EE-559 Deep Learning

Mini-project 1: Noise2Noise auto-encoder using PyTorch Framework

Tom MERY **297217**

Spring 2022

Contents

List of Figures

1 Introduction

In this mini-project, a Noise2Noise model is implemented using the standard PyTorch Framework. In this report, the experiments and their results that have been carried out during the building process of the architecture are presented. The theoretical background about Noise2Noise auto-encoder is available in the original paper [1].

The provided dataset for training is composed of two tensors of the size $50000 \times 3 \times 32 \times 32$ that corresponds to 50000 noisy pairs of images. Each of the 50000 pairs provided corresponds to downsampled, pixelated images. The goal is to train a network that uses these two tensors to denoise i.e., reduce the effects of downsampling on unseen images. A validation set of 1000 pairs of images of the same size is also provided. The performance is assessed on Peak Signal-to-Noise Ratio (PSNR) metric. It is known that a network with 8 convolutional layers achieves 24 dB PSNR on the validation data provided and an extremely simple benchmark achieves 23 dB.

2 Implementation

The implementation of the model described in section 4 is available in the file `model.py`. In this file a class `Model` that inherits from the base class `torch.nn.Module` (see `https://pytorch.org/docs/stable/generated/torch.nn.Module.html`) is defined. The architecture of the model as well as the optimizer and the loss used during training are defined as attributes of the class during the class initialization. The forward function is explicitly defined as a class method. The `Model` provides also 3 additional methods for loading, training and predicting. More comments about implementation are available directly in the source code.

3 Building process of the model

In this section, the process of building the model is described in chronological order for clarity purpose. This section is here to motivate the choices that have been made.

3.1 Initial model

In order to validate the implementation of the `Model` class, a first simple model is implemented.

3.1.1 Network architecture

The network of the first model is composed of two convolutional layers, with an arbitrary number of channels, each followed by ReLU activation, and with two transposed convolutional layers, one followed by ReLU activation, and the other one followed by sigmoid activation function in order to range the output in value between 0 and 1, which facilitates the computation of the PSNR metric. Weights and bias are initialized under the default method of PyTorch implementation (see `https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html`). Stride for each layer are set to 1 as the input image is already of a small dimension (32x32). Table 1 in appendix recaps the network architecture.

3.1.2 Loss function

The theoretical background from [1] shows that minimizing the expectation of the mean squared error (squared L2 norm) between an image with a noise ϵ and the same image with a noise δ is formally equivalent to minimizing the expectation of the mean squared error (squared L2 norm) between the image with a noise ϵ and the original clean image if ϵ and δ are different additives, unbiased, and independent noises. For this purpose the network is therefore trained under MSE loss function.

3.1.3 Training parameters

The model is trained using ADAM optimizer with a learning rate of 1e-3. The mini-batch-size is arbitrarily fixed to 50 and the model is trained over 10% of the full train set. With this configuration, this initial model already achieves 23.88 dB PSNR on the validation dataset and therefore the overall implementation is validated.

3.2 Deep Vs. Wide network

To choose the optimal number of layers for the network, multiple model with different number of convolution layers are trained over the same conditions. In order to make their performance comparable, when increasing the number of layers, the number of channels out are decrease such that every model keeps approximately the same number of trainable parameters. Table 2 in appendix recaps their structure, where the first half of the layers are convolution with kernel 3x3 followed by ReLU and the second half are transposed convolution with kernel 3x3 followed by ReLU except for the last one followed by sigmoid. The results in table 2 in appendix are most likely due to the fact that as the network gets deeper, the back propagated gradient becomes smaller and smaller in the first layer. One could try batch-normalization for deeper model but in this case, regarding PSNR after training a model with 4 layers seems to work pretty well. So to keep things simple, a model with 4 layers is kept.

3.3 Number of channels in hidden layers

In the initial model C_{out} of layers 1 and 2 are both equal. This choice has been done arbitrarily and it has been chosen to keep this structure as it was offering good results. However to increase the performance one could try to increase the number of channels. A grid search (from 10 to 400 channels with a step of 10) has been done for the number of channels in the hidden layer. Every models has been trained on 1 epoch over 10% of the train set with a mini-batch-size of 5. Figure 1 in appendix shows the results. One can see that from $C_{out} = 100$ the increase of performance becomes less significant while the computational power to train increases drastically. Also choosing a model with fewer parameters will prevent to a possible over-fitting when training over the full train set. Therefore $C_{out} = 100$ for the hidden layers of the final model is chosen.

3.4 Skip connections

Skip connection between layer 1 and layer 4 is added to the initial model and then both models are trained over 25 epochs under the same condition than described in section 3.1.3. After training the initial model achieves 24.43 dB PSNR while the model with skip connection achieves 24.69 dB PSNR. Figure 2 in appendix shows that skip connection makes the training loss decrease faster. Thus skip connection method will be kept for the final model.

3.5 Mini-Batch-Size and Number of Epochs

A grid search is done for every mini-batch-size between 1 and 50 such that 50000 is a multiple of the mini-batch-size. The initial model is trained on 1 epoch over the full train set 5 times for every mini-batch-size and the mean PSNR is computed for each. Figure 3 in appendix shows the results. Regarding the results a mini-batch-size of 8 is kept for the final model.

Similarly a grid search between 1 and 15 with a step of 1 is done for the number of epoch for the same model with a mini-batch-size of 8 on the full train set. Figure 4 in appendix shows the results. Regarding the results, a number of training epochs of 10 is kept for the final model since beyond this point the model starts to overfit.

4 Final model and results

The architecture of the final network is shown in Table 3. MSE loss function as well ADAM optimizer with a learning rate of 1e-3 are kept. To achieve the best performance (model saved in `bestmodel.pth`) the model is trained over 10 epochs on the full dataset with a mini-batch-size of 8. The model has taken 197.95s to train on Google Colab's GPU to finally achieves of performance of 25.32 dB PSNR on the validation dataset. Results of the denoising on 3 images taken randomly in the validation dataset are shown in figure 5 in the appendix.

References

[1] Jaakko Lehtinen; Jacob Munkberg; Jon Hasselgren; Samuli Laine; Tero Karras; Miika Aittala; Timo Aila. *Noise2Noise: Learning Image Restoration without Clean Data*. `https://arxiv.org/abs/1803.04189`. 29 Oct 2018.

Appendix

Layer	C_{in}	C_{out}	Function	Activation
1	3	32	Convolution kernel 3x3	ReLU
2	32	32	Convolution kernel 3x3	ReLU
3	32	32	Transposed Convolution kernel 3x3	ReLU
4	32	3	Transposed Convolution kernel 3x3	Sigmoid

Table 1: Architecture of the initial model

Nb of layers	Channel out of each layer	Nb of parameters	PSNR after training
4	32,32,32,3	20259	23.69 dB
6	26,22,22,22,26,3	20533	22.63 dB
8	20,20,18,18,18,20,20,3	20729	20.91 dB
10	18,16,16,16,16,16,16,16,18,3	20131	20.10 dB
12	16,16,14,14,14,14,14,14,14,16,16,3	20253	19.31 dB

Table 2: Architecture with different number of layers

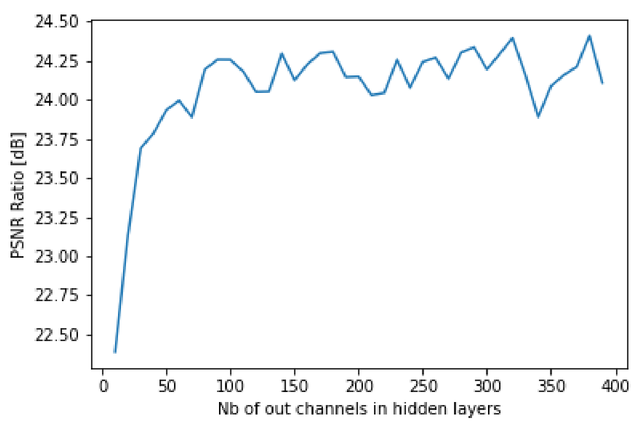

Figure 1: Performance of the model for different number of out channels in hidden layers

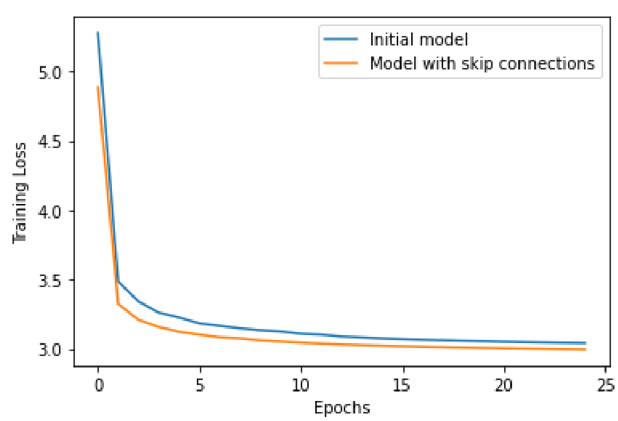

Figure 2: Comparison of model with and without skip connections

Layer	C_{in}	C_{out}	Function	Activation
1	3	100	Convolution kernel 3x3	ReLU
2	100	100	Convolution kernel 3x3	ReLU
3	100	100	Transposed Convolution kernel 3x3	ReLU
4	200 (skip connection)	3	Transposed Convolution kernel 3x3	Sigmoid

Table 3: Architecture of the final model

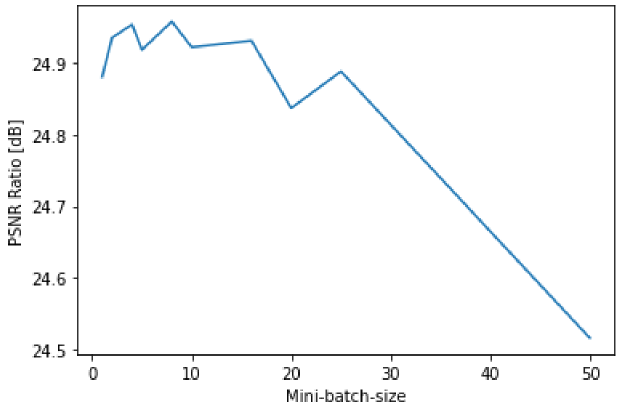

Figure 3: Mean performance of the model for different mini-batch-size

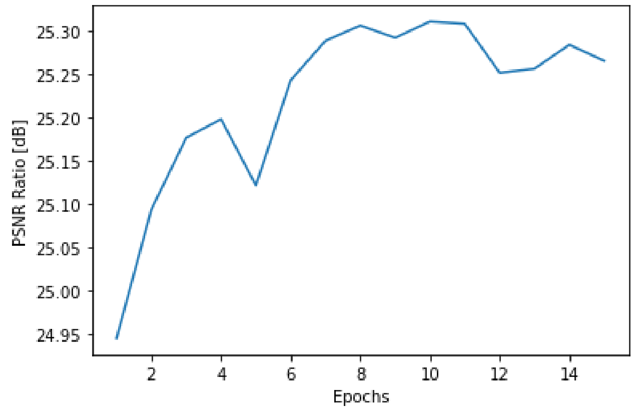

Figure 4: Mean performance of the model for different number of epoch

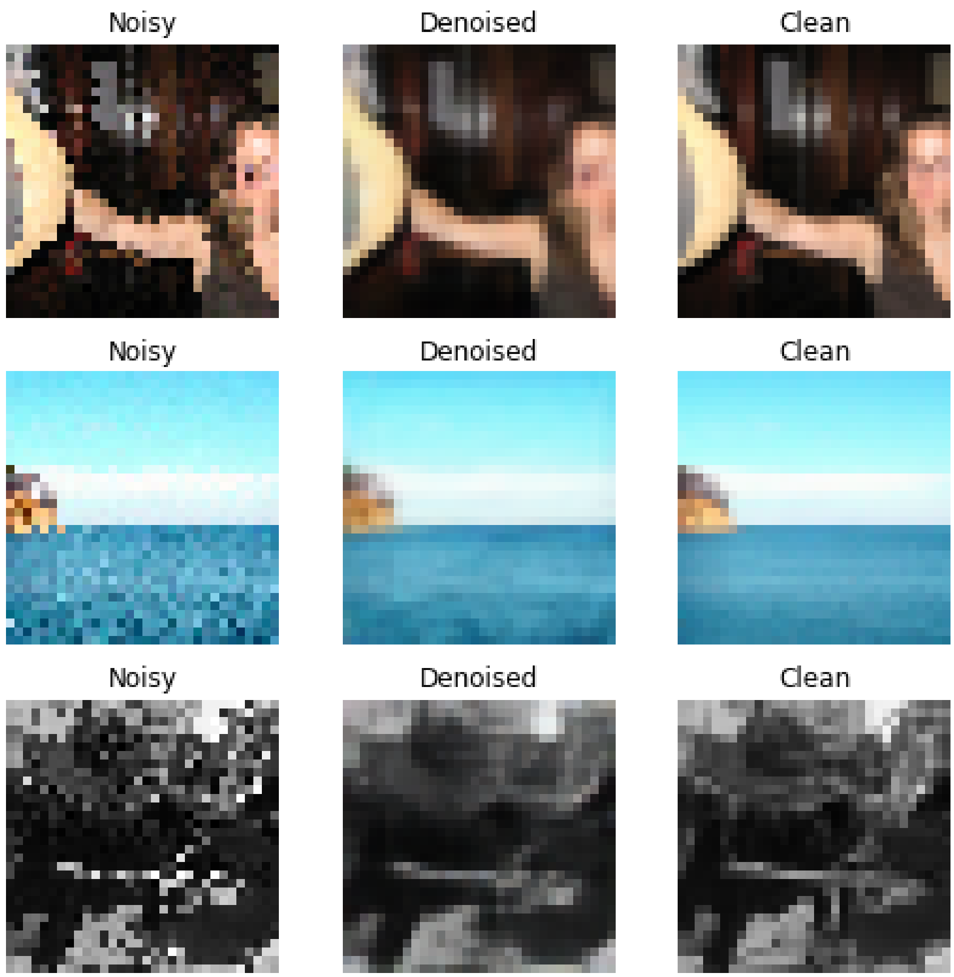

Figure 5: Results of the final model on random samples from the validation set