by a factor 0.1 every 10 steps. The loss used is the Mean Squared Error (MSE), batch size is 400 and the number of epochs depends on the model (to match the 10 minute constraints). The number of epochs is displayed on the result table for each of the model.

3.1 Baseline results

As we can see from section 3, the Noise2Noise-like architecture obtained the best scores overall. The structure proposed in [1] indeed gave the best score with a PSNR of 25.5373. In 2, we can see the result of the neural network given a noisy input. We can see that the denoising operation is indeed correctly handled by the network. Nevertheless, outputs look a bit blurry. This might be due to the fact that the network performs a lot of averaging operations between neighbours in order to remove the high-frequency noise.

Another interesting fact is that deeper network (with $\frac{675795}{488739} = 1.3827$ times more parameters than the original network, and a reduced number of epochs to fit the 10 minutes constraint) obtained a lower performance score despite its potentially increased representational power.

Finally, *Truncated Noise2Noise* achieves almost the same PSNR as the original network, with $488739 - 320211 = 168'528$ less parameters. We choose to only train for 20 epochs and not more because the PSNR starts to drop in the validation set (100 first images of the given test set). With this *truncated* network, we have a smaller training time, a smaller computational power required to perform either training or testing and a smaller memory space requirements. Therefore, depending on the platform on which we have to run the neural network, for example if we have to run it on a mobile phone, we can choose the *truncated* version as the computational load will be smaller.

3.2 Custom networks results

From section 3, we can see that the results for the different "custom" architectures we propose are not as good as the baseline results. The main issue with both the HLS network and the network using Inception modules is that the computational resources are higher, implying that we can do only half the number of epochs required to train the Noise2Noise network in the given time. For HLS, this is due to the fact that we perform the same number of convolutions but with larger image, leading to an increased computational time, whereas for the network with inception modules, this is due to the fact that we perform several convolutions in each downsampling layer. This is correlated with the fact that the network has 3 times more parameters than any other network. See section 4 for further ideas on the use of these two networks. Finally, we can also see that separating the high from the low frequencies doesn't help the network to outperform the original Noise2Noise network. However, the performance becomes really close to the one of the *truncated network*.

4 Limitations

The main limitation we faced in this project was the time constraint. In the beginning, we considered doing some data augmentation based on random crops and rotations of the training data. Unfortunately, the data augmentation factor is almost the same as the training time augmentation factor. This means that even with only 2 rotations of the training images and the original images, we have 3 times more training images. Therefore, without taking the time to transform the images at the beginning of the training into account, we already triple the training time. Without the time constraints, we could have tried to train the bigger network (HLS and Inception) with augmented data and see if their increased number of weights is able to fit the data better and outperform the regular Noise2Noise network.

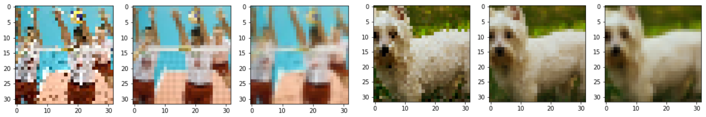

Figure 2: *Each triple represent a noisy input, its corresponding clean target and the output of our best network, i.e. the Noise2Noise network according to section 3.*

5 Conclusion

In this project, we managed to implement and assess the performances of the original Noise2Noise network, 3 different variants and 2 other networks with original ideas. The goal was to train a denoising autoencoder whose training data are pairs of noisy data. Although the best results were obtained using the architecture from the *Noise2Noise* paper[1], we were able to show that some of our simplified and original ideas were able to get results similar to the one obtained in the network.

References

[1] J. Lehtinen, J. Munkberg, J. Hasselgren, *et al.*, *Noise2noise: Learning image restoration without clean data*, 2018. DOI: 10.48550/ARXIV.1803.04189. [Online]. Available: https://arxiv.org/abs/1803.04189.

[2] N. Cohen and A. Shashua, *Inductive bias of deep convolutional networks through pooling geometry*, 2016. DOI: 10.48550/ARXIV.1605.06743. [Online]. Available: https://arxiv.org/abs/1605.06743.

[3] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. DOI: 10.48550/ARXIV.1512.03385. [Online]. Available: https://arxiv.org/abs/1512.03385.

[4] C. Szegedy, W. Liu, Y. Jia, *et al.*, *Going deeper with convolutions*, 2014. DOI: 10.48550/ARXIV.1409.4842. [Online]. Available: https://arxiv.org/abs/1409.4842.

[5] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. DOI: 10.48550/ARXIV.1412.6980. [Online]. Available: https://arxiv.org/abs/1412.6980.