REPORT: MINI-PROJECT 1

Yifeng Chen, Zhipeng Mao, Maocheng Xu

***Abstract*—The first project is to use the PyTorch framework to implement a Noise2Noise model and perform a denoise task on the given dataset. This part of the report will contain the model framework, the experiment, and the results of the first mini-project.**

I. MODEL

With the help of the whole PyTorch framework, the model in [1] is implemented. The model is based on the structure of the U-network, an autoencoder designed for image segmentation.

The idea of Noise2Noise is to use noisy images to perform a denoise task without a clean target. The model's input is (source, target), in which the source is the image waiting to be denoised, and the target is the same image with a different noise. In this project, the image size is $(3, 32, 32)$, which means a 32×32 image with three channels. The model consists of several convolution, max-pooling, and upsampling layers. The numbers of channels we use are half of the original paper. Except for the last layer, whose activation function is the sigmoid function, the other layers' activations are all leaky relu, as mentioned in [1]. One anomaly we noticed when implementing this model is that with linear activation in the last layer, it is not guaranteed that the model's output is between 0 and 1, which must be a tensor of an image. As a result, we use a sigmoid function to map the output tensor into $[0, 1]$. The specific structure is listed in table I.

NAME	# Out	FUNCTION
INPUT	3	
ENC_CONVO	24	Convolution 3×3
ENC_CONV1	24	Convolution 3×3
POOL1	24	Maxpool 2×2
ENC_CONV2	24	Convolution 3×3
POOL2	24	Maxpool 2×2
ENC_CONV3	24	Convolution 3×3
POOL3	24	Maxpool 2×2
ENC_CONV4	24	Convolution 3×3
POOL4	24	Maxpool 2×2
ENC_CONV5	24	Convolution 3×3
POOL5	24	Maxpool 2×2
ENC_CONV6	24	Convolution 3×3
UPSAMPLE5	24	Upsample 2×2
CONCAT5	48	Concatenate output of POOL4
DEC_CONV5A	48	Convolution 3×3
DEC_CONV5B	48	Convolution 3×3
UPSAMPLE4	48	Upsample 2×2
CONCAT4	72	Concatenate output of POOL3
DEC_CONV4A	48	Convolution 3×3
DEC_CONV4B	48	Convolution 3×3
UPSAMPLE3	48	Upsample 2×2
CONCAT3	72	Concatenate output of POOL2
DEC_CONV3A	48	Convolution 3×3
DEC_CONV3B	48	Convolution 3×3
UPSAMPLE2	48	Upsample 2×2
CONCAT2	72	Concatenate output of POOL 1
DEC_CONV2A	48	Convolution 3×3
DEC_CONV2B	48	Convolution 3×3
UPSAMPLE 1	48	Upsample 2×2
CONCAT1	48+3	Concatenate INPUT
DEC_CONV1A	32	Convolution 3×3
DEC_CONV1B	16	Convolution 3×3
DEC_CONV1C	3	Convolution 3×3

TABLE I
MODEL STRUCTURE

II. IMPLEMENTATION

PyTorch is the main framework used to implement the model. The U-Network is defined and constructed in the file "unetwork.py", and the primary model of Noise2Noise is in the file "model.py". The Noise2Noise model class contains five parts, the initialization, the method to load the pre-trained model, the train loop, the predict method, and one static method to calculate the PSNR.

The loss function of the model is MSE loss, which can be defined as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2,$$

Where N is the minibatch size, y and $\hat{y}$ denote the target and the output tensor of the model, respectively.

The optimizer used in training is ADAM, which is efficient in training such a network.

The parameters used in the experiment are the same as in the original paper except for the mini-batch size. The parameter values for ADAM are $(\lambda, \beta_1, \beta_2, \epsilon) = (0.001, 0.9, 0.99, 10^{-8})$. The batch size is set to be 100 since the training set has 50,000 samples.

The Data Loader method (torch.utils.data.DataLoader) loads the data in batches. The source and target images are stacked as a $(N, 2, C, H, W)$ tensor to be put into the data loader method. During the training loop, they are then split into two images.

To monitor the training process, TensorBoard is used. The TensorBoard tracks the average loss and average PSNR of 100 batches and the network graph. The results thus can be quickly reviewed in a browser (figure 1).

III. RESULTS

The training is on a remote Linux machine with NVIDIA QUADRO P4000 GPU. The number of epochs is ten, and the entire training process lasts around 400 seconds.

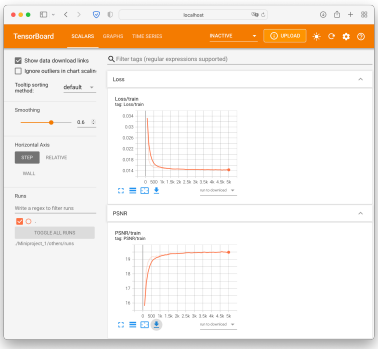
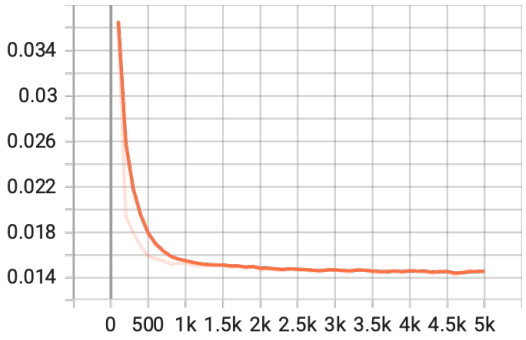
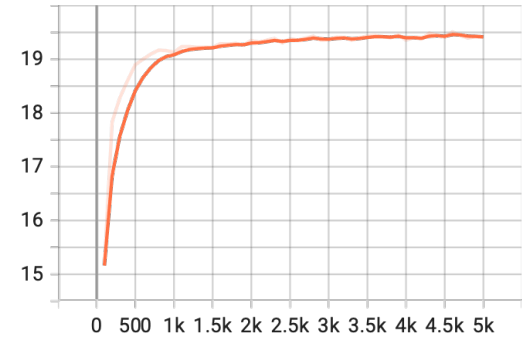


Fig. 1. TensorBoard Interface in a browser

The metrics we use in training are MSE Loss and PSNR, which can be visualized in figure 2.



(a) Loss



(b) PSNR

Fig. 2. Training Metrics

During the training process, the loss on the training set is reduced to 0.014, and the PSNR is around 19.40 dB in the last epoch. In the validation set, the PSNR is 25.31 dB after the process, whereas the original PSNR on the validation set is 20.72 dB. The first five images in the validation set are shown in figure 3.

During the evaluation process, the output of the model caused some trouble. Since the model takes in images with values between 0 and 1, the data was pre-processed before training into $[0, 1]$ by dividing by 255. Some errors occurred when the same data was divided by 255 twice, so a simple condition was added before training and predicting.

Another common error raised during the process is PyTorch cannot handle data on different devices. This is because the input data was not assigned to the same device (CPU or GPU) as the model. As a result, we ensured every data the model took in both in training and validating was assigned to ”self.device”, which is trivial but significant in solving the issue.

Fig. 3. Five targets and outputs of the validation set

Finally, we noticed that the saving method should be changed during the testing. At first, we used “torch.save(model, path)” directly, everything worked fine on the cloud machine. However, when running the test on the local machine, a ”ModuleNotDoundError” was raised. The reason behind this is that if we saved the whole model, the path this model relied on would also be saved, which will cause trouble when using another machine. As a result, instead of saving the whole model, we saved the “state_dict” of the model, then the test succeeded on both devices.

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.