Comparison of Noise2Noise models implemented using the PyTorch framework

Luca Zampierin (343174), Saverio Nasturzio (299307), Zhecho Mitev (323387)

[EE-559] Deep Learning, EPFL, Switzerland

***Abstract*—Image denoising has emerged as a valuable pre-processing step when working with images, either for aesthetic purposes or to facilitate down-stream tasks. In particular, denoising images without having the clean version as a reference has developed as area of research in last few years. In this paper we present a comprehensive comparative study of different Noise2Noise approaches to image denoising. We present our best-performing model which achieved a PSNR value of approximately 25.83 db on a test set.**

I. Introduction

Image denoising is a fundamental area of research in the image processing and computer vision fields. Its main purpose is the development of algorithms able to restore the original quality of a degraded image. The practical application of this task is extremely relevant, as all imaging systems rely on the availability of information-carrying clean images. On top of that, image denoising is useful to improve the user experience when using image-related tools [1].

Over the years, many approaches have been proposed in the field of image denoising. The interesetd reader can refer to [1] for a well-structured summary of image denoising algorithms. With the advent of Deep Neural Networks (DNNs), and in particular Convolutional Neural Networks (CNNs), able to automatically learn the hidden mapping between the noisy and the clean versions of an image, the denoising performance has improved drastically. Zhang *et al.* [2] proposed the DnCNN network consisting of stacked convolutional blocks, that is, a convolutional layer followed by batch normalization and ReLu activation. The performance achieved by this model attracted attentions from researchers. Image denoising can be broadly categorized as either residual structures, such as DnCNN, or encoder-decoder structures, such as U-Net [3].

While the aforementioned algorithms have been trained and tested on noisy-clean image pairs, Lehtinen *et al.* [4] propose a Noise2Noise model, that is, an image denoising network trained using noisy references instead of a clean version of the image. In particular, the authors show that by adjusting the loss function used for training, the RED30 model they used was able to remove additive Gaussian, Poisson, Bernoulli, Brown, and Random-values impulse noise.

In this project, we propose a Noise2Noise denoiser network trained using 50000 noisy pairs of images that achieves a performance of approximately 25.83 db on unseen test data after training for less than 20 minutes on one NVIDIA GeForce GTX 1650 GPU.

The remainder of this report is organized as follows: Section II introduces the data used for training and testing. Section III introduces the models that we analyzed. Section IV explains the training procedure we used while Section V-B presents a complete ablation study together with quantitative results of our different experiments. Finally, Section VI concludes this report with some final remarks and recommendations.

II. Data

The training dataset consists of 50000 pairs of noisy images. Furthermore, a validation set consisting of 1000 noisy-clean images was provided. We split the latter into a validation set of 400 images used for tuning the hyperparameters of each model and a test set containing the remaining 600 images used to report the final performance of each tuned model. Two examples of the noisy-clean pairs from the validation dataset can be found in Figure 1.

III. Architectures

A. DnCNN

The first architecture we experimented with is an adapted version of the DnCNN model developed in [2]. The architecture is constructed by stacking seven convolution layers, each one, except the last one, followed by batch-normalization and ReLU activation function. The first convolution layer uses 64 filters to generate as many feature maps which are then kept fixed by each convolution layer. Moreover, each convolution layer exploit padding to maintain the size of the original image. The last convolution layer uses 3 filters to reconstruct the three channels corresponding to the input image and uses a linear activation. Finally, the output of the architecture is added to the input image to obtain the final denoised image. As suggested in [2], adding the original image forces the architecture to learn a residual mapping of the noise. As a variant of this architecture, we also explored the effects of including skip connections, we refer to this method as DnCNN skip.

B. U-Net

This architecture was proposed by Ronneberger *et. al* [3] and it was then used by Lehtinen *et al.* [4] to develop the Noise2Noise architecture. The architecture is an

autoencoder-like network. The encoder path gradually decreases the size of the image by alternating convolution layers with maxpooling layers while keeping the number of channels fixed at 32. Each convolution layer is followed by batch-normalization and Leaky ReLU activation function. On the other hand, the decoder path progressively increases the size of the image to its original dimensions by means of a series of alternating Upsampling layers and convolution layers, again followed by batch-normalization and Leaky ReLu activation function. Moreover, the output of the Upsampling is concatenated with the corresponding Maxpooling output from the encoder path, leading to a constant number of 64 channels in the decoder path (in the last Upsampling the input image instead is concatenated). As for the DnCNN, the last convolution layer is followed by a linear activation. The reader can find the visual representation of the architecture extrapolated from the original paper in Figure 3 in the Appendix.

Given the constraints in the running time, we also developed a shallower version, referred to as Shallow U-Net, which contains three, instead of five, mMxpooling and Upsampling layers.

C. Shallow Deep ResUnet

Initially proposed by Zhang and Liu [5] for road segmentation, the Deep Residual U-Net (Deep ResUnet) has then been applied to many other tasks. In this project, we test its performance on image denoising. The architecture is the same as that of the Shallow U-Net, but each convolution block is constructed with a Residual Unit [6] and the number of channels in the encoder/decoder paths are progressively increased to 64 and then reduced back to 3.

D. Shallow Parallel U-Net

This architecture adapts the model introduced during the NTIRE 2020 Challenge on Real Image Denoising by the Tyan group [7]. The basic idea of this architecture is to combine global and pixel-wise denoising. The former is obtained via a U-Net architecture, while the latter is achieved using a DcCNN with dilated convolutions and unchanged feature map size. These two models are parallelized and their outputs are concatenated. After concatenation, the channels are reduced to the required output size by a convolution layer whose output is added to the input as in DnCNN. A visual representation of the architecture can be found in Figure 4 in the Appendix.

IV. Training

In this section we introduce the training procedure we used for each model. We have experimented with four different losses to optimize the model, namely: L2, L1, L0, and the relative MSE proposed in [4]. The choice of the best loss to use was made during the hyperparameter tuning phase which we explain later. In the end, the L2 loss provided the best results on the validation set for each model and thus we decided to optimize our models using this criterion.

The optimizer and its parameters were again selected based on the performance obtained during the hyperparameter tuning phase. We experimented with two well known optimizers, that is, SGD with momentum and Adam [8], and with two new optimzers developed in the past two years, that is Madgrad [9] and AdaBelief [10]. Since the latter two are not yet implemented in the torch.optim package, we only considered the former two for the best model we handed in. However, in the result section we do report some findings about regarding the use of these new optimizers. The parameters of the models are then updated using PyTorch's implementation of the backpropagation algorithm [11].

A. Hyperparameters Tuning

In order to tune the hyperparameters, we used the optuna framework [12] which implements the Tree Parzen Estimators (TPE) algorithm [13] for hyperparameter selection. For each architecture, we ran a separate hyperparameter optimization in order to perform a fair comparison. We report the optimal hyperparameters in Table II in the Appendix. Note, to align with the requirements of the project, the tuning of the hyperparameters was done considering only Adam and SGD with momentum as possible optimizers. For Madgrad and Adabelief, we ran a quick tuning in a second moment only the best-performing model.

V. Results

In this section we present the main results of this project. We first introduce the self-ensamble technique [14] we used to improve the prediction of the models and then we report and compare the performance of each fine-tuned architecture.

A. Self-ensamble

The self-ensamble technique can be viewed as a post-processing data augmentation. The idea is to enhance the prediction for every image by considering the average of the model's predictions for transformed versions of the original image. In particular, we average the model's denoised output for eight transformations of the input image (flipped and rotated 90, 180, 270 degrees). This method has often been used in practice [7] and has been proven helpful in improving the overall performance.

B. Performance evaluation

The evaluation metric we used in this project is the Peak Signal-to-Noise Ratio (PSNR). The PSNR performance obtained by each model after training for thirty epochs is reported in Table V-B. We present the averaged PSNR over five independent runs to minimize the impact of randomness. Clearly, the DnCNN model with self-ensamble achieves the best performance with a test PSNR of 25.832 db.

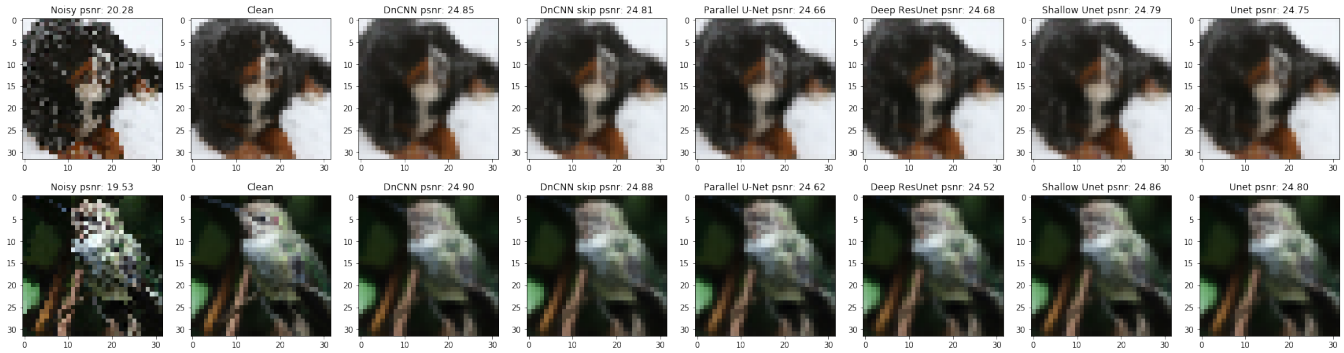

Figure 1. Comparison of visual denoising performance of various models on two samples from the test-set.

Architecture	Test PSNR (db)
DnCNN	**25.832±0.005**
DnCNN without self-ensamble	25.637±0.006
DnCNN skip	25.819± 0.003
Shallow Deep ResUnet	25.470±0.007
U-Net	25.673±0.007
Shallow U-Net	25.726±0.02
Parallel U-Net	25.501±0.011

Table I
EXPERIMENTAL RESULTS

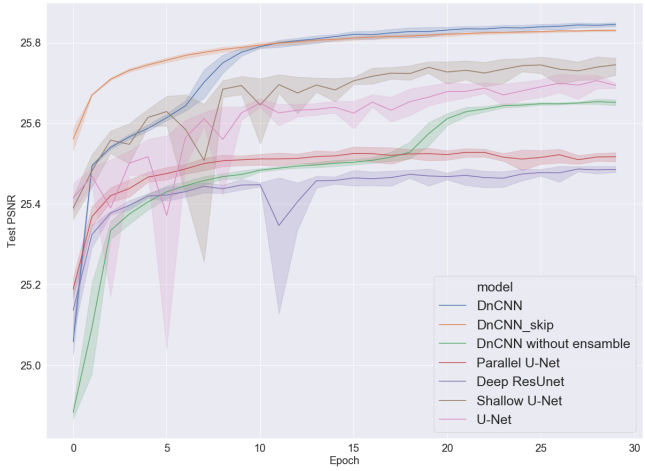

Figure 2. Test PSNR as a function of the number of epochs. The plot displays the average and the 95% confidence interval over 5 independent runs

It is interesting to notice that, possibly due to the small dimensions of the given images, simpler models seem to achieve better performances compared to deeper ones, and they seem to be doing so more consistently (tighter confidence interval). We further investigate this last point by analyzing the evolution of the PSRN obtained on the validation set by each model during the training phase. In Figure 2 we plot the evolution epoch-by-epoch of the validation PSNR for each model during training. We trained each model for thirty epochs since after this number of epochs the loss plateaued. Here, again, the plot displays the average over 5 different runs and plots the 95% confidence interval. It can be noticed that the models based on the DnCNN architecture present lower variances and more stable developments. Moreover, the plot evidences the benefits of employing the self-ensamble technique which boosts the performance of the DnCNN model by approximately 0.2 db. Finally, in our experiments with Madgrad and Adabelief, we noticed that the final performance after tuning the hyperparameters was not significantly different. However, it was interesting to see that they were able to achieve very good results with fewer tuning trials, that is they seemed less dependent on the hyperparameters.

Overall, the DnCNN architecture outperformed all others but the difference in performance between the models does not seem extremely large. In fact, the maximum performance difference on the test set is of around 0.35 db. To see if this difference is significant, in Figure 1 we show the noisy and clean versions of two images randomly selected from the test set together with the denoised output of each model. While the reported PSNR is different, the estimated image does not show significant differences. All the images seem slightly over-blurred, likely due to the use of a L2 loss criterion during training.

VI. CONCLUSIONS AND FUTURE IMPROVEMENTS

In this project, we presented our solution for the Noise2Noise project. After experimenting with different architectures, each one of them properly tuned, we showed that the simple DnCNN architecture combined with the self-ensamble technique has proven to be superior to all other models in terms of the PSNR achieved on a held-out test set. This model achieved an average test PSNR of 25.83 db. Moreover, we have shown that simple architectures displayed a more stable development of the test PSNR over the epochs as opposed to the lager variances characterizing more complex architectures. Nevertheless, all architectures evidenced a similar denoising abilities in terms of their visual outputs.

References

[1] S. Gu and R. Timofte, "A brief review of image denoising algorithms and beyond," *Inpainting and Denoising Challenges*, pp. 1–21, 2019.

[2] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising," *IEEE transactions on image processing*, vol. 26, no. 7, pp. 3142–3155, 2017.

[3] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[4] J. Lehtinen, J. Munkberg, J. Hasselgren, S. Laine, T. Karras, M. Aittala, and T. Aila, "Noise2noise: Learning image restoration without clean data," *arXiv preprint arXiv:1803.04189*, 2018.

[5] Z. Zhang, Q. Liu, and Y. Wang, "Road extraction by deep residual u-net," *IEEE Geoscience and Remote Sensing Letters*, vol. 15, no. 5, pp. 749–753, 2018.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[7] A. Abdelhamed, M. Afifi, R. Timofte, and M. S. Brown, "Ntire 2020 challenge on real image denoising: Dataset, methods and results," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 496–497.

[8] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[9] A. Defazio and S. Jelassi, "Adaptivity without compromise: a momentumized, adaptive, dual averaged gradient method for stochastic optimization," *arXiv preprint arXiv:2101.11075*, 2021.

[10] J. Zhuang, T. Tang, Y. Ding, S. C. Tatikonda, N. Dvornek, X. Papademetris, and J. Duncan, "Adabelief optimizer: Adapting stepsizes by the belief in observed gradients," *Advances in neural information processing systems*, vol. 33, pp. 18 795–18 806, 2020.

[11] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[12] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.

[13] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *Advances in neural information processing systems*, vol. 24, 2011.

[14] R. Timofte, R. Rothe, and L. Van Gool, "Seven ways to improve example-based single image super resolution," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1865–1873.

Appendix

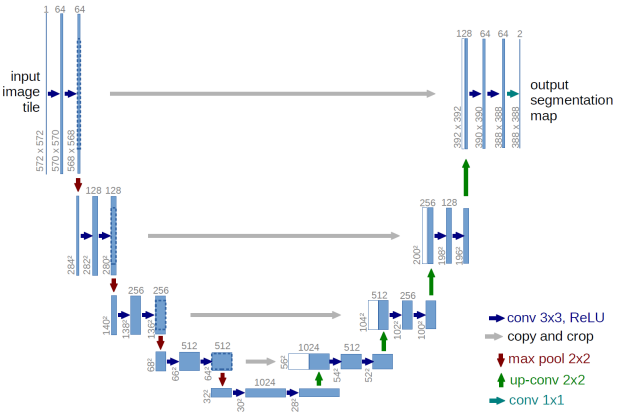

Figure 3. Representation of the architecture of the U-Net. The image is taken from the original paper, so the specific parameters do not match our implementation

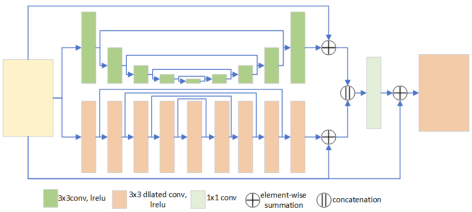

Figure 4. Representation of the architecture of the Parallel U-Net. The image is taken from the paper describing the challenge in which this architecture was proposed.

Table II
Optimal hyperparameters for each model

Par.	DnCNN	DnCNNs	Par UNet	ResUNet	UNet	Sh UNet
lr.	0.0031	0.0031	0.0036	0.0021	0.0031	0.0011
optim.	Adam	Adam	Adam	Adam	Adam	Adam
β_1	0.875	0.875	0.875	0.875	0.925	0.875
β_2	0.991	0.991	0.993	0.993	0.993	0.983
loss	L2	L2	L2	L2	L2	L2
activ.	ReLU	ReLU	PReLU	L ReLU	L ReLU	PReLU
batch.	25	25	25	25	25	25
w. decay	1e-4	1e-4	2e-8	2e-8	5e-8	5e-8