Project 1: Noise2Noise using the standard PyTorch framework.

Manon Béchaz, Thomas Castiglione, Emilien Seiler
EE-559 Project 1, EPFL 2022

***Abstract*—This report presents deep learning models that use Noise2Noise approach to denoise images. The goal is to denoise an image by training models with noisy images only. Among multiples models, we obtained the best performance with an architecture derived from known RedNet networks. The final model was trained on a set of 50000 noisy pairs of images with a training time limit of 10 minutes on a GPU, and achieved a PSNR of 25.42 dB on the validation set.**

I. Introduction

Image denoising is a well studied problem in many applications. The objective of this first part of the project is to implement, train, and fine-tune a model to solve Noise2Noise [1] type problems. The idea behind Noise2Noise is to denoise images with a model trained only on noisy versions of images. This comes valuable in a lot of use cases where one does not have access to clean images for training but only to multiple noisy versions of the same image.

Considering two samples with independent, additive and unbiased noises ϵ and δ, one can derive that:

$$\mathbb{E}\left[\|\phi(X+\epsilon;\theta)-(X+\delta)\|^2\right] = \mathbb{E}\left[\|\phi(X+\epsilon;\theta)-X\|^2\right]+\mathbb{E}\left[\|\delta\|^2\right], \quad (1)$$

meaning that the expectation of the squared L2-norm between the transformation of a noisy image and another noisy version of the image is equal to the expectation of the squared L2-norm between the transformation of a noisy image and the clean version of the image plus the L2-norm of the second noise which does not depend on the model. Hence, minimizing the squared L2-error between the transformation of a noisy image and another noisy version of it is the same as minimizing the squared L2-error between a transformation of a noisy image and its clean version:

$$\underset{\theta}{\operatorname{argmin}}\,\mathbb{E}\left[\|\phi(X+\epsilon;\theta)-(X+\delta)\|^2\right] = \underset{\theta}{\operatorname{argmin}}\,\mathbb{E}\left[\|\phi(X+\epsilon;\theta)-X\|^2\right]. \quad (2)$$

These equations allow to train denoising models on noisy images only as long as some hypothesis are met: one should have at least 2 versions of the same image, with different and independent, additive, and unbiased noises.

II. Methodology

A. Dataset

We are provided with a training dataset of 50'000 pairs of downsampled noisy images, as well as a validation dataset of 1'000 pairs of noisy and clean images. This second dataset is used to track the progress and the performance of differents models. Note that the noisy images have unknown noise characteristics, but we assume the hypothesis mentioned above are met. An example of a pair of noisy images is given in Figure 1.

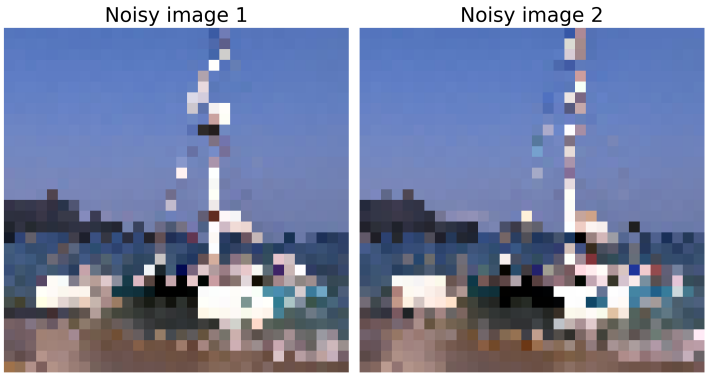

Fig. 1. Example of noisy pair image.

B. Models

To choose our model architecture, we decided to search in literature for models having a good potential at solving our problem and adapt them to it. The main architectures used for denoising are slightly different denoising autoencoder models, as the one presented in the reference paper [1]. We came up with 3 base models: a RedNet model (deep Residual Encoder-Decoder Network), the Unet from the reference paper, and a Swin Transformer model. Each model main characteristics are explained below.

1) RedNet: The residual network enables us to increase the network depth without increasing the number of parameters. Adding skipconnections between encoder and decoder helps the gradients reach all the layers of a network more easily and allows information related to finer details in the early stage of the encoder to be fully utilized in the decoder [2].

2) UNet: The network consists of a contracting path and an expansive path, which gives it the u-shaped architecture. The contracting path is a typical convolutional network that consists of repeated application of convolutions, each followed by a rectified linear unit (ReLU) and a max pooling operation. During the contraction, the spatial information is reduced while feature information is increased. The expansive pathway combines the feature and spatial information through a sequence of up-convolutions and concatenations with high-resolution features from the contracting path [3].

3) Swin Transformer: The Swin Transformer model is an attention-based model adapted for computer vision. The main characteristic of the model is its use of shifted windows: it starts by computing self-attention on small non-overlapping patches of 4x4 pixels, and gradually merges neighboring patches along the Transformer layers. This enables it to benefit from both advantages of CNN (process large scale image with local computation) and Transformers (account

for long-range interactions). Although long-range interactions may not be intuitively useful for denoising, we found a recent implementation of of a Swin Transformer (SwinIR) outperforming state-of-the-art models on benchmark denoising datasets [4].

C. Training

The training of the networks is both computationally and time-consuming. In this part, we discuss the different choices made for training.

1) Training time: The main bottleneck at this step was the GPU walltime allowed of 10 minutes maximum for training on the whole dataset. All models complexities had to be restrained in order to fit in this maximum time of training. Note that the results presented in the rest of this report were therefore obtained after 10 minutes of training on GPU only.

2) Epochs: The number of epochs, reported in Table I was adapted to the model complexity to ensure a 10 minutes training.

RedNet	Unet	SwinIR
40	8	4

TABLE I
Number of epoch corresponding to a 10 minutes training.

3) Pre-processing: Images are used without any transformation. Normalization from [0, 255] to [0, 1] was tried but did not improve the results significantly (note however that the images are normalized between [0, 1] for the PSNR calculation).

4) Loss function: The Mean Squared Error (MSE) loss function was used to learn the task. This common loss function is given by:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - Y_i)^2, \tag{3}$$

where y is the output of the model and Y is the target.

5) Optimizer: As a stochastic optimizer, we chose Adam. [5] This method is computationally efficient, has little memory requirements and is well suited for problems that are large in terms of data and/or parameters. We set the learning rate at 0.001 after cross-validation.

6) Metrics: The PSNR (Peak Signal to Noise Ratio) was used to track the progresses of the training and evaluate models performances. The latter can be expressed as:

$$\text{PSNR} = -10 * log_{10}(MSE + 10^{-8}). \tag{4}$$

PSNR is commonly used to quantify reconstruction quality for images. Note that the calculation of the PSNR is done on the validation set with a batch size of 1.

7) Batch size: The batch size was set to 256 after cross-validation for the training of the model.

8) Cross validation: The provided training dataset was split into a training set and a testing set with a split ratio of 0.1 in order to track overfitting and generalization of different models. At each epoch the loss over the test set is calculated.

9) Hyperparameters: 3-fold Cross Validation on different hyperparameters was used to select the models with the best performances. For each model, we used a Bayesian Search over chosen sets of value for each hyperparameter, trained each for about 5 minutes only for computation time reasons. The test loss was then averaged over the 3 runs. For each of the three architectures, we eventually chose the model with the lowest test loss.

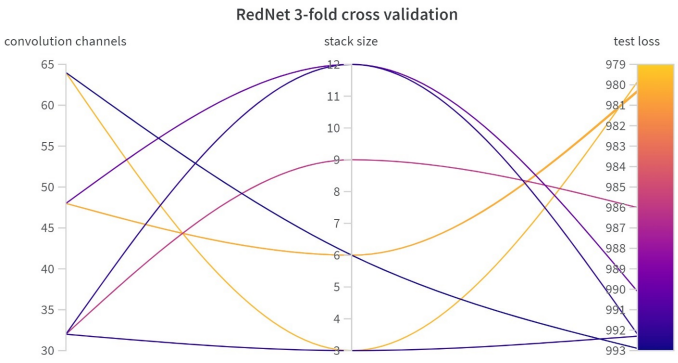

Fig. 2. ResNet optimization hyperparameter

Figure 2 presents an example of cross-validation for the redNet. Note that when several hyperparameter configurations yielded nearly identical losses, we chose the one with the smallest number of parameters in order to train on more epoch. The best hyperparameter sets of each model are respectively:

- ***RedNet***: number of channel=64, kernel size=3, stack size=3, number of parameter=70k
- ***Unet***: number of channel=64, encoder and decoder depths= 3, number of parameter=1.2M
- ***Swin Transfomer***: patch size=4, patch embedding dimension=32, Transformer layers depths=[4,4,4], number of attention heads in each layer=[4,4,4], number of parameter=200k

10) Implementation: The neural networks are implemented using the PyTorch framework, associated with Torchvision. For hyperparameter optimization the Weights & Biases experiment tracking tool was used [6].

The training procedure is performed using a GPU on Google Colab, since it is too heavy for a CPU.

The 3 optimally fine-tuned models were eventually compared by training them for 10 minutes and evaluating them on the whole provided validation set.

III. Results

Figure 3 presents the evolution of the loss during the training. The training loss decreases over the time, thus indicating that the models indeed learn the task. Due to the absence of increase of the test loss, it is safe to say that the models do not overfit the training set after 10 minutes of training.

To evaluate the performance of each model the average PSNR over the whole validation set was calculated at each epoch.

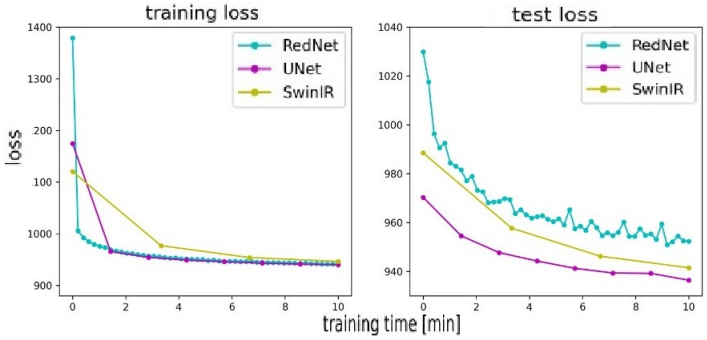

Fig. 3. Train and test loss

Results are provided in Figure 4. The PSNR increases with time for the three models, reaching a value of 25.4 dB after 10 minutes of training.

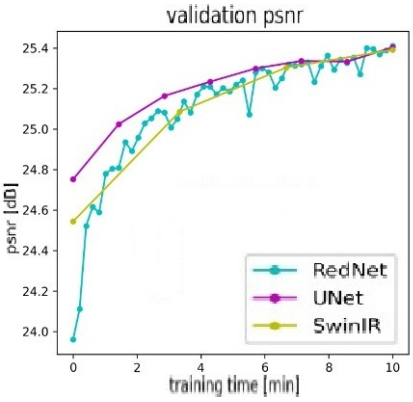

Fig. 4. Evolution of average psnr over the validation set

The final PSNR of the models are provided in Table II. Even if the redNet seems to outperform the other networks on the validation set, we observe that the performances of the model are comparable.

RedNet	Unet	SwinIR
25.42 dB	25.40 dB	25.39 dB

TABLE II
Mean PSNR of the three models on the validation set, after 10 minutes of training.

Eventually, a few examples of noisy, denoised and clean images obtained with the redNet architecture are given in Figure 5. All three images are correctly denoised, and the output images are very close to the clean/reference ones.

IV. Discussion

A. Unsuccessful attempts

Various other methods for improving performance have been tested, however none have proven to be successful. These include, but are not limited to :

- Mean Absolute Error (MAE) loss, that did not improve the result obtained with MSE (note that PSNR is a function of MSE so this result can be expected with a evaluation based on PSNR)
- Dropout ($p = 0.2$), tried for the RedNet and the Unet which did not enhance the result either.

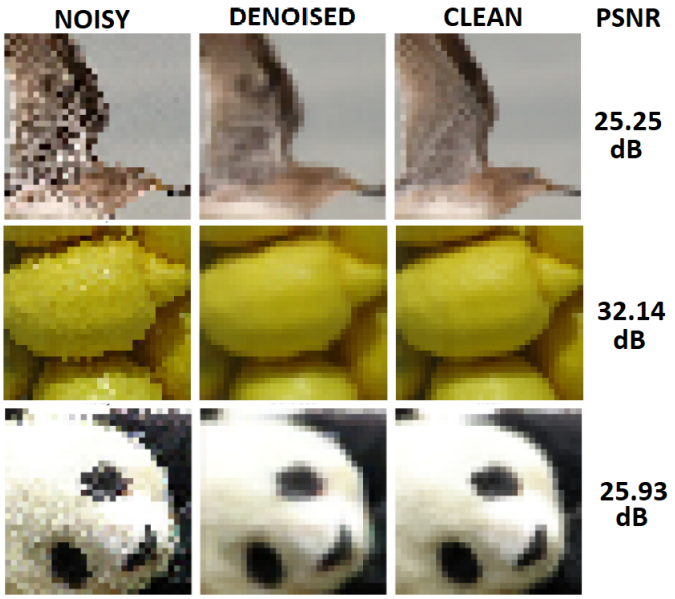

Fig. 5. Exemple of denoised image with the RedNet

- Batch normalization, tested for the RedNet and the Unet by adding batch normalization layers in each block before the activation function, without any notable improvement.

B. Future Improvements

- Data augmentation such as the flipping/croping of the training images can increase the size of the dataset and help model generalization. We decided not to use it because of the time limitation for training.

V. Conclusion

The three different architectures result in very good results. It is however worth mentionning the large difference in the number of trainable parameters between the two convolutional autoencoders (70k for the redNet vs 1.2M for the Unet). The more complex Transformer architecture of SwinIR also provide us with good results. With larger training time, better results could undeniably be reached.

References

[1] J. Lehtinen, J. Munkberg, J. Hasselgren, S. Laine, T. Karras, M. Aittala, and T. Aila, "Noise2noise: Learning image restoration without clean data," 2018. [Online]. Available: https://arxiv.org/abs/1803.04189
[2] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015. [Online]. Available: http://arxiv.org/abs/1505.04597
[3] J. Jiang, L. Zheng, F. Luo, and Z. Zhang, "Rednet: Residual encoder-decoder network for indoor rgb-d semantic segmentation," 2018. [Online]. Available: https://arxiv.org/abs/1806.01054
[4] J. Liang, J. Cao, G. Sun, K. Zhang, L. Van Gool, and R. Timofte, "Swinir: Image restoration using swin transformer," 2021. [Online]. Available: https://arxiv.org/abs/2108.10257
[5] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: https://arxiv.org/abs/1412.6980
[6] L. Biewald, "Experiment tracking with weights and biases," 2020, software available from wandb.com. [Online]. Available: https://www.wandb.com/