EPFL EE-559 Deep Learning

Miniproject 1

Author:
Nicola Antonio Santacroce, SCIPER : 286331

Report : 3 pages, Appendix : 4 pages
May 27, 2022

1 Introduction

This report details the methods and results from the first miniproject. In this project, a Noise2Noise deep learning model implementation is proposed, trained and tested.

The model is implemented using the Pytorch framework. The training data consists in 50000 pairs of 32 pixels x 32 pixels images with different noise applied to both. The validation data consists in 1000 images pairs with a noisy input and a clean target.

2 Architecture

The architecture of the model is identical to the one proposed in the reference Noise2Noise paper. It is a U-Net type model, which means that it is made of encoding and decoding convolutional blocks. When decoded, the signal is concatenated with its corresponding encoded state, giving it its U-shape.

the network architecture is provided on the table 1. The *Layer Name*, *Number*, *Channels out* and *Function* columns correspond the the table values reported in the reference paper. The *Size* column corresponds to the height and width values of the signal at the output of each layer.

The layers have been implemented in blocks as some steps were repetitive. These blocks can be seen on the last column of the table 1 and the implemented functions for each block can also be retrieved from the table. Note that in practice, concatenation has not been implemented inside blocks, but directly in the forward pass.

All Convolutions have the following parameter set : (kernel size : 3, padding : 1, stride : 1, dilation : 1). All Maxpoolings have been performed with a kernel size : 2. All Upsamplings

are performed with scale : 2 and the method : nearest neighbour. All convolution blocks are activated using a leaky ReLU of negative slope : 0.1 except for the last one which has a linear activation.

The optimizer chosen to train this model is Adam, like described in the reference paper.

LAYER NAME	NUMBER	CHANNELS OUT	SIZE	FUNCTION	BLOCK NAME
INPUT		3	32		
ENC_CONV	0	48	32	Convolution	enc_conv0
ENC_CONV	1	48	32	Convolution	enc_conv_pool1
POOL	1	48	16	Maxpool	
ENC_CONV	2	48	16	Convolution	enc_conv_pool2
POOL	2	48	8	Maxpool	
ENC_CONV	3	48	8	Convolution	enc_conv_pool3
POOL	3	48	4	Maxpool	
ENC_CONV	4	48	4	Convolution	enc_conv_pool4
POOL	4	48	2	Maxpool	
ENC_CONV	5	48	2	Convolution	enc_conv_pool5
POOL	5	48	1	Maxpool	
ENC_CONV	6	48	1	Convolution	enc_conv6
UPSAMPLE	5	48	2	Upsample	upsample5
CONCAT	5	96	2	Concatenate w/ POOL 4	
DEC_CONV	5A	96	2	Convolution	dec_conv_conv_up4
DEC_CONV	5B	96	2	Convolution	
UPSAMPLE	4	96	4	Upsample	
CONCAT	4	144	4	Concatenate w/ POOL 3	
DEC_CONV	4A	96	4	Convolution	dec_conv_conv_up3
DEC_CONV	4B	96	4	Convolution	
UPSAMPLE	3	96	8	Upsample	
CONCAT	3	144	8	Concatenate w/ POOL 2	
DEC_CONV	3A	96	8	Convolution	dec_conv_conv_up2
DEC_CONV	3B	96	8	Convolution	
UPSAMPLE	2	96	16	Upsample	
CONCAT	2	144	16	Concatenate w/ POOL 1	
DEC_CONV	2A	96	16	Convolution	dec_conv_conv_up1
DEC_CONV	2B	96	16	Convolution	
UPSAMPLE	1	96	32	Upsample	
CONCAT	1	99	32	Concatenate w/ INPUT	
DEC_CONV	1A	64	32	Convolution	dec_conv1A
DEC_CONV	1B	32	32	Convolution	dec_conv1B
DEC_CONV	1C	3	32	Convolution (linear act)	dec_conv1A

Table 1: Model architecture and implementation

3 Data Augmentation

The Model class in the model.py file contains an additional method called *augment_data()* that allows to augment data before using it for training. Augmenting data allows to mitigate the risk of overfitting as the model can't learn the dataset directly. It also increases the validation performance as the model generalizes more by seeing randomly created variants of each document at each epoch.

The implemented data augmentation consists in three transformations. The first one is the addition of gaussian noise centered in zero and with a standard deviation of 10% of that of the batch. Initially, both input and target where augmented with noise, but tests showed that augmenting inputs only provided slightly better results. It has thus been decided to add noise to the inputs only. Then, random transpose and random rotation (with a step of 90 degrees) are applied to both input and target.

4 Parameters Tuning

The model has been trained using various configurations of loss functions and hyperparameters, whose values can be seen on the table 2. All trials have been performed via grid search, with a batch size equal to 100 on 1000 augmented image pairs for 100 epochs. Training progressions have been visualised using the Weight and Biases tools The results of the grid search indicated a preferred configuration of (Learning Rate : 1e-3, Betas : (0.9, 0.99), Epsilon : 1e-8 and Loss function : MSE Loss). The train losses and validation PSNR scores of the four best models can be seen on the figures 1 and 2 in the appendix A section.

Parameter	Learning Rate	Betas	Epsilon	Loss
Value	1e-3, 1e-4, 1e-5	(0, 0.99), (0.9, 0.99), (0.9, 0.999)	1e-7, 1e-8, 1e-9	MSELoss, L1Loss

Table 2: Hyperparameters for grid search

5 Final Model

Additional tests with the full training data, batch sizes of 10, 25 and 100 and learning rate tuning resulted in the following final choice for the training framework (note that decreasing the batch size allowed to decrease the learning rate too) : Learning rate : 5e-5, betas : (0.9, 0.99), Epsilon 1e-8, loss : MSE, batch size : 10.

This configuration resulted in the best model, achieving a 25.757 PSNR after 100 epochs Note that with this configuration, a plateau is reached at roughly 40 epochs but the models still improves without overfitting signs. The training of the model can be visualised on the figures in the appendix B. The state of the model has been saved in the *bestmodel.pth* file. Predictions on the first three validation images are displayed in the appendix C.

A Appendix : grid search best results

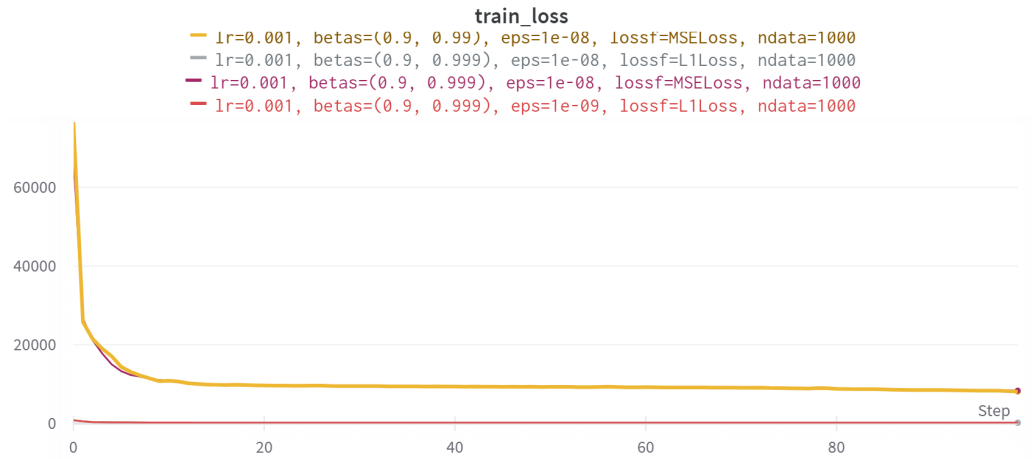

Figure 1: Train loss

Figure 2: Validation PSNR

B Appendix : Best model training

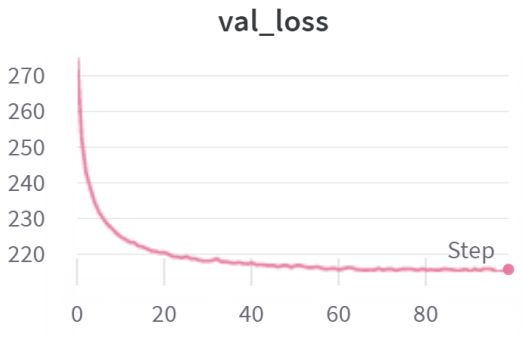

Figure 3: Validation loss

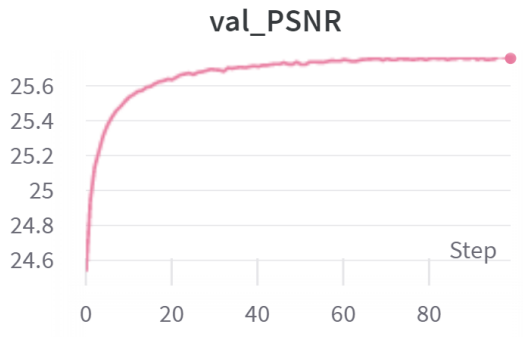

Figure 4: validation PSNR

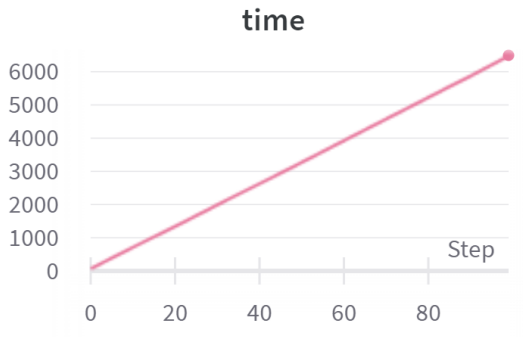

Figure 5: Training time

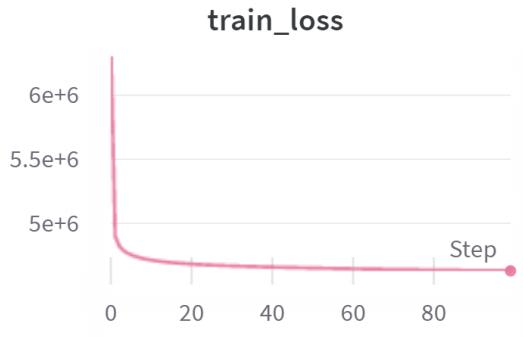

Figure 6: Training loss

C Appendix : Best model output examples

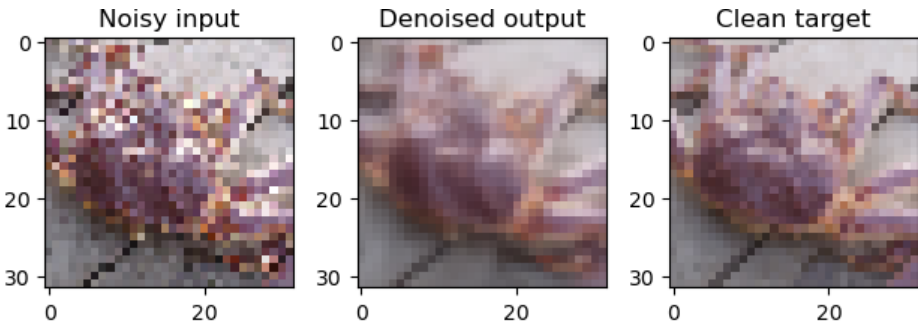

Figure 7: validation image 1

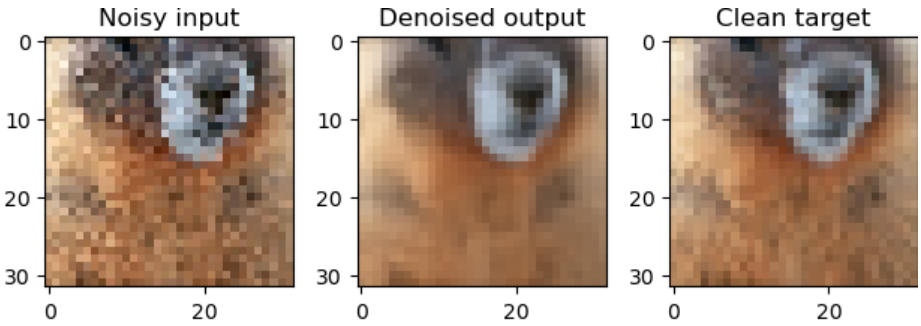

Figure 8: validation image 2

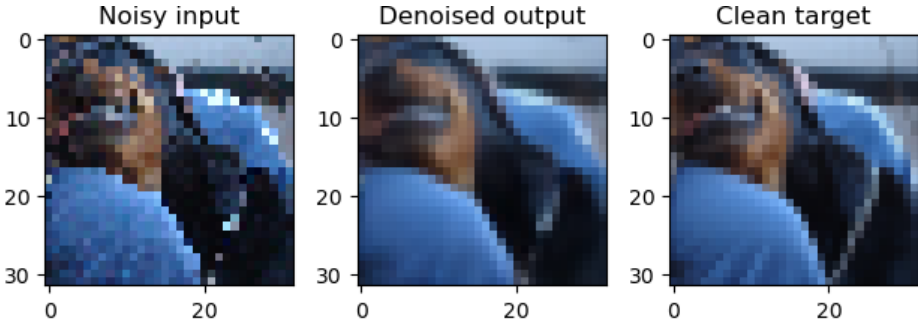

Figure 9: validation image 3

D Appendix : GAN to train a Noise2Noise model

Using the U-Net as a generator and a simple classifier as a discriminator model allows to train both models in an adversarial way. This method has been implemented, based on the pix2pix paper. It provided good results, but only slightly better than the U-Net alone as the discriminator trained way more rapidly than the generator and thus the adversarial loss did not enhance training except for really early steps. It has thus been chosen to discard this method as it required more time to train and an additional best model save for the discriminator. The implementation can be found in the others/gan_architecture folder.