ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Deep Learning (EE-559)

Mini-project 1

Group Members:

Hugo Miranda Queiros

SCIPER: 336706

Francesco Nonis

SCIPER: 300540

Erfan Etesami

SCIPER: 337448

Spring 2022

1 Introduction

The goal of the mini-project is to implement a *Noise2Noise* model using PyTorch framework. This model is meant to be trained with examples of paired noisy images and is supposed to learn to restore cleaned reference images. Our training set consists of two sets of 50,000 RGB noisy images of size 32×32 pixels. To detect the amount of noise in an image compared to the ground truth, the *Peak Signal-to-Noise Ratio* (PSNR) metric is utilized. In the remainder of this short report, the current network architecture and the corresponding results are comprehensively described first. Afterward, our design choices and process that led to the current structure is discussed.

2 Current Architecture

The network architecture we chose for this mini-project is presented in fig. 1. This structure is inspired by the U-net deep architecture [1]. At first, three convolution layers as well as three max-pooling layers are applied on the input image accordingly. Max-pooling layers are set to reduce width and height of their inputs by a factor of 2 channel-wise. Afterward, three sets of convolution layers and interpolation (upsampling) are applied on the output of the third max-pooling layer. The upsampling layers use nearest-neighbor method to double the width and height of their inputs. The kernel size for all the convolution layers is 3×3 and padding is also utilized to maintain the width and height of the inputs of the convolution layers. Note that ReLU activation function is applied on the output of all convolution layers.

The most important feature of this architecture is using the original image and two of its hidden inputs twice in its structure. More specifically, as presented in fig. 1, the output of the first two max-pooling layers and the network original input are getting concatenated (along the channel dimension) in the network path. Keeping these hidden connections avoid the gradients from vanishing such that this feature results in a better performance and faster convergence to the optimal signal-noise ratio while also eliminating the need for batch normalization as it will be addressed later.

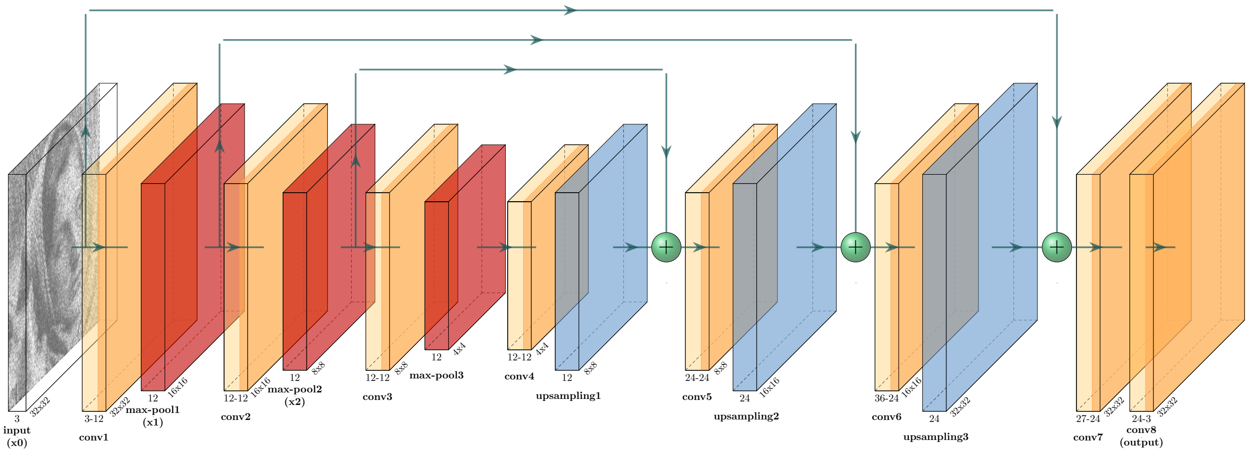

Figure 1: Our current network architecture. Yellow boxes are representative of convolution layers, and dark yellow color in each is associated with ReLU activation function. The horizontally written numbers below each convolution layer correspond to number of input and output channels respectively. The diagonal numbers are showing width $\times$ height of each output channel. Red and blue boxes are representative of max-pooling and upsampling layers respectively. The horizontal number below each box correspond to number of channels and the diagonal ones are width and height of each output channel.

Prior to start training the model, all the images (both for training and validation sets) are normalized by 255 to make sure all pixels have positive values in $[0, 1]$ interval. This makes each pixel to have a smaller distribution across images and thus it brings about a faster convergence. Moreover, as the model criterion and the optimizer, MSE loss and ADAM optimizer (with learning rate of 0.001 and default β values) are chosen respectively. The mini-batch size is set to be 50. Based on this described setup, the obtained results are summarized in table 1.

Table 1: Results

Noise Metric	**Noise w/o denoising**	**Noise w/ denoising**	**Epochs**	**Elapsed Time**
PSNR 1	19.62 dB	24.47 dB	25	598 s
PSNR 2	20.72 dB	25.39 dB	25	598 s

The two metrics of PSNR in table 1 are based on the provided codes on the instruction and `test.py` respectively. PSNR 2 seems to have around 1 dB shift compared to PSNR 1. Regardless, we trained our model for 25 epochs which fits within maximum 10 minutes of running as stated in the instruction. Each epoch takes $21 \sim 25$ s. With PSNR 1 (PSNR 2), the network can reach 24 dB (25 dB) quickly ($70 \sim 80$ s corresponding to epochs number 3 and 4). Afterward, learning becomes slower and finally reaches 24.47 dB (25.39 dB) for PSNR 1 (PSNR 2). Note that, *Noise w/o denoising* in table 1 is attributed to applying PSNR metric on the noisy and clean images of the validation set.

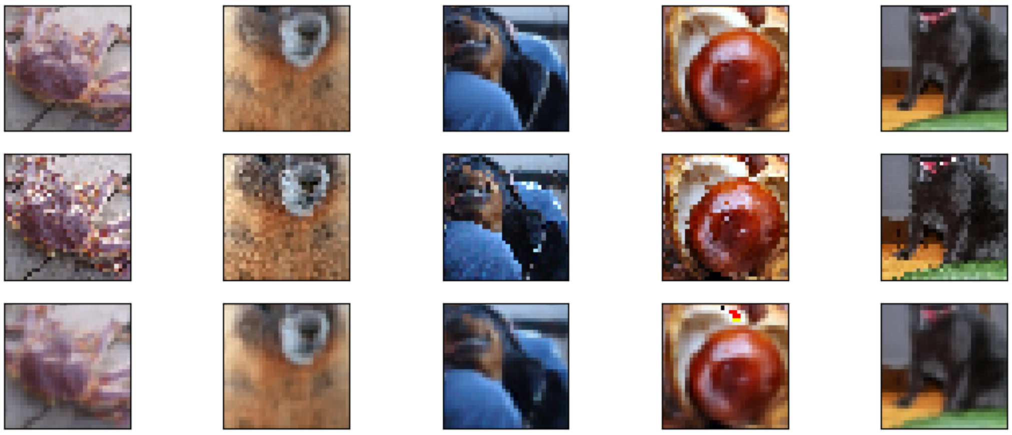

Figure 2: Five examples of clean, noisy, and denoised images. Images in the top and middles rows are extracted from clean and noisy images of the validation set respectively. The Images in bottom row are associated with the output of the network for the corresponding images in the validation set.

Furthermore, some examples of clean, noisy, and denoised images are shown in fig. 2. As it can clearly be seen, output images of the network (bottom row of fig. 2) are somewhat blurry. This is due to the properties of the MSE loss function acted on two sets of noisy images as the train input and train target. More specifically, the loss can only be decreased down to a certain point since the targets are noisy and we only improve thanks to the averaging effect of the MSE. Thus the output images are blurry such that noise is reduced overall.

3 Design Process

In this section, a brief overview is given on our findings and design choices during working on this mini-project. In the first place, note that normalizing the image inputs by subtracting the mean and then division by standard deviation resulted in a worse performance. This can be due the fact that we are dealing with noisy images and each feature has outliers such that doing so still does not force each pixel to have a value between -1 and 1. Besides, as we tested and also as it is required by the PSNR metric, it is more interesting for each pixel to have a positive value.

Moreover, we avoid shuffling the training data in the beginning of each epoch although it is a good practice in general since it reduces the chance of overfitting. Our reasoning is that, we already have two sets of 50,000 images at our disposal distributed among many different objects under the effect of some random noise. This, for itself, reduces the chance of overfitting. Besides, in our case shuffling the data took some time and made the whole learning process slower, without any improvement.

Furthermore, among various optimizers, particularly SGD and ADAM, ADAM brought about a better performance thanks to its considering running averages of the gradient which also resulted in a faster convergence. We also considered batch normalization; however, it made the training longer and also performed poorer. One reason for this phenomena is that batch normalization is sensitive to batch size (especially small ones) and also requires significant memory to store all batch statistics in the layers such that we cannot use its benefits using the ordinary GPUs of our computers. In addition, forcing the mean to be 0 is not particularly interesting in our case as discussed previously.

As to the activation functions, we tested different combinations of ReLU, Leaky ReLU, and Sigmoid. The best result we obtained is attributed to using ReLU after each convolution layer as shown in fig. 1. More concretely, Sigmoid made training a bit slower (because of the vanishing gradient phenomena) and Leaky ReLU also did not show any significant advantage over ReLU in our case.

Last but not least, we also tried various channel sizes, kernel sizes, number of layers, and also different architectures. First note that, we were limited both in terms of GPU memory and training time which restricted our design choices. For instance, in our current architecture, using 24 channels instead of 12 only made training slower without any particular progress in the final result. Kernel size of 3 is also better in our case since it squeezes more information out of each noisy image. We also tried using other architectures. One choice was a sequence of four convolution layers followed by four transpose convolution layers accordingly. Using this architecture and with proper choice of parameters, we were able to reach around 24.4 dB with PSNR 1. However, the convergence were slower than our current structure. We also implemented a structure similar to RED-Net [2] which was also slower compared to our current network.

References

[1] Lehtinen, J., Munkberg, J., Hasselgren, J., Laine, S., Karras, T., Aittala, M., & Aila, T. (2018). Noise2Noise: Learning image restoration without clean data. *arXiv preprint arXiv:1803.04189.*

[2] Mao, X., Shen, C., & Yang, Y. B. (2016). Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections. *Advances in neural information processing systems, 29.*