EE-559 Deep Learning
Report on mini-project 1

Spring 2022

Yixuan Xu
Master student in computer science
EPFL
Lausanne, Switzerland
yixuan.xu@epfl.ch

Yifei Song
Master student in computer science
EPFL
Lausanne, Switzerland
yifei.song@epfl.ch

Kanlai Peng
Master student in applied physics
EPFL
Lausanne, Switzerland
kanlai.peng@epfl.ch

***Abstract*—In this report, we discuss our choice for the architecture of the network, the experiments that we perform, and make analysis on the final results that we get.**

I. Introduction

In this mini-project, our goal is to implement a Noise2Noise model, which has been introduced in section 7.3 of the course, using the PyTorch framework. A Noise2Noise model is an image denoising network trained without a clean reference image, the idea was first developed in the paper (Lehtinen et al. 2018)[1]. The main task of this mini-project is to choose an appropriate architecture of the model so as to achieve the best denoising performance.

II. Architecture of the network

A. The general structure

In the paper about Noise2Noise model[1], the authors mainly use a specific type of deep convolutional network called 'U-Net' (Ronneberger et al. 2015)[2] for their experiments, which consists of a contracting path and an expansive path, gives it the u-shaped architecture.

Following this idea, in our work we have chosen the same architecture as stated in the Noise2Noise paper[1] which is composed of a chain of convolution layers and symmetric deconvolution layers which are combinations of nearest-neighbor upsampling and convolution layers.

B. Encoder & Decoder blocks

The underlying design principle and architecture of U-Net is based on autoencoders, where we have an encoder that maps the data from the original space to a latent space, and decoders that map it back to the original space. To make the architecture of the network clearer, we have built up several encoder blocks and decoder blocks in our model.

A encoder block consists of a 3×3 convolution layer, followed by a ReLU activation and a 2×2 maxpooling layer which dowansamples the freature map. Essentially, the encoder works like feature extractor, which preserves the features which are the most important in the image and eliminated the corruptions. Consequently, by continuously forwarding through these encoder blocks, we are capable of converting the corrupted input images into 'clean' ones.

A decoder block is generally made up of two 3×3 convolution layers each followed by a ReLU activation, plus a 2×2 nearest-neighbor upsampling layer. These decoder blocks are able to recover the details of image contents and we will get the recovered clean versions of the input images as the output of these decoder blocks.

It is also worth mentioning that in all the convolution layers of our model, we have added a padding = 1 to maintain the size of the input images after they have passed through these convolution layers.

III. Experimental Settings

A. Loss function

1) L_0 Loss: One loss function that has been used in the Noise2Noise paper[1] is the L_0 loss, which is defined as:

$$L_{l0} = \frac{1}{N} \sum_{i=1}^{N} (|f_\theta(\hat{x_i}) - \hat{y_i}| + \epsilon)^\gamma \tag{1}$$

where $\epsilon = 10^{-8}$ and γ is annealed linearly from 2 to 0 during training.

2) L_1 & L_2 Loss: The other types of loss function that we consider are respectively L_1 and L_2 loss, which are defined as:

$$L_{l1} = \frac{1}{N} \sum_{i=1}^{N} |f_\theta(\hat{x_i}) - \hat{y_i}| \tag{2}$$

$$L_{l2} = \frac{1}{N} \sum_{i=1}^{N} |f_\theta(\hat{x_i}) - \hat{y_i}|^2 \tag{3}$$

B. Kaiming Weight Initialisation

In our work, the network weights were initialised following Kaiming weight initialisation strategy proposed by He et al.[3] and mentioned in the original Noise2Noise paper[1]. Such initialisation method takesthe non-linearity of the ReLU

activation function we used into consideration, and the resulting weight values will be sampled from $\mathcal{U}(\text{ - bound, bound })$ where:

$$\text{bound} = \text{gain} \times \sqrt{\frac{3}{\text{fan mode}}} \tag{4}$$

C. Optimizer

In our work, we have used the Adam optimizer which was first introduced by Diederik Kingma and Jimmy Bain in their 2015 ICLR paper[4]. The main benefit of using Adam is that it combines the advantages of two other extensions of stochastic gradient descent, respectively Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). Specifically, the algorithm calculates an exponential moving average of the gradient and the squared gradient, using two parameters β_1 and β_2 to control the decay rates of these moving averages.

D. Ramp-down Learning Rate Scheduler

As suggested in the Noise2Noise paper[1], learning rate is fixed except when the model reaches 70% of all epochs, at which time we use a customised ramp-ling down learning rate scheduler to gradually decrease learning rate until 0 at the last epoch. The intuition behind this is that we would like to quickly traverse from the initial parameters to a relative good parameters, then use smaller learning rate to explore more deeply the loss function.

IV. Experiments & Analysis

A. Evaluation metric

To evaluate the performance of our model, we use the Peak Signal-to-Noise Ratio (PSNR) metric, which can be simply defined as:

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) \tag{5}$$

where MAX_I is the maximum possible pixel value of the image ([0,255]) and MSE is the mean squared error between the input and the target image.

B. Performance of the model with different choices of criterion

As stated in last section, we have considered using 3 different loss functions as the criterion of our model. To find the best one that works for our task, we respectively train the model for 5 epochs with these 3 loss functions using the full training set. We then compare their performance using the validation set by evaluating the final PSNR value that we get. The results we get are shown in table I, the unit of PSNR is in [dB]. For reference, the original PSNR value of the validation data is 19.62 dB.

From the table above we can see that L_0 loss and L_2 loss perform quite well and similarly on our task in terms of the final PSNR value. The performance of L_1 loss is a little bit worse but is also acceptable. Generally speaking, L_2 loss seems to perform even better than L_0 loss. So, for the rest of our experiments, we will choose L_2 loss as the criterion of our model.

Epochs	L_0	L_1	L_2
1	24.38	23.52	24.34
2	24.41	23.53	24.46
3	24.44	23.53	24.48
4	24.44	23.55	24.45
5	24.50	23.65	24.50

TABLE I
PSNR VALUES ACHIEVED BY THE MODEL WITH DIFFERENT CRITERION

C. Evolution of loss and PSNR value during training

To test whether our model works well on the denoising task, we train it for 15 epochs (not too many to avoid over-fitting) and first plot the evolution of the training loss and validation loss during the training process. As we can see from Fig.1 that both training loss and validation loss keep decreasing until they have reached a certain minimum. It is worth noticing that training loss exhibits a significant drop during the first epoch and goes down smoothly the rest of the time. While the validation loss keeps a stable rate of decline, except for some slight increase during certain epochs.

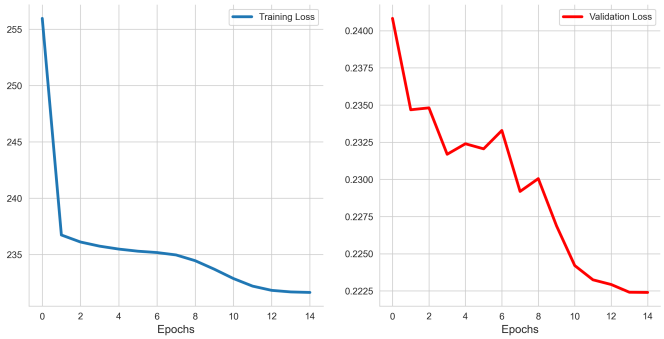

Fig. 1. Evolution of training loss and validation loss

Looking at the evolution of PSNR value as shown in Fig.2, we observe that the final PSNR value produced by our model changes in the same way as the validation loss does. Notice again that the original PSNR value of the validation data is 19.62 dB, we can conclude that our model truly works for denoising tasks.

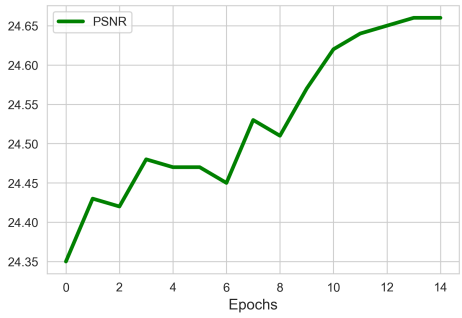

Fig. 2. Evolution of PSNR value

D. Qualitative results

To intuitively understand how our model performs for denoising task, we have printed some image examples, as show in Fig.3, Fig.4 and Fig.5. The left one is the clean target of the validation set which doesn't have noise. The image in the middle is the corrupted image which suffers from noise. Lastly, the right one is the output of our model.

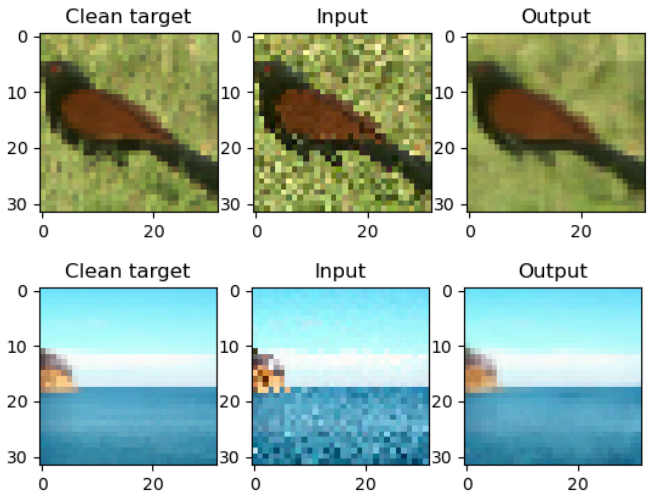

Fig. 3. Examples on uncomplicated images with simple color

As we can see from Fig.3 that, when the input image has a relatively homogeneous color distribution, or in other words, is not complicated in terms of types and variations of colors, the denoising performance of our model is relatively good. It preserves the general shape of the object and fix the incoherent color changes between adjacent pixels to make the corrupted image look more natural. There is nearly no significant difference between the output image and the clean target in this situation, except for a potential drop in the image resolution.

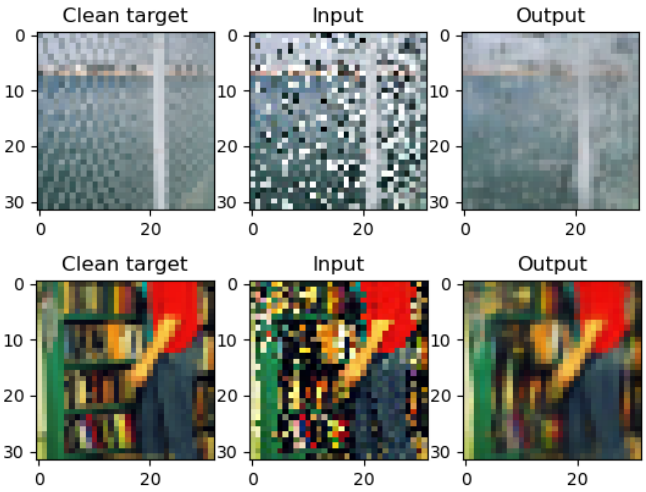

Fig. 4. Examples of situations when our model doesn't perform well

If we carefully look at the input images in Fig.4, we can find that these images are more complicated than the previous ones as they have frequent color changes between adjacent pixels and the patterns have more details. In this situation, our model doesn't performs well on the denoising task and the output images are quite blurry and look different than the clean target.

Especially, in the first target image of Fig.4, we can clearly recognize that there is a metal mesh. However, in the output image of our model, the pattern is so blurry that it is hard to recognize the metal mesh anymore. Similarly, in the second target image of Fig.4, there are many books on the bookshelf with different colors which makes the color distribution of the image quite complicated. Therefore, looking at the output image, we see that the details of these books are not well presented and the whole bookshelf looks blurry.

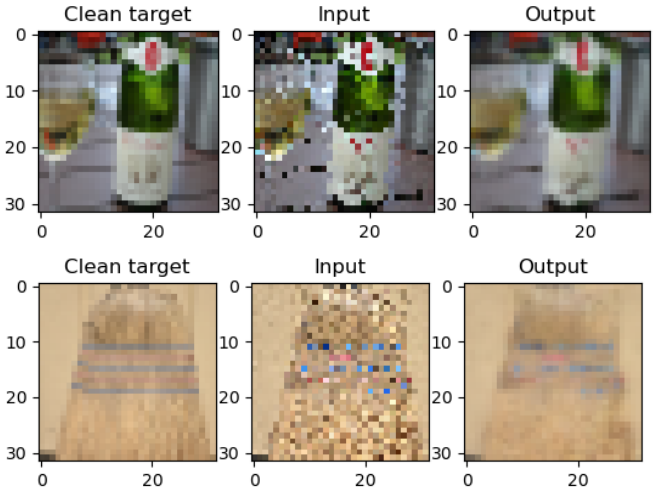

Fig. 5. Example of losing details

We can try to understand how the added noise influences the prediction results of our model by looking at Fig.5. It is clear that the details of the main patterns are heavily affected by the noise and it seems that it is hard for the model to recover the true details when they are influenced by the noise.

For example, in the first image of Fig.5, we can see that the detailed pattern on the labeling of wine bottles of the output image is strongly affected by the noise and it looks different than the one in the clean target. Also, in the second image of Fig.5, the red and blue lines are discontinuous, contrary to the clean target where they should be continuous.

V. Conclusions

Based on the previous experiments and analyses, we can conclude that our model, which has a general structure as a U-Net, works for denoising tasks. However, the quality of the output image depends on the input image itself and the type of added noise. Generally speaking, the current performance of our model is acceptable. Better results may be achieved by further tuning the hyper-parameters, like number of channels, learning rate and also the batch size.

References

[1] J. Lehtinen, J. Munkberg, J. Hasselgren, *et al.*, *Noise2noise: Learning image restoration without clean data*, 2018. DOI: 10.48550/ARXIV.1803.04189. [Online]. Available: https://arxiv.org/abs/1803.04189.

[2] P. F. O. Ronneberger and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, 2015. DOI: 10.48550/ARXIV:1505.04597. [Online]. Available: https://arxiv.org/abs/1505.04597.

[3] K. He, X. Zhang, S. Ren, and J. Sun, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, 2015. DOI: 10.48550/ARXIV.1502.01852. [Online]. Available: https://arxiv.org/abs/1502.01852.

[4] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. DOI: 10.48550/ARXIV.1412.6980. [Online]. Available: https://arxiv.org/abs/1412.6980.