Project 1: Noise2Noise model

Estelle CHABANEL 284197, Clara LE DRAOULEC 287280, Hugo WITZ 284336
EE-559 - Deep Learning, EPFL, Switzerland

I. INTRODUCTION

Some researchers have shown that image denoising model could be built and trained directly from noisy images [1]. The goal of this project is to implement and tune an architecture solving this same task.
Starting from a dataset of pairs of blurry images, several networks have been implemented. Their architecture and choice of parameters will be described along with their performances on a test dataset.

II. DATA PREPROCESSING AND AUGMENTATION

Two datasets are provided for this project: a training one containing 50 000 pairs of noisy images, and a testing one with 1 000 pairs of noisy-clean images. All the images have size HxW=32x32 pixels and C=3 channels corresponding to RGB images. As the neural networks will be implemented using the PyTorch framework, the datasets are loaded as tensors of float64, using a customed Dataset class and *DataLoader*, to process the images by batch.

Moreover, to improve the performance of our models, some data augmentations are performed. It is implemented using *Kornia*, an open source module that can perform data augmentation of images as tensors [2]. For each pair of images of the training set, another pair is created: the images are flipped horizontally and/or vertically both with a probability of 0.5, then, the images are rotated from a random identical angle chosen in the range [-180, 180]. A pair of augmented images of pair 1 is shown in figure 2. Performance of the chosen architecture will be tested both on the augmented and original dataset.

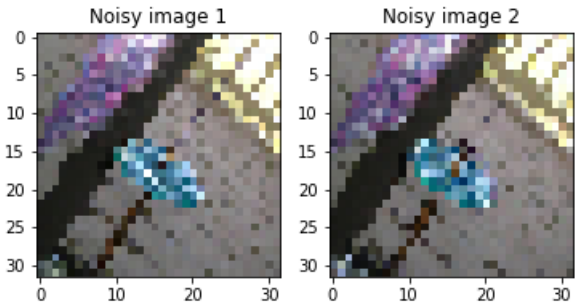

Figure 1: Pair of images of the training dataset

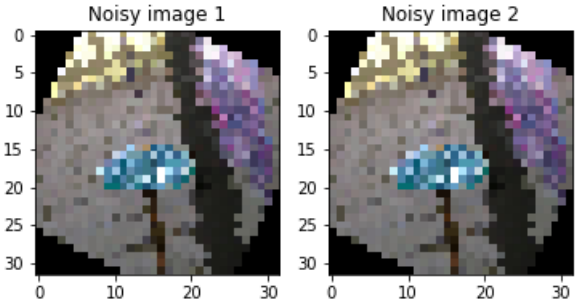

Figure 2: Pair of augmented images

III. AUTO-ENCODER MODELS

A. Base model

In order to construct our model, we first take inspiration on the Noise2Noise model provided in the paper [1]. This one is a rather complex one, with a general auto-encoder structure composed of several layers of convolution followed by a LeakyReLU.

The implemented base model is simplified in comparison with this structure, with 3 encoding blocks and 3 decoding blocks composed of convolution and ReLU activation. The model is supposed to learn the best "compression" and "decompression" method for the input images by reducing their HxW dimensions, while increasing their number of channels C in the encoder part, and re-increasing H and W while decreasing C in the decoder part.
In order to reduce the dimensions of the input faster, a maxpooling layer follows the

first convolution layer, and a corresponding upsampling layer will be used in the last convolution of the decoding part. After each decoding sequence, the output of the corresponding encoding sequence is concatenated to the current output before going through the next layer. Finally, there is a final sequence, composed of two convolutions in order to get back to the initial dimension of the images, followed by a ReLU and a sigmoid activation function respectively. A scheme of the general structure of the models is also given in figure 6.

The table I gathers the results of the base model, along with some variants derived from it. All results are obtained with an ADAM optimizer, with initial learning rate 0.001 and weight decay 10^{-8}. As visible, the PSNR score (24.23 dB) obtained on the testing set, in 50 epochs of training is already quite satisfying and indicates that this structure is quite efficient.

B. Variants of the base model

Variant 1: The sigmoid is generally preferred for classification task, in order to output a probability. The first variant consist of replacing the sigmoid last activation function by a ReLU. As this function outputs the positive part of the argument, it makes sense to use this one for images, the training images pixels having values in the range [0,1].

Variant 2: For the decoding part, it is very common to use transposed convolutions. These last ones consist in changing the order of the dimensions in the convolution operation matrix in order to output another "version" of the input [3]. Hence, in this model, convolutions are used to "compress" the input images and transposed convolutions "decompress" them into a better representation (a non blurry one in this case).

Variant 3: Some auto-encoder models also use "double convolution layers". This consists in adding an extra convolution and ReLU activation function in each encoding/decoding sequences.

Variant 4: As explained before, maxpooling layers can be used as another method to "compress" the input data in the encoding part. This variant consist of adding a maxpooling/upsampling layer respectively at the end of each encoding/decoding block.

	PSNR score on test set [dB]
Base model	24.2284
Variant 1	24.4905
Variant 2	24.5029
Variant 3	24.52291
Variant 4	24.3921

Table I: Final results of the tested models

Results in the table I show the efficiency of some variants. Changing the last activation function to a ReLU clearly improved the model. Other changes are less drastic, changing the convolution to transposed convolution in the decoding blocks seems to have a good impact without impacting the training time. Adding double convolution sequences presents the best final PSNR score (24.53 dB) but slightly increases the training time. Adding more maxpooling/upsampling layers does not seem to improve the model.

Moreover, the output images of each model can be seen on figure 3. All models are capable of recovering the main features of the picture. The distinguishable differences are on the very small details, like the bird's beak which is almost never recognizable, or the thin tree branch that crosses the picture. Variants seem to do a little bit better on this last detail but it is quite subjective. This could be a point to improve in the model, maybe via specific data augmentation.

Finally, evolution of the loss and the PSNR score at the end of each epoch can be seen for all models in figure 5 in Appendix. Curves are very similar, with better final scores for the variants.
Taking all of this into account, variant 3 will be

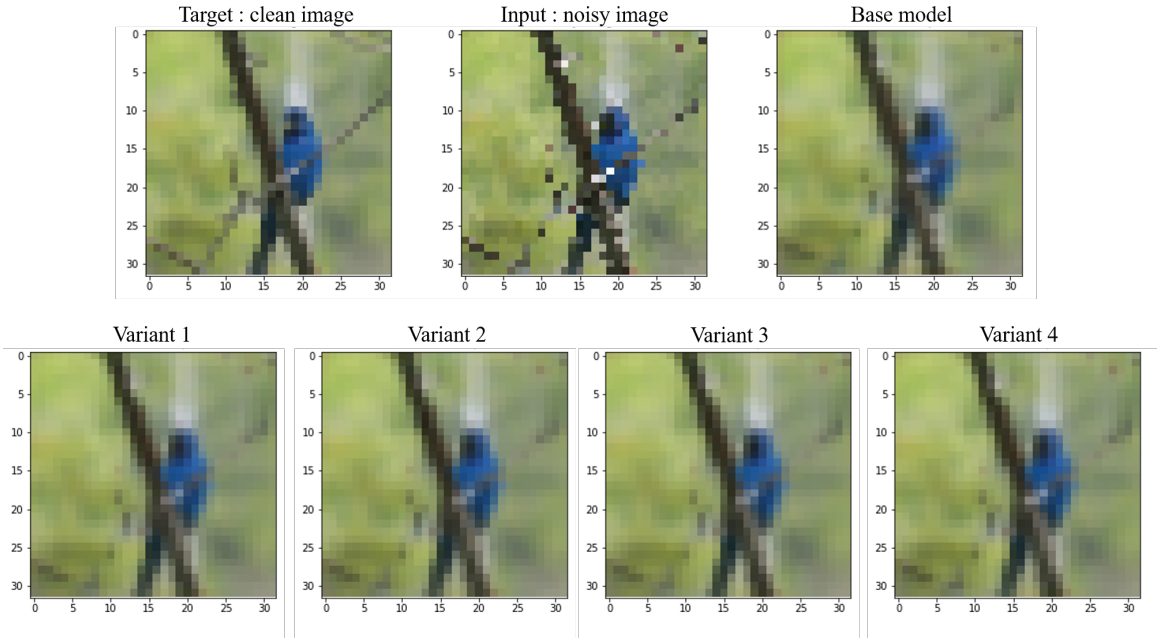

Figure 3: Outputs of the different tested

the one preferred and studied further in the next section.

Other attempts have been realized, like adding some convolution block or changing the activation functions... This first attempt was very efficient in term of output, but very low in term of training time. Other attempts were either too slow, or less efficient.

IV. Training and tuning

Models are trained with the Mean-Squared Error loss function that computes the error between the current transformed noisy images $\mathbf{x}$ and their corresponding "targets" $\mathbf{y}$, which are in our case, also noisy.

The model is applied by batch of size 512 to reduce the training time, thus the gradient is computed and reduced by back-propagation after each batch. Several optimizers are tested to do so, SGD and ADAM. Several initial learning rates and weight decays are tried to select the most efficient one as well.

Moreover, a good metric for the quantification of the sharpness of an image being the Peak Signal-to-Noise Ratio (PSNR) (equation (1)), this metric is computed on a new set of images, the 500 first images of the test set, at the end of each epoch. We then use a scheduler (*ReduceLROnPlateau*) that reduces the learning rate according to the improvement of the PSNR. With a *patience* value of 2, the scheduler reduces the learning rate by a factor 0.5 if the PSNR has not improve after 3 consecutive epochs.

$$\text{PSNR} = -10 * \log\left(\text{MSE} + 10^{-8}\right) \qquad (1)$$

The 500 other pairs of images of the testing dataset are used to evaluate the final performance of our model, after training.

The final performance of the model Variant 3 are gathered in the table II for different parameters, on 50 epochs of training. To reduce the running time, 28 epochs can be train in less than 10 minutes on **GoogleColab's** GPU and already gives satisfying results, above 24.4.

V. Conclusion

In this study a Noise2Noise model has been built, learning the task of image denoising from pairs of noisy images. After testing of several architectures and tuning of hyperparameters, the final model reached a PSNR score of 24.56 dB. Visualizing the output denoised image revealed the efficiency of the model even if this one could still be improved by performing more specific image augmentation or adding more encoding and decoding layers.

REFERENCES

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

[2] *Kornia Documentation for data augmentation*, https://kornia.readthedocs.io/en/latest/augmentation.module.html#

[3] *Transposed Convolutions explained with... MS Excel!*, Thom Lane, November 2018, https://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8

APPENDIX

Auto-encoder models

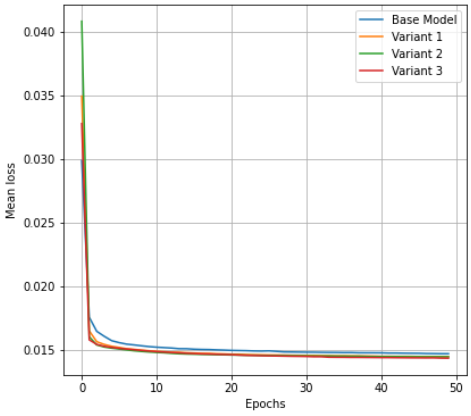

Figure 4: Evolution of the loss during training of models tested

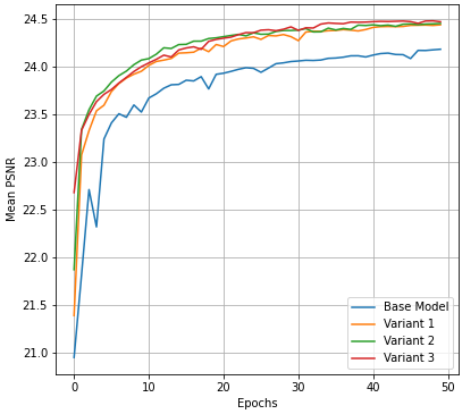

Figure 5: Evolution of the PSNR score during training of models tested

Training and tuning

data aug.°	**optimizer**	**initial l.r.**	**weight decay**	**batch_size**	**PSNR dB**
No	SGD	0.01	10^{-8}	512	22.2017
No	SGD	0.001	10^{-8}	512	17.7417
No	ADAM	0.0001	10^{-8}	512	23.9831
No	ADAM	0.001	10^{-8}	512	24.5189
No	ADAM	0.005	10^{-8}	512	23.7006
No	ADAM	0.01	10^{-8}	512	22.5935
No	ADAM	0.001	10^{-5}	512	24.3693
No	ADAM	0.001	10^{-10}	512	24.5193
Yes	ADAM	0.001	10^{-10}	512	24.5606

Table II: Parameters tuning for model Variant 3

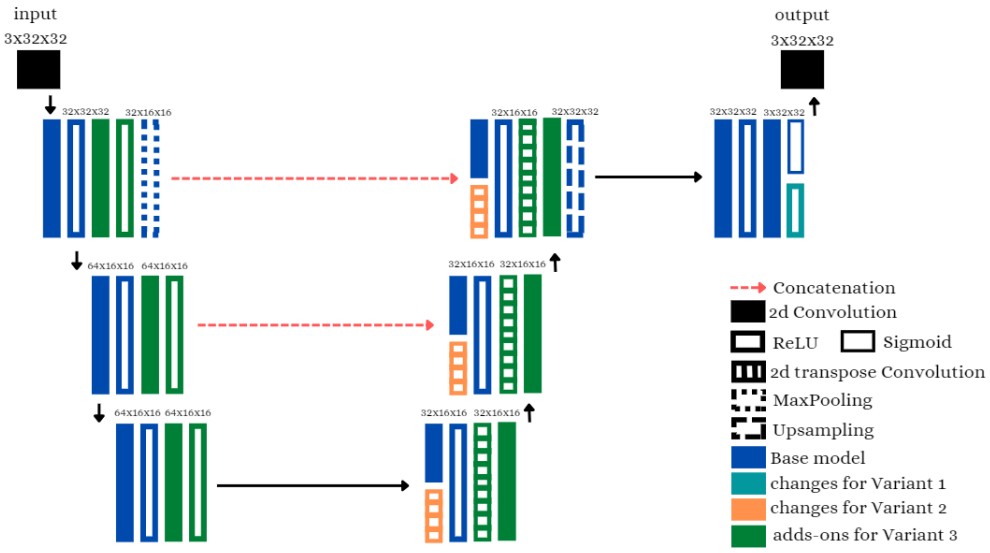

Figure 6: Structures of the tested models