Denoising corrupted images using Noise2Noise EPFL EE-559, Miniproject 1

Loris Constantin, Jeremy Goumaz, Garance Haefliger

***Abstract*—Denoising corrupted images has be shown to be a highly learnable deep learning task. Using a UNet architecture to build a Noise2noise[1] network, we trained a model on 50'000 pairs of noisy 32x32 images. After tuning, notably by removing batch normalization and decreasing features complexity, we could reach a performance of 24.77 dB using the PSNR metric.**

I. Introduction

Reconstruction of corrupted images is an important image analysis field. It can be done using deep learning approaches. A conventional approach would be training a deep neural network on a dataset that contains corrupted images and their clean version. However, a *Noise2Noise* model does not follow that procedure; the dataset for such a model contains two sets of noisy images, corrupted with two different sources of independent and unbiased noise. This model can reconstruct clean images while having never seen one, as it will eventually learn the expectation of the noisy images during training, which is actually the clean version.

The goal of this project was to implement a Noise2Noise model in PyTorch without using other external libraries. We disposed of two datasets. The training set was composed of two tensors of the same 50'000 colored images 32x32 with different downsampled pixelated applied noises ; see the first column of the figure 3 to get a few examples. The test set also contained two tensors of the same images, but this time, it was a noisy and clean versions of the images, to evaluate the capacity of our model to indeed reconstruct clean images.

II. Noise2Noise model

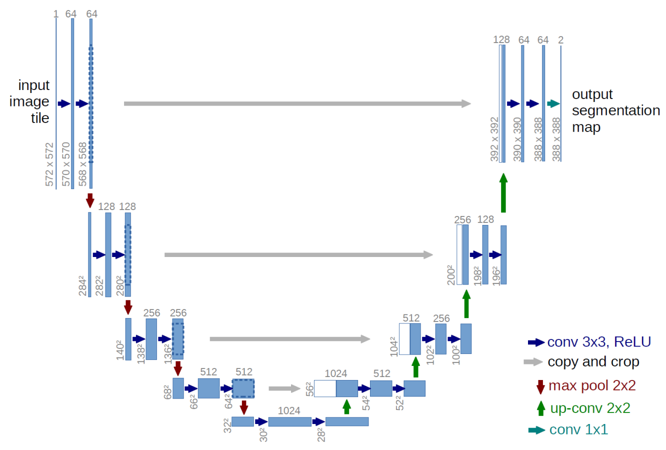

Fig. 1: UNet Architecture [2]

Our model was based on the architecture used in the original Noise2noise paper[1]: a UNet with alternating convolutional and pooling layers to encode the images, followed by alternating convolutional and upsampling layers to decode them and a final convolutional layer with linear activation. The model contains skip connections, which improves training efficiency. We reduced the general depth and feature number compared to common literature, as we worked with 32x32 images, which is rather small and therefore requires much less complexity in the layers to learn the patterns in the data.

III. Training, tuning hyperparameters

To tune the hyperparameters, we first did a comparison of nine different models with a baseline case and then, depending on the results of the first nine models, we did once again a comparison between some other models. With these two steps, we were able to distinguish performing parameters to optimally train our model. During the whole procedure, we used a batch size of 50 and 15 epochs.

To evaluate the different model settings, we split the training data into a training set and a validation set (80% - 20%). We stored the parameters of the model when the mean loss per epoch on the validation set was the lowest. Then, we could load these best parameters to apply the model on the the second dataset provided ; the one with a pair of noisy and clean images (test set). Finally, we calculated the mean Peak Signal-to-Noise Ratio (PSNR) on this dataset to get a clear indication of the model performance. PSNR is a decibel metric ; hence, the higher the PSNR is, the closer the predicted image is to the ground-truth.

Models were trained on GPU: NVIDIA Geforce GTX 950M.

A. Baseline cases

The first baseline case was : two convolutions per levels, an average pooling, batch normalization, SGD optimizer, mean square error (MSE) loss function and respectively (16, 32, 64) channels for the features. The second baseline case was the same as the previous one but with one convolution per layer and no batch normalization.

B. Loss function

We decided to not tune the loss function as we tested our models with the Peak Signal-to-Noise Ratio (PSNR) metric which is defined as : $20 * \log_{10}(MAX) - 10 * \log_{10}(MSE) = -10 * \log_{10}(MSE)$. The first term disappears because the images were normalized before to calculate the PSNR; so, the maximum of the images was 1. As the PSNR depends on the MSE, we chose it as loss function. As a reminder, $MSE = \frac{1}{MN}\sum_{i=1}^{M-1}\sum_{j=1}^{N-1}(denoised_{ij} - groundtruth_{ij})^2$ for images of size MxN.

C. Optimizer

We compared three different optimizer : stochastic gradient descent (SGD) with a learning rate of 0.01 and a momentum of 0.9, Adam with a learning rate of 0.001 and betas=[0.9, 0.999] (default parameter) and Adagrad with learning rate 0.01 (default parameter).

D. Batch normalization

We included in the model the possibility of normalizing the batches. During training, it shifts and rescales according to the mean and variance estimated on the batch.

E. Pooling

We experimented with both max and average pooling for the downsampling process. We expected that average pooling would be more appropriate given the nature of the task we were trying to learn.

F. Dropout

Dropout consists of randomly dropping out neurons in a layer with a probability that corresponds to the dropout coefficient. We implemented the possibility to include it in the model. The main advantage of this technique is that it prevents overfitting as the neurons of a layer become less dependent on particular inputs.

G. Number of convolution per layer

We implemented the possibility to choose between one or two convolutions performed per layer. In the original UNet paper [2], they used 2 convolutions per layer, but in our case with relatively small images, a lower complexity of the model was expected to perform better. All the convolutions were performed with kernel size 3 by 3, padding and stride equal to 1.

H. Number of features

Our baseline number of features were 16, 32 and 64 channels for a corresponding number of 3 convolutional layers in the contracting (and upscaling) path. We experimented with 2 to 5 layers and 4 to 64 output channels to optimize the complexity.

I. Data augmentation

To ensure the model would not become conditioned on the entry and output noise being always the same, we exchanged half the pairs of images from dataset (not actually an augmentation per se). Then, we trained models on the data provided as well as on an augmented version, containing rotated images and pairs with one of the images blurred (random gaussian filter) to diversify the noise sources.

IV. Results

By running all the different models, we arrived to a selection of the best parameters. We got best results with the SGD optimizer compare to Adam or Adagrad. Batch normalization and dropout did not improve the performance of the model; it corresponds to the results obtained in the Noise2Noise paper [1]. Average pooling was better than max pooling. One convolution per layer was preferred to two and the baseline features of 16, 32 and 64 channels were performing the best. Finally, as expected, data augmentation improved our results. In the following next two tables I and II, there is the mean PSNR (dB) on the test set for all the models we trained. As a reminder, baseline 1 corresponds to two convolutions per levels, an average pooling, batch normalization, SGD optimizer, mean square error (MSE) loss function and (16, 32, 64) for the features. Baseline 2 is the same as baseline 1 but with one convolution per layer and no batch normalization. In the tables, we indicated which parameter change with respect to their corresponding baseline.

	baseline 1	w/o batch norm	max pooling	1-conv
PSNR	15.99	24.20	17.56	24.31
	Adam	Adagrad	data aug.	dropout 0.5
PSNR	16.03	14.60	17.18	15.04
	more features			
PSNR	14.03			

TABLE I: Mean PSNR (dB) on the test dataset of the different models. The tuned parameter is indicated and is the only change from the baseline 1.

	baseline 2	max pooling	data aug.	feat 4,8,16,32
PSNR	24.53	24.53	24.67	23.40
	feat 2,4,8,16	feat 8,16	feat 16,32	feat 4,8,16,32,64
PSNR	12.68	24.45	24.48	23.38

TABLE II: Mean PSNR (dB) on the test dataset of the different models. The tuned parameter is indicated and is the only change from the baseline 2.

At this point, we had all the parameters to construct our final model. We trained the final model but changed the number of epochs to 50. This model reached its best validation loss at the last epoch. On the test set, we reached a PSNR of 24.77 dB. We can visualize the resulting images in figure 3 and the distribution of the PSNR across the test set in figure 2.

Loss	Optimizer	Batch norm	Pooling
MSE	SGD	No	Average
Dropout	Nb of conv.	Features	Data aug.
0	1	16,32,64	Yes

TABLE III: Summary table of the final best parameters

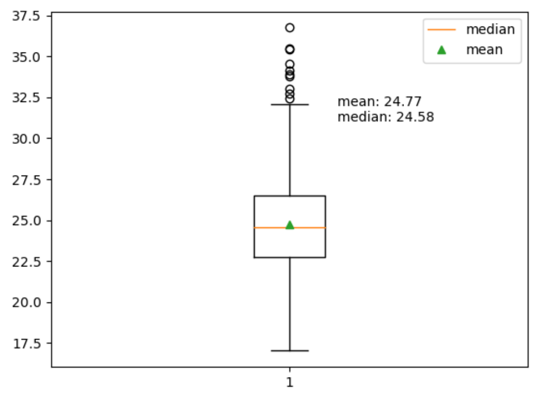

Fig. 2: Boxplot of the PSNR for the best model on the test dataset

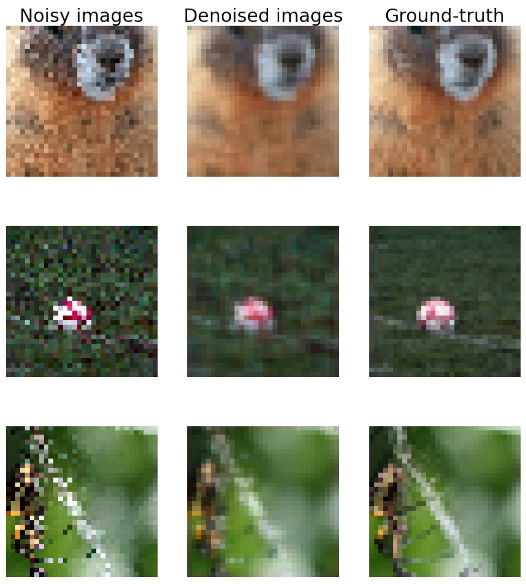

Fig. 3: Noisy, denoised and ground-truth images for some samples of the test set.

V. Conclusion

We could create a Noise2noise model reaching a mean PSNR performance of 24.77 dB on 32x32 images and reasonable tuning. The small size of the images demanded for a great reduction of the model complexity in terms of layers and features per layer compared to the usual range of the literature to reach that performance. The reconstructed images are reasonable, but we can observe a non-negligible smoothing effect compared to the original version. This might simply be due to the low resolution of the images, which comes with a higher variance between neighboring pixels and a greater difficulty to predict them from their surroundings.

References

[1] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, Timo Aila, “Noise2Noise: Learning Image Restoration without Clean Data”, 2018.

[2] O. Ronneberger, P. Fischer, en T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation”, in Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015, 2015, bll 234–241.