Deep Learning Project Part 1 — Autoencoders and Noise2Noise

Nicolas Wagner, Hugo Lanfranchi, Rémi Delacourt
School of Computer and Communication Sciences, EPFL, Switzerland

***Abstract*—Computer vision, and more specifically denoising images is an important and complicated topic in machine learning. Denoising images is the process of transforming corrupt data into meaningful data or simply the process of removing noise from a contaminated image (e.g noise on images can be caused by the environment or the sensor used). Different type algorithms exist to resolve this task, by using different training dataset : pairs of noisy and clean images (N2C) or pairs noisy and noisy images (N2N). In this project, the goal is to implement a neural network that use an N2N dataset. In this report, we present our approach based on a convolutional UNet implemented in the given paper [1], in which we tried to explore various part of the training pipeline.**

I. Introduction

The goal of the first project is to implement an image denoising network trained without a clean reference image following the Noise2Noise (N2N) paper [1]. The data given comprises a training dataset containing 50000 pair of noisy images of size 32x32 and validation dataset of 1000 images of size 32x32 to verify our performance. The dataset is pretty diverse and pair of images represent of a different scene(e.g. a dog, an insect, ...). This report presents how we handled the task, by presenting our model, which modifications and the data augmentation techniques we used.

II. Model selection

A. The base Model

We explored different models throughout our project. As suggested in the paper the main architectures used were RED30 and UNet. We first started comparing the two models using some basic parameters and we soon had better results with UNet (22.93 dB for RED30 vs 24.5 dB for UNet, with similar parameters), so we decided to explore the latter.
The UNet is a renowned CNN architecture developed by *Ronneberger et al. (2015)* [2]. The architecture was initially developed for biomedical image segmentation but it turns out to generalize well to computer vision tasks, and that's why it was chosen for our problem.
The model architecture has 2 main components: The contraction and the expansion. The contraction part shrinks the image using pooling layers, increasing the number of channels to capture more underlying feature maps, and the expansion part expands back into the original image size with 3 channel. The expansion part uses upsampling and combines the image in the equivalent contraction level with the given image. To further speedup the convergence of our model until an optimal loss, we also added batch normalization at each convolutional block.

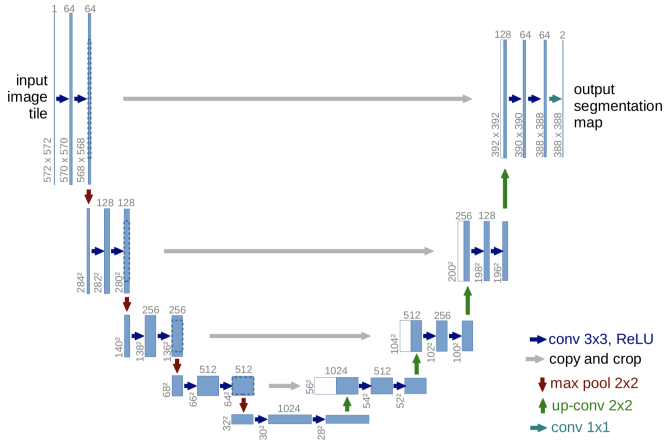

Fig. 1. UNet architecture - Ronneberger et al. (2015)

After careful training with different parameters, we ended up deciding that the best parameters we obtained for our model were a batch size of 100, and learning rate of 10^-4 and 15 epochs, the following results presented in this report assume this parameters are the ones our model has.

B. Depth

As neural networks (empirically) tend to better predict as the network gets deeper, we decided to try increasing the depth of our UNet. Initially we used the UNet with depth 4, and for input images of size 32*32 with 3 channels this gives about 34.5 millions total parameters to train. For the same input image size, increasing the architecture to have depth 5 gives more than 1383 millions parameters to train. We were actually limited by the memory size of our hardware and could not try for depth greater than 5 and limitations also arise with the respect to the time given to train the model (at first 10min).

C. Loss functions

We used different loss functions to train our model, notably some of the ones proposed in [1]: L2Loss (MSELoss) and L1Loss. But, we also tried to look at some of the loss functions offered by pytorch (CrossEntropyLoss, BCELoss (with sigmoid), ...), but only the HuberLoss (equivalent to L1SmoothLoss in our case) seemed to be useful for our task.

Loss	PSNR (dB)
MSELoss (L2)	**25.1356**
L1Loss	24.2437
HuberLoss (L1Smooth)	25.0376

Table 1 : PSNR score on validation dataset according to different training loss functions

Table 1 presents the results of using different losses to train our neural network and shows that the MSE Loss gives us the best PSNR.

D. Activation functions

We use the ReLu activation function to avoid the vanishing gradient problem. We tried to derive from the initial UNet of the paper and changed the activation function of the expansion and convolution layers between LeakyReLu and ReLu.

Conv2D activation	Decode activation	PSNR (dB)
ReLu	ReLu	**25.20**
LeakyReLu	LeakyReLu	25.07
LeakyReLu	ReLu	25.00
ReLu	LeakyReLu	24.85

Table 2 : PSNR score on validation dataset according to different activation function

Table 2 presents the results of using different activation function to train our neural network and shows that using two ReLu gives us the better PSNR.

E. Dropout rate

To speedup the training time and to prevent the model from over-fitting we added dropout layers at each convolutional block of the architecture and each expansion layer. We tried different percentage of dropout for both dropout rates.

Conv2D dropout rate	Decode dropout rate	PSNR (dB)
0.0	0.2	25.0216
0.0	0.5	**25.2346**
0.0	0.7	25.1144
0.005	0.2	25.0565
0.005	0.5	**25.1996**
0.005	0.7	25.0592
0.05	0.2	25.0635
0.05	0.5	25.0447
0.05	0.7	25.1277
0.1	0.2	25.0781
0.1	0.5	24.8250
0.1	0.7	25.1326
0.2	0.2	24.7096
0.2	0.5	24.8398
0.2	0.7	24.6155
0.5	0.2	22.3995
0.5	0.5	22.7929
0.5	0.7	23.1315

Table 3 : PSNR score on the validation dataset according to different dropout rates

Table 3 presents the results of using different dropout rates to train our neural network. This shows that using a dropout rate for the convolution layer is obsolete in this case unless we are expected to use a very low percentage. But, the dropout module is useful in the expansion layer, in fact 0.5 as dropout rate give us the best PSNR.

III. Data augmentation

To increase the number of data samples available for training we applied some data augmentation techniques.

A. Transformations

We explored a few data augmentation strategies. A package of pytorch, Torchvision offers a lot of transformations on tensor images: rotation, horizontal and vertical flip, Gaussian blur, hue, saturation, contrast, brightness, etc. We decided to make use of rotation, the two flips and the hue transformation. And use those to implement our own transformations to be able to apply them to two images, since noisy images come by pair. We performed the following augmentations :

- A rotation of random angle of 90, 180, 270 degrees and a vertical or horizontal flip
- A hue change with a factor of either 0.5 or 1.5
- The two previous augmentation combined

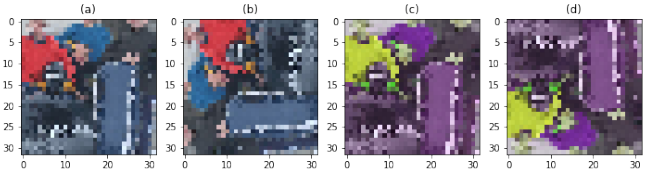

Fig. 2. Example of augmentation: (a) is the original noisy image, (b), (c) and (d) are respectively the result of the first, second and third augmentations

Figure 2 show the result of augmenting an image according to the previously described transformations. The idea is that the model might predict the noise image differently on the rotated image because it is able to capture different underlying information from the image. The angles chosen are all the multiples of 90° in the 360°circle, i.e. 90°, 180°, 270°. This is to prevent loss of information, if the angle is not a multiple of 90°, pixels in the corners are lost because the scale of the image increases. Rotation doesn't affect the noise distribution of images and therefore no information about the noise is lost or distorted. The idea for the change in the hue is very similar, we suspect that changing it might enable our model to capture more information about the noise distribution. And in the same way since the change in hue factor is the same for all pixels, we suspect that information about the noise still remains and that it might indeed make our model more robust to new data.

Augmentation	PSNR (dB)
None	25.1996
RotFlip	25.4144
Hue	**25.4604**
RotFlipAndHue	25.4515

Table 4 : PSNR score on the validation dataset according to different data augmentation techniques

What we see in Table 4 is that indeed augmenting our data does augment our PSNR in each of the cases and that a combination of both transformation also produce better results than without. Still we should be careful since the difference is not that significant.

B. Region swap

We tried an another strategy to generate new training datasets. According to [3], when the available dataset is limited, you can generate new pairs from the initial one to use them as new data to train on. Let's take x and y a pair noisy images used to train our N2N Unet, the idea to get new images is to select a region of pixel in x and swap it with the same region in y. This way as long as x and y are interchangeable (”y and x are interchangeable as long as the images are well-aligned, and the noise is not correlated (or correlation is not destroyed in the process)” [3]), then by randomly selecting this region of pixel we can generate a lot of new noisy pairs. And this is what we tried in our training, our version is bit simplified in comparison to the studied version, for every epochs we create for each noisy pair a new noisy pair based on this idea mentioned previously. This way at each epochs we have a training dataset that is slightly different but still represent our noise (if the previous condition are assumed true).

Figure 4 show use an example of the pixel region swap explained (it is quite difficult to see which small region changed). We trained the model using this method but the results was disappointing, the PSNR was sometimes higher but often equal or lower than without this method (from **25.05 dB** to **25.34 dB**).

IV. Final model used and result

The final model we settled on is a model of 5 layers with a batch size of 100, a learning rate of 10^-4 and 15 epochs. The dropout rates were fixed at 0.005 and 0.5 for respectively the convolution and expansion layer. The strategy of data augmentation we decide on is the hue transformations. Finally, the model was trained with an MSE Loss and an Adam optimizer. Figure 4 is example of prediction from our model obtained. Our best PSNR obtained with this model is ?? dB.

V. Discussion & Future Work

Overall our denoising algorithm performs well. Still further improvements can be done in various part of the training pipeline. Firstly, a more in depth look into learning rate, could be useful (e.g reducing the learning rate during the training at some threshold value on the loss). Secondly, survey more

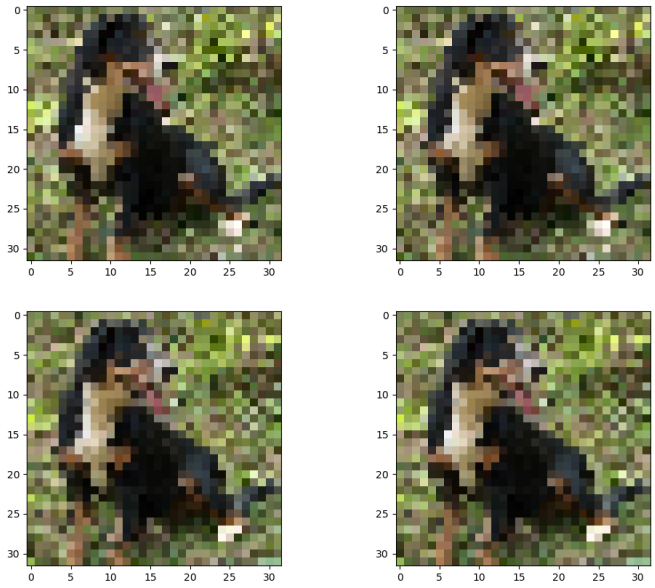

Fig. 3. Example of region swap: to the left the original noisy pair, to the right the new noisy pair after the swap

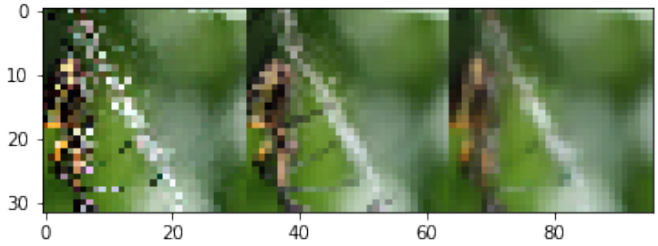

Fig. 4. Example of prediction: to the left the original noisy validation image, to the middle the clean validation image, to the right the prediction of our model

closely the overfitting of our model (e.g a module such as early stopping can be used). Lastly, As our model is very costly in space, one could look for solutions that would reduce the number of training parameters of our model while still obtaining good results.

The strategy of data augmentation with the pixel region swap seems pretty interesting and innovative, a more specific analysis of this method and how to adapt our model to it could lead to promising results.

References

[1] J. Lehtinen, J. Munkberg, J. Hasselgren, S. Laine, T. Karras, M. Aittala, and T. Aila, “Noise2noise: Learning image restoration without clean data,” *CoRR*, vol. abs/1803.04189, 2018. [Online]. Available: http://arxiv.org/abs/1803.04189

[2] P. F. Olaf Ronneberger and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” 2015.

[3] A. F. Calvarons, “Improved noise2noise denoising with limited data,” in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2021, pp. 796–805.