Denoising corrupted images using Noise2Noise EPFL EE-559, Miniproject 2

Loris Constantin, Jeremy Goumaz, Garance Haefliger

***Abstract*—Denoising corrupted images has be shown to be a highly learnable deep learning task. Using a convolutional architecture to build a Noise2noise network, we trained a model on 50'000 pairs of noisy 32x32 images. The goal of this project was to implement a library similar to PyTorch by our own and it was a success. The best achieved PSNR is 24.37 dB.**

I. Introduction

The idea of this project is to reproduce the same kind of result as the project 1 but by implementing our own PyTorch-like library. The model architecture was given but we tuned some hyperparameters in order to improve the results.

II. Implementation

We can see in Figure 1 the architecture we used during the project. It was a requirement to create it like this and we did not try to modify it.

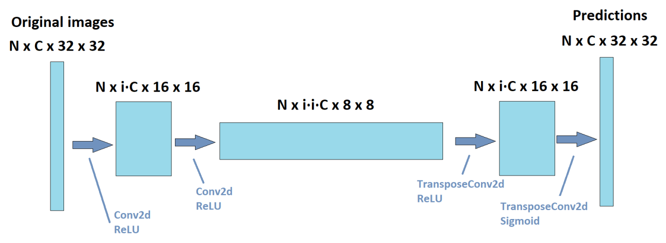


Fig. 1: Model architecture. N is the number of images, I the channel increase and C the in$_{c}hannel$.

```
Sequential(Conv (stride 2),
           ReLU,
           Conv (stride 2),
           ReLU,
           Upsampling,
           ReLU,
           Upsampling,
           Sigmoid)
```

Fig. 2: The model architecture required for miniproject 2. Upsampling layer have been replaced by TranposeConv2d layer in our case.

A. ReLU

Implementation of the activation function ReLU (Rectified Linear Unit). It is a very simple function with fast computations. The output can be calculated as following: $y = \max(0, x)$.

B. Sigmoid

Implementation of the activation function Sigmoid. This functions is very convenient in our case to end the model by rescaling the output between 0 and 1 (useful before transforming images back to RGB 0-255). However, this function uses exponential which is slower and could lead to numerical imprecision. The output can be calculated as following: $\sigma(x) = \frac{1}{1+e^{-x}}$.

C. Conv2d

Conv2d performs 2D convolutions over the images. Many parameters can be fed to this layer, we only implemented a few including stride (set to 2 as a project requirement), input/ouput channels number, kernel size but we also added padding and dilation. Considering stride=2 and dilation=1, the image shape usually reduces by a half on each of its two dimensions.

D. TransposeConv2d

We implemented TransposeConv2d as our upsampling layer (Instead of Nearest Neighbor Upsampling). This transpose convolution can reverse the shape transformation done by Conv2d which is particularly suitable. This layer can be seen as the "inverse" of Conv2d with respect to the data shape (but not with respect to the values). We also implemented the same parameters as Conv2d; stride, input/output channels number, kernel size, padding, dilation but we also added output padding, a specific parameter useful to recover the same original shape after Conv2d + TransposeConv2d.

E. MSE

MSE is the classic Mean Squared Error loss function $MSE = \frac{1}{N}\sum_{i=1}^{n}(y-\hat{y})^2$. This functions is used to evaluate the performance of the model and the backward pass is initiated from this loss. The goal is to minimize this loss in order to get better results. This loss has a reduction parameter which is either 'mean' either 'sum', this parameter determines if the term $\frac{1}{N}$ is added before the sum.

F. SGD

SGD (Stochastic Gradient Descent) is the optimisation algorithm we used here to update the weights and biases of the model. The update is classic ($\theta_{t+1} = \theta_t - \text{lr}\,\frac{dL}{d\theta_t}$).

G. Sequential

Sequential is the modules container. It stores all the layers and is able to perform forward/backward pass through the whole model. The predictions can be obtained from this class. The full architecture of the model is shown in Figure 2.

H. PSNR

PSNR (Peak Signal-to-Noise Ratio, [dB]) is not a class or a module but a metrics which can be used to evaluate the quality of a denoising model. It is defined as following: $20 * \log_{10}(MAX) - 10 * \log_{10}(MSE) = -10 * \log_{10}(MSE)$. It is important to note that minimizing MSE is equivalent to maximizing PSNR which is our goal.

III. Training, tuning hyperparameters

In order to improve the accuracy of the predictions, we have tuned 4 hyperparameters: batch size, learning rate, channel increase and kernel size. Batch size (Model training), learning rate (SGD) and kernel size (Conv2d and TransposeConv2d) are known parameters but channel increase is the multiplicative factor of channels between the convolutional layers.
We trained the models with different parameters for 10 epochs with a training dataset of 50000 images (shape $= 50000 \times 3 \times 32 \times 32$). The training set was mad of pairs of the same image with two different noise (Noise2Noise model), but after training, the model is able to predict denoised image from noised ones.
Our implementation is compatible with CPU or GPU (CUDA) computations. In our case, we used GPU computations using a NVIDIA Geforce RTX 2060.

A. Baseline cases

The baseline model includes batch size = 64, learning rate = 5.0, channel increase = 8, kernel size = 3. However many other parameters are present and they are shown in Figure 3.

IV. Results

Here are the results of the different runs with respect to the four tuned hyperparameters. We can observe in Table I that the best batch size is probably between 32 and 64. Since it takes more time to run the model with smaller batch size, we prefer to use batch size = 64. In the Table II, we can observe the effect of learning rate on the model. The best one is clearly 5.0, indeed smaller learning rate perform very bad (learning rate = 0.01 gives PSNR = 12.69). In the Table III, we can observe the effect of channel increase. The best ones are 4 and 8. Using 8 takes a lot more time but the increase is slightly significant therefore we decided to use channel increase = 8 in our best model. In the Table IV, we can observe kernel size of 3 or 5. It is common to take uneven number to remove unwanted side effects on the images (we did not try 1 because it is not a convolution and 7 because it is too large). We can observe that kernel size = 3 is the best one. It is also worth noting that we need to use padding = 2 instead of padding = 1 to use kernel size = 5. With all that said, we can compute our best model (batch size = 64, learning rate = 5.0, channel increase = 8, kernel size = 3). The best model architecture is shown in the Figure 3. The best model gives us a PSNR of 24.37 dB which is very good. Some predictions are shown in Figure 4

Batch size	64 (baseline)	32	128	256
PSNR	23.46	23.56	22.18	21.37

TABLE I: Mean PSNR (dB) on the test dataset of the different models. The tuned parameter is batch size here.

Learning rate	5 (baseline)	25	1	0.01
PSNR	23.46	21.94	21.84	12.69

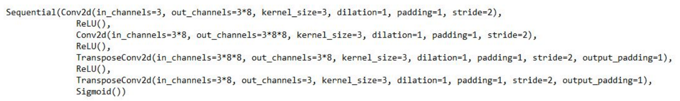
TABLE II: Mean PSNR (dB) on the test dataset of the different models. The tuned parameter is learning rate here..

Channel increase	4 (baseline)	2	8
PSNR	23.46	22.17	23.81

TABLE III: Mean PSNR (dB) on the test dataset of the different models. The tuned parameter is channel increase here..

Kernel size	3 (baseline)	5
PSNR	23.46	22.40

TABLE IV: Mean PSNR (dB) on the test dataset of the different models. The tuned parameter is kernel size here..



```
Sequential(Conv2d(in_channels=3, out_channels=3*8, kernel_size=3, dilation=1, padding=1, stride=2),
           ReLU(),
           Conv2d(in_channels=3*8, out_channels=3*8*8, kernel_size=3, dilation=1, padding=1, stride=2),
           ReLU(),
           TransposeConv2d(in_channels=3*8*8, out_channels=3*8, kernel_size=3, dilation=1, padding=1, stride=2, output_padding=1),
           ReLU(),
           TransposeConv2d(in_channels=3*8, out_channels=3, kernel_size=3, dilation=1, padding=1, stride=2, output_padding=1),
           Sigmoid())
```

Fig. 3: Best model architecture with the hidden parameters also shown.

V. Conclusion

We can conclude that this experiment has given great success. We almost got the same results as miniproject 1 and it shows that sometimes simpler architectures can be very good. The best PSNR achieved here is 24.37 dB.

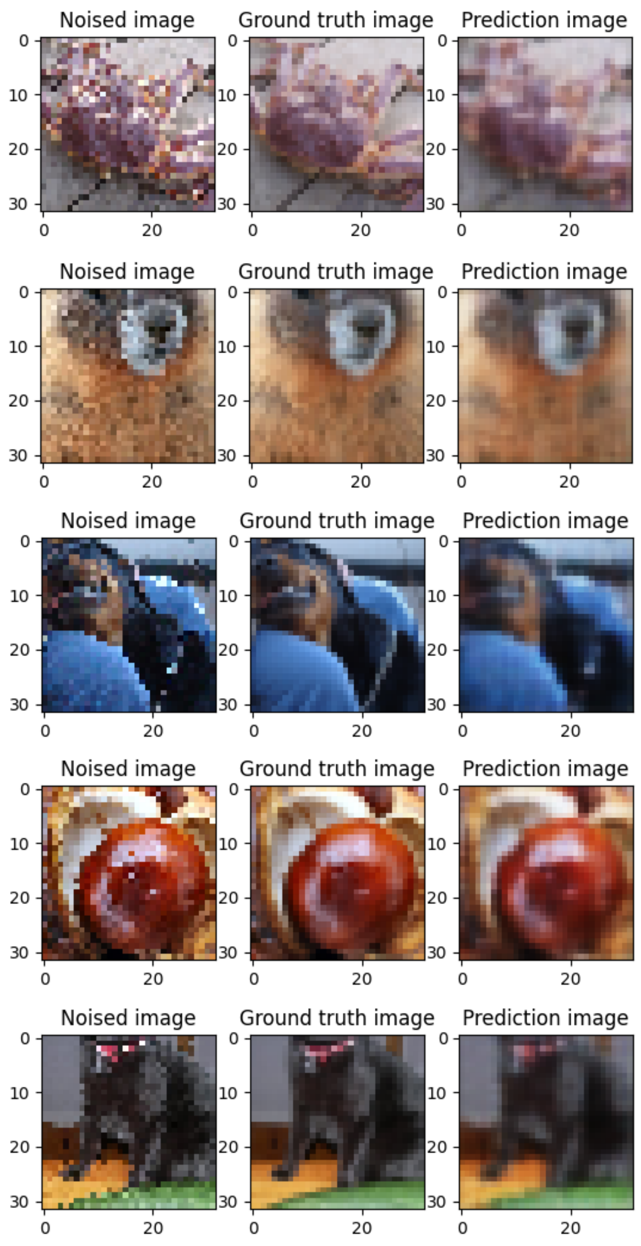


Fig. 4: Noisy, ground-truth and denoised images for some samples of the test set.

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.