EPFL

EE-559: Deep Learning mini-project 1

Chen JIAWEI
Chrysanthou MELINA
Su JINGRAN
May 26, 2022

***Abstract*—The method of Noise2Noise has been an important tool in the fast-paced rate of improvement in the area of signal reconstruction by machine learning. The aim of this report is to implement Noise2Noise model, which is able to de-noise images with any kind of noise.**

I. INTRODUCTION

This project aims at implementing a Noise2Noise model that is able to denoise images without having clean data. We trained the network based on a pair of noisy images, which could generate the same outcome as using clean data[1]. In this work, input and target are corrupted 32×32 RGB-images, and we trained the data using a U-net style model.

II. METHODOLOGY AND MODELS

Python 3.7.7 was used as the programming environment. The project was implemented with PyTorch framework only. The files *train_data.pkl* and *val_data.pkl* provided in the course were used as data sets for training and testing. The packages shown below were used, to be more specific, torchvision were only used in data augmentation.

- PyTorch 1.10.2
- Torchvision 0.11.3

A. Data preprocessing

Our training data set corresponds to 50000 noisy pairs of images. This images have 3 channels corresponding to red, green and blue. Each channel was normalised in order to have values between [0,1]. This normalisation was done by a simple division by 255 (highest value of each channel).

B. Implemented methods

1) Convolution: Makes use of filters or kernels to detect features in the images. Has learnable parameters, which are: input size(i), padding(p), kernel size(k), stride(s), and the output size follows the formulas:

$$o = \lfloor \frac{i + 2p - k}{s} \rfloor + 1 \tag{1}$$

2) Max pooling: Computes the maximum values over non-overlapping blocks. It reduces the computational cost by reducing the number of parameters to learn and provide basic translation invariant to the internal representation.

3) Transposed convolution: The reverse operation of convolutional layer, essential for our network in order to upsample optimally. Has learnable parameters, and the output size follows the formulas:

$$o = (i - 1) \times s - 2p + k \tag{2}$$

C. Loss functions and optimisation

As the main purpose of this project is to tackle a denoising problem, the following functions were used to train the model.

1) Mean Square Error loss: Measures how accurately our model is able to predict the expected outcome. To calculate the MSE, we take the difference between our model's predictions and the targets, square it, and average it out across the whole data-set. The MSE is great for ensuring that the trained model has no outlier predictions with huge errors, due to the squaring part of the function. We also tried L1 loss when tuning models, since the noisy is random, there is no much difference between L1 loss and L2 loss. From this perspective, we choose MSE as our loss function.

2) Adam: Adam has been used as the optimizer. It is a combination of momentum and root mean square propagation gradient descent methodologies, which could generates better result than other gradient descent methods. It can be used to update network weights iterative based in training data.

D. Activation function

The first choice of activation function is ReLU because it can solve the problem of gradient vanishes in the deep neural network. However, classic ReLU still suffers the problem of "dying gradient", which means gradient would be zero when input becomes negative. In our training, input value of ReLU would be negative in some cases. Since we don't want to lose information in these parts, we finally choose Leaky ReLU with slope 0.1.

E. Network structure

Our network was inspired by the structure of the original paper and receives in input a 3 x 32 x 32 tensor corresponding to pairs of 32 x 32 RGB images. Since the images were 960×540 in the original paper, which has much higher resolution than our training data, we reduced the number of channels from 48 to 16 in order to increase training speed as well as keeping good performance. We also simplified the network structure and decrease the depth of network. The structure of our network is divided in two main categories: the encoders and the decoders, as stated below:

- Encoding is the process of detecting features. In our case we have two encoders to fulfil this purpose. The first one is composed of a convolutional layer of 3 channels, 16 layers, a kernel of size 3 and stride and padding of 1 and then a Leaky ReLU non linearity. Subsequently we

have a second convolutional layer of 16 channels and 16 layers with a kernel size of 3, stride and padding of 1, followed a Leaky ReLU non linearity and a pooling layer of type Max pool of kernel 2. The output of this encoder is then passed as input to the second encoder. In the second encoder we have a convolutional layer of 16 channels and 16 layers with a kernel size of 3, stride and padding of 1, a Leaky ReLU non linearity and a Max pooling layer of kernel 2. Then the output is passed to the decoders.

- To decode means to convert a coded message or features into original space. This is why in our three decoders we are using the transposed convolution. The first decoder has a simple structure with a convolutional layer of 16 channels and then a transpose convolution layer of 16 channels. The second decoder is composed of a first convolutional layer of 32 channels(combining the output of encoder1 and decoder 1), a second convolutional layer with the same characteristics and a transpose convolutional layer of 32 channels with a kernel size of 3, stride of 2 and padding of 1. After that, the third decoder is composed of two convolutional layers and output a $32 \times 32 \times 32$ image. Finally, it was sent to the output convolutional layer and returned the demoising image.
- As we can see in the figure 1 the different elements of our architecture are communicating between them. But not all data follow the same path. We can see that some of the data are directly passed from the input to the third decoder. Same for some data at the end of the first encoder are directly passed to the second decoder. This technique is used as we want to keep some of the features when training and therefore we don't need to train them all from scratch.

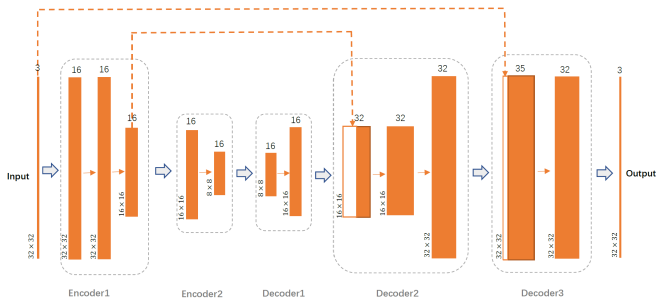

Fig. 1. Architecture of the model

F. Other techniques

Data augmentation strategies were also implemented in our model, using torchvision.transform. We expanded the training data by adding random cropped images, random rotate images, grey images and colour-transformed images.

III. Results

A. Tuning parameters

After determining the network structure, we tuned the hyperparameters of learning rate and batch size, both of them are crucial in the training process. As we can see, high learning rate would leads to a sub-optimal and less stable outcome, whereas a small learning rate results in a slow training and could get stuck easily. When considering batch size, large batch size normally take less training time but poorer generalization ability, whereas small batch size leads to more oscillates, means time-consuming, hard to converge but able to coming out of local minimum. Therefore, we need to tune these parameters and find the best trade-off. Here, we use grid search on learning rate and batch size, the values used for the tuning are:

- *Number of epochs*: 3
- *Learning rate*: [0.0002, 0.0004, 0.0.0006, 0.0008, 0.001]
- *Batch size*: [2, 4, 16, 64, 256]

The best outcome was given by lr = 0.0004, and batch size = 4. As you can see from figure 2, small learning rate and small batch generally have a better performance in our training.

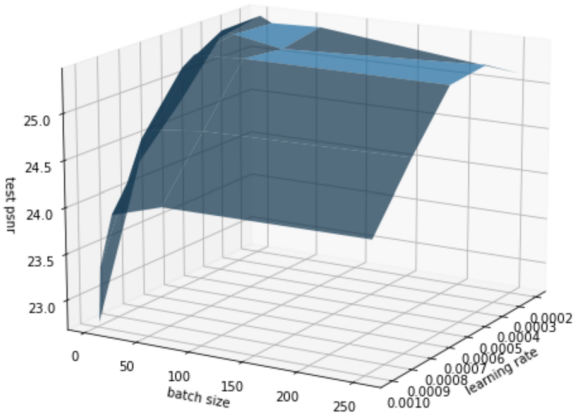

Fig. 2. Tuning hyperparameters

B. Training loss

We tested 25 epochs on the *val_data.pkl*, and obtained the following test loss and PSNR evolution:

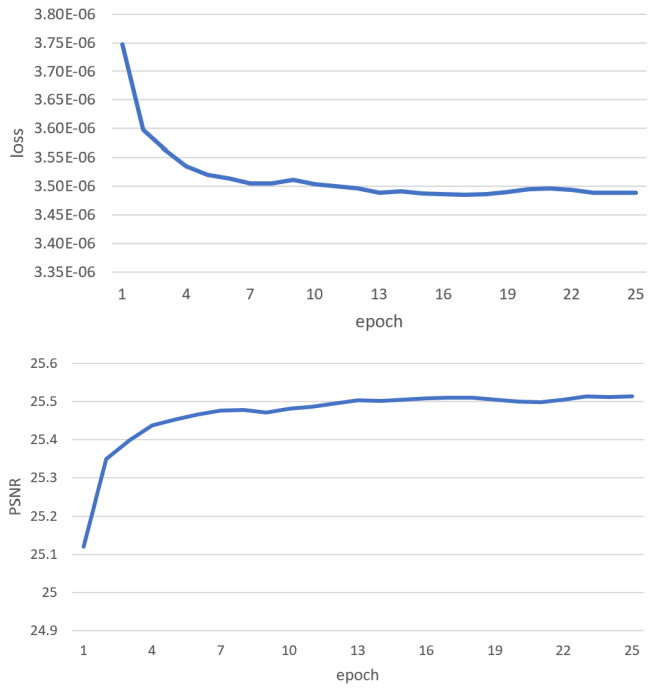

Fig. 3. Test loss and test psnr

C. *Performance of the model*

By using the best model and testing the outcome on the *val_data.pkl*, we got the average PSNR of 25.52 dB, compared with 20.72 dB before denoising.

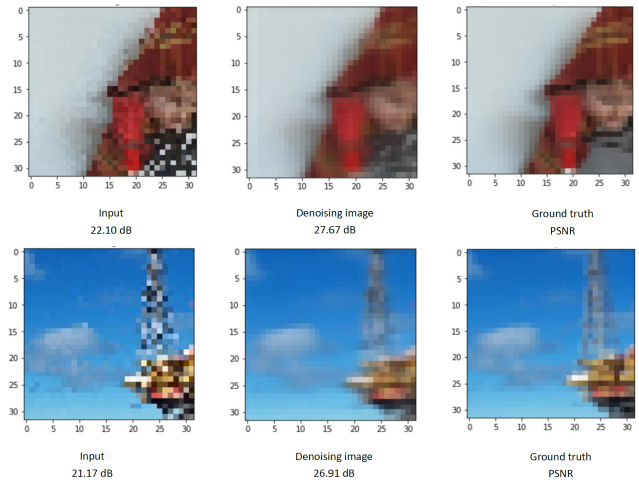

Fig. 4. Example result of denoising

IV. Summary and Discussion

In our work, we implemented a U-Net style network, which is composed of 2 encoder blocks and 3 decoder blocks, and denoised images by using only pairs of noisy data. The trained model can be used as a denoiser and return clean images with average PSNR of 25.52 dB , training hyperparameters are: learning rate 0.0004 and batch size 4. Due to the training time limit, this is the best outcome we can get, further improvement could be done by considering deeper network, adding more channels or getting more training data.

References

[1] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. Noise2noise: Learning image restoration without clean data. *arXiv preprint arXiv:1803.04189*, 2018.