REPORT: MINI-PROJECT 2

Maochen Xu, Zhipeng Mao,Yifeng Chen

I. INTRODUCTION

To obtain a better understanding of the libraries for deep learning like Pytorch, we develop some of the modules by ourselves for the second mini-project. In this project, we will implement our own framework to denoise images without using autograd or torch.nn modules.

Instead of applying the Unet structure illustrated in mini-project 1, we implement a simple structure mentioned in the mini-project 2 description, which combines 2 convolutional layers, 3 Relu activation functions, 2 upsampling layers and 1 sigmoid activation function. Besides, mean square error loss criteria is applied and stochastic gradient descent optimizer is implemented. We train our architecture based on the train dataset(50000 samples) and test our architecture based on the validation dataset(1000 samples), and still get satisfied results, where psnr value is 22.81dB.

II. MODULE IMPLEMENTATION

We implemented modules **Conv2d**, **Upsampling**, **ReLU**, **Sigmoid**, **MSE**, **Sequential**, all of which inherit from base class **Module**, and optimizer **SGD**.

A. Module

All subclasses of **module** should implement methods **forward**, **backward** and **param**.

- **forward** method do calculations on the input and return the result, which will be passed to the next layer.

- **backward** method calculate the loss gradient w.r.t the parameters as well as the input tensor based on the gradient w.r.t the output of this layer.

- **param** method return the parameters of this layer. In this project, only **Conv2d** and **Upsampling** have parameters.

B. Conv2d

Conv2d module do convolution on the input tensors.

- For **forward**, the formula is

$$Z = W * A + b \tag{1}$$

where $*$ stands for convolution. We use the method mentioned in the appendix of the project description to convert the convolution operation to matrix multiplication, which speed up the computation a lot compared to using for-loop to do convolution.

- For **backward**, the formula is

$$\delta_{out_i} = \Sigma_m \delta_{in_m} * W_{im}^{(rot180)} \tag{2}$$

$$\delta_w = A * \delta_{in} \tag{3}$$

$$\delta_b = \Sigma_x \Sigma_y \delta_{in} \tag{4}$$

We also use the unfolded matrix multiplication to do convolution.

- For **param**, it returns a 2d list, whose format is [[weight, weight gradient], [bias, bias gradient]].

This weights and biases of this module is initialized by Xavier initialization, which are initialised by Gaussian distribution, the mean iszero, and the variance is $\sqrt{\frac{2}{m+n}}$

C. Upsampling

This layer consists of an NNUpsampling layer and a Conv2d layer.

For **NNUpsampling** layer:

- The **forward** method resize the tensor by a scale factor and interpolate the tensor with the value of the nearest element.

- The **backward** method, conversely, sum up the gradients of one neighbourhood into one element.

For these two methods, we tried the for-loop implementation, which is too slow to work. So we exploit the **repeat interleave** method to do interplolating and **unfold** method to do gradient calculation.

D. ReLU

- For **forward**, it returns the relu function value of the input,

$$Relu(x) = \max(0, x) \tag{5}$$

- For **backward**, the gradient formula is

$$\delta_{out} = \max(0, x)\delta_{in} \tag{6}$$

E. Sigmoid

- For **forward**, it returns the sigmoid function value of the input,

$$\sigma(x) = 1/(1 + e^{-x}) \tag{7}$$

- For **backward**, the gradient formula is

$$\delta_{out} = \sigma(x)(1 - \sigma(x))\delta_{in} \tag{8}$$

F. Mean Square Error

- For **forward**, given input and target, it returns the MSE loss,

$$l(x, y) = \frac{\Sigma_i (x_i - y_i)^2}{n} \tag{9}$$

- For **backward**, the gradient formula is

$$\delta_{out} = \frac{2(x - y)}{n} \tag{10}$$

G. Sequential

This layer is a list of modules.

- For **forward**, it passes the input through the modules.

- For **backward**, it passes the gradient through the modules reversely.

- For **param**, it returns all the parameters and their gradients of the modules.

H. Stochastic gradient descent

SGD is an optimizer. It receives the parameters of the modules when it is constructed and updates the parameters. The formula is

$$\theta_t = \theta_{t-1} - \gamma g_t \tag{11}$$

γ is learning rate, when its **step** method is called. Before backwarding the modules, its **zero grad** method should be called.

III. Experiment

A. Network architecture

Based on the description for mini-project 2, the structure for this mini-project is as table 1:

Name	N_{out}	Function
Input	3	
Conv2d	44	Convolution 2×2
ReLU		
Conv2d	44	Convolution 2×2
ReLU		
Upsampling	44	scale factor=2, Convolution 3×3 padding=1
ReLU		
Upsampling	3	scale factor=2, Convolution 3×3 padding=1
Sigmoid		

TABLE I: the network structure

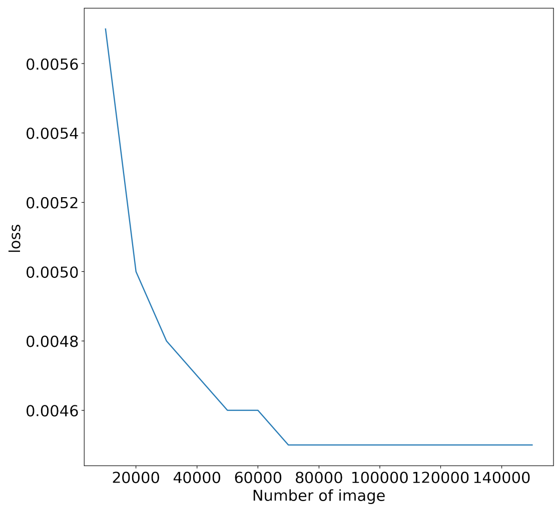

Fig. 1: Evolution of training loss

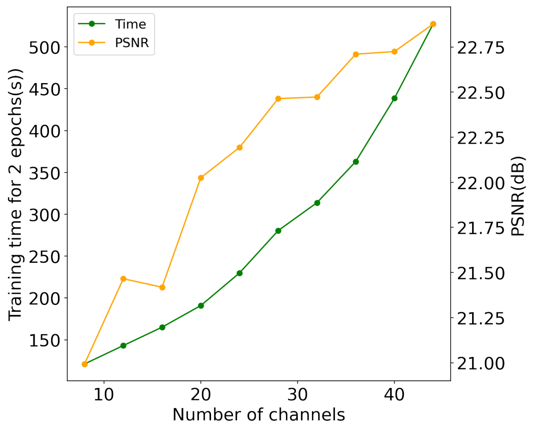

Fig. 2: Impact of channels

B. Training process

Here we will discuss how we train our model. The size of our training dataset is 50000 sample pairs, with 50000 noisy images as train input and 50000 clean images as train target, the size of each sample is 32×32, whose value is from 0 to 255. Before our training, we adjust the value of each element of each sample by dividing 255. Each time we train 10000 samples, we will calculate the average of accumulated loss, the evolution of training 2 epochs is shown in Fig.1:

C. The choice of parameters

In this section we will discuss the choice of the parameter in the network architecture and the choice of hyper parameters based on the training time, prediction results, which is calculated by PSNR formula, to find the best parameters.

First, we try to analyse the impact of channels of conv2d for the training time and the denoising effect of our network architecture, which is shown in Fig.2:

As we can see, with the increasing of the number of channels, the PSNR value is increasing, but at the same time, the training time is also increase, since the requirement for the training time of 1 epoch is below 5 minutes, we will use 44 channels for each convolutional layer.

D. The choice of hyper parameters

In this section, we will discuss the choice of learning rate and mini-batch size for our structure.

Firstly, we set different learning rate to find out the best one. Since if the learning rate is too small, it will take several epochs to get convergence, which is too time-consuming, we just consider the learning rate from 0.1 to 7, try to find the best one, the result is shown in Fig.3 As we can see, the best learning rate is 4, so we choose it as our final choice.

Secondly, we consider the impact of mini-batch size for the denoising effect, which is shown in Fig.4. As we can see, with the increasing of the mini-batch size, the PSNR in decreasing,

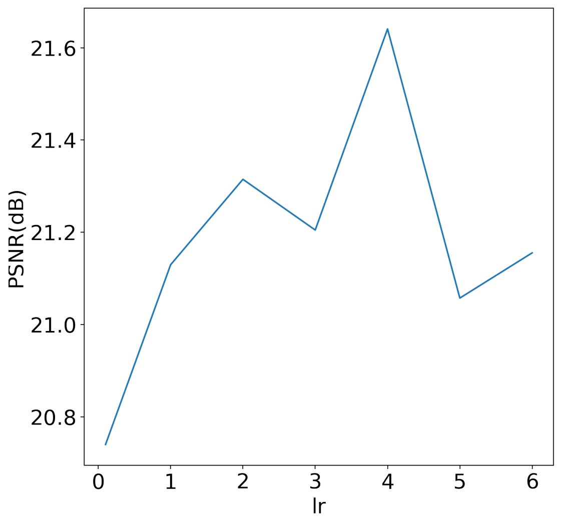

Fig. 3: Impact of learning rate

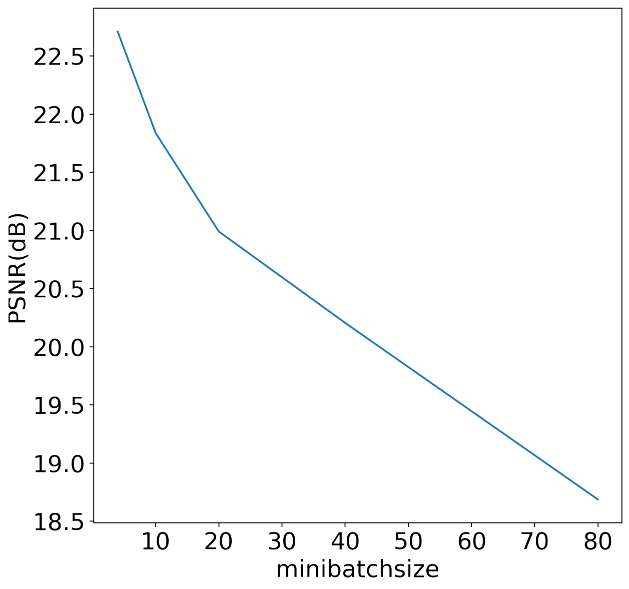

Fig. 4: Impact of mini-batch size

so we choose 4 as our final mini-batch size, which is the same as in the paper[1]

Based on our best model architecture as well as the best hyperparameters, the average PSNR for the validation dataset is 22.81dB. Several denoised examples are also shown in Fig.5:

IV. Conclusion

During our process for this mini-project, we understand each module in the proposed framework and also understand

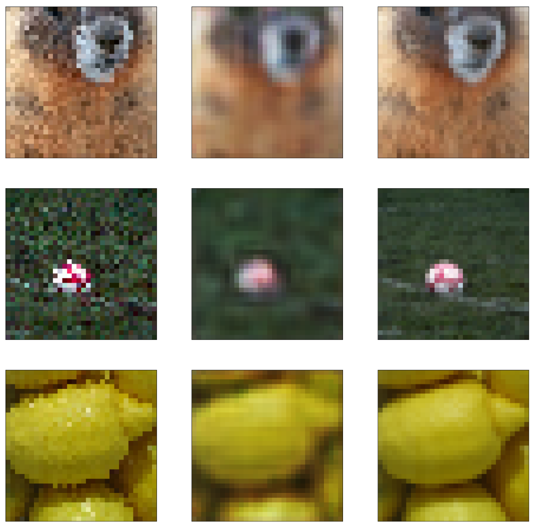

Fig. 5: Comparison between the validation target and prediction.The samples in the left column are the validation input samples, the samples in the middle column are our prediction results, the samples in the right column are the validation targets. The psnr of each row is 25.62dB, 24.77dB and 26.32dB.

the choice for each parameter as well as hyper parameter. However, we also met 2 problems during the process of building our own structure:

1. A mistake we made when implementing parameter updating is that we make the gradients inside the modules reference a new object everytime we do backwarding so that the gradients in **SGD** instance and those in the modules don't reference the same objects. Then we realized that we should do inplace operation on the gradients inside the modules (here we use **add** operation) and the gradients should be reset to zero every time before calling backward. 2. For writing the upsampling module, at first we use 4 for loops to finish the forward as well as backward function, which is not efficient. Then we apply repeat interleave as well as unfold methods, which increases our training speed significantly.

Acknowledge

We thank our professors and TA for teaching us this practical lessons as well as helping us solve some difficult problems when we had no ideas for our problems.

References

[1] J. Lehtinen, J. Munkberg, J. Hasselgren, S. Laine, T. Karras, M. Aittala, and T. Aila, "Noise2noise: Learning image restoration without clean data," 2018.