PyTorch Implementation of Noise2Noise

1. Implementation details

- **Network**: We implement the U-network (Ronneberger et al., 2015) in ‘others/unet.py‘. For all basic noise and text removal experiments with RGB images, the number of input and output channels were n = m = 3. For Monte Carlo denoising we had n = 9, m = 3, i.e., input contained RGB pixel color, RGB albedo, and a 3D normal vector per pixel. The network weights were initialized following Kaiming initialization. No batch normalization, dropout or other regularization techniques were used.
- **Optimizer**: Training was done using ADAM (Kingma & Ba, 2014) with parameter values beta1 = 0.9, beta2 = 0.99, eps = 1e-8. Learning rate was kept at a constant value during training except for the last rampdown period at where it was smoothly brought to zero. Initiall learning rate of 0.001 was used for all experiments except Monte Carlo denoising, where 0.0003 was found to provide better stability. We implement the learning rate scheduler as ‘RampedLR‘ in ‘others/utils.py‘. The initial learning rate is ‘params.lr‘, and the percent of last ramped down epochs is ‘params.ramp_down_percent‘.
- **Loss**: We use L1, L2 loss (default L2) for all basic noises, and HDR loss (implemented as ‘HDRLoss‘ in ‘others/utils.py‘) only for Monte Carlo denoising.
- **Training step**: We use ‘params.epoch‘ to specify number of total epochs, ‘params.batch_size‘ to specify the batch size, ‘params.iteration_per_epoch‘ to specify number of batches per epoch.
- **Model evaluation**: The evaluation is baed on the Peak Signal-to-Noise Ratio (PSNR), implemented in ‘psnr‘ in ‘others/utils.py. We evalute the model every epoch and save the model with the best PSNR ”bestmodel.pth”.
- **Logging**: If tensorboard is installed, we use it for logging. Otherwise, we do not log.
- **Hyperparameters**: We implement ‘Model()._parse()‘ to deal with hyperparameters input for initialization.

2. Experiment & Ablation

In the following, we perform hyperparameter ablation. The default loss is ”l2”, the default batch size is 4, the default ramped down percent is 0.7 (approximately last 2/3 of total epochs), and the default upsample method is ”nn.Upsample”.

2.1. Loss

We first perform experiments on different loss function: L1, L2. Since the input images only has 3 channels, it is impossible to belong to Monte Carlo noise. See Figure 3. Clearly, ”L2” is the better choice.

2.2. Batch size

WE test batch size as 4, 8, 16, 32. See Figure 6. Too large batch size will be harmful to the accuracy. There is no major difference among different batch size in terms of PSNR and loss when batch size <= 16. But smaller batch size will be more efficient in terms of training, so we set batch size to be 4.

2.3. Ramped down percent

WE test ramped down percent as 0.3, 0.5, 0.7, 0.9. See Figure 9. Clearly, setting ramped down percent is 0.7 or 0.9 is better.

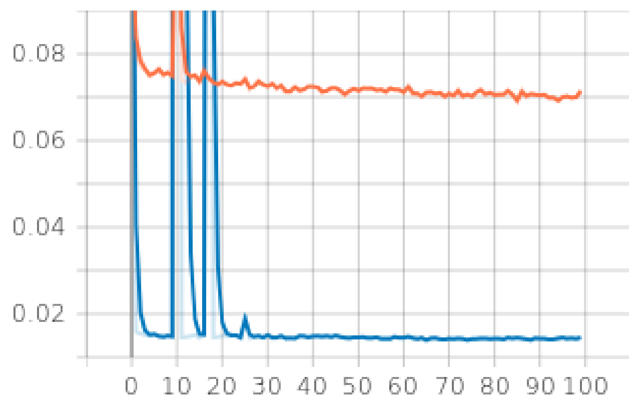

Figure 1: Training Loss

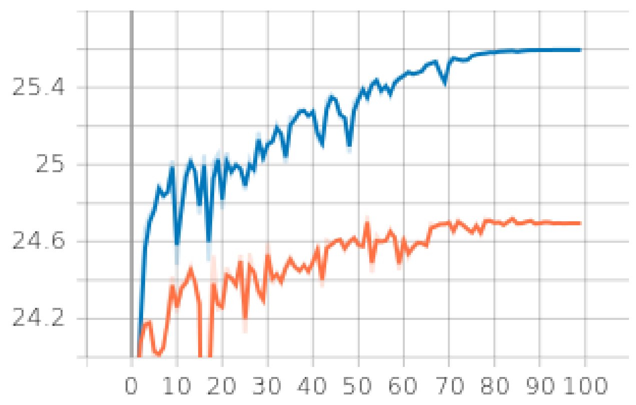

Figure 2: PSNR

Figure 3: Orange: L1 Loss. Blue: L2 loss.

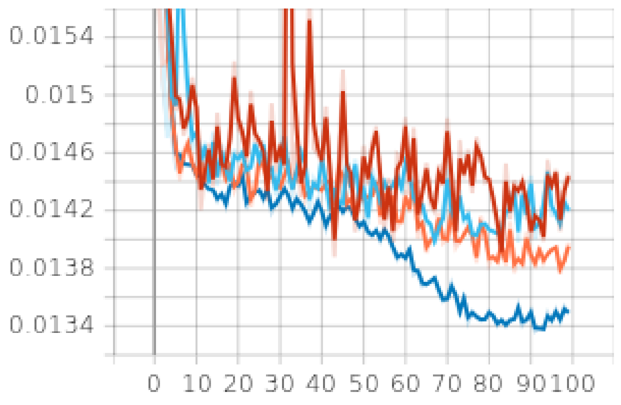

Figure 4: Training Loss

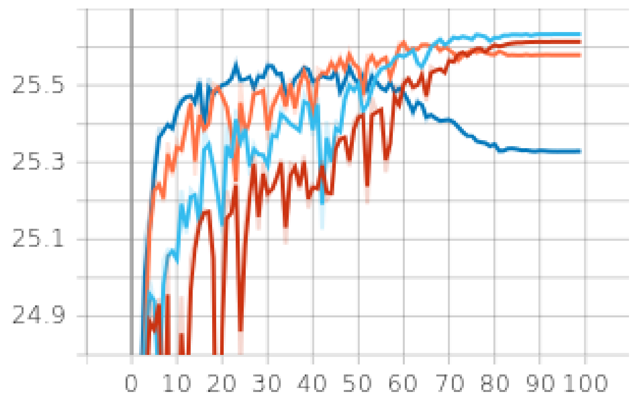

Figure 5: PSNR

Figure 6: Red: 4, Light Blue: 8, Orange: 16, Navy blue: 32.

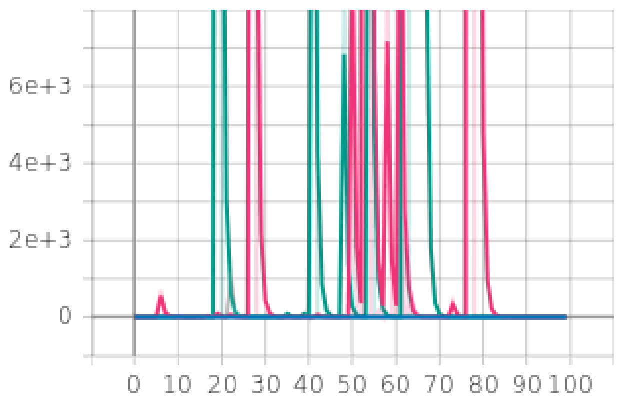

Figure 7: Training Loss

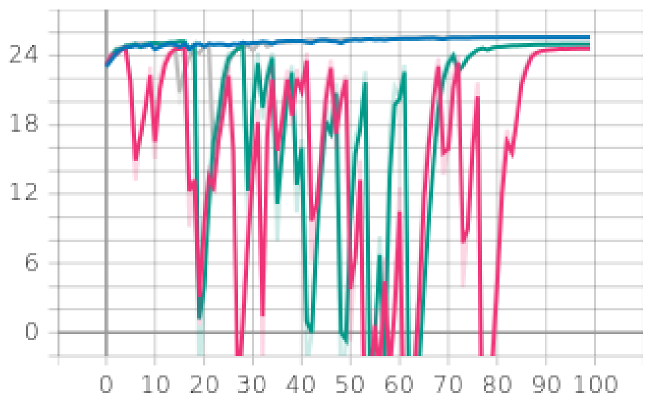

Figure 8: PSNR

Figure 9: Pink: 0.3. Green: 0.5, Blue: 0.7, Grey: 0.9

2.4. Upsampling

We either use

```
nn.Upsample(scale_factor=2, mode='nearest')
```

or

```
nn.ConvTranspose2d(48, 48, 3, stride=2, padding=1, output_padding=1)
nn.ConvTranspose2d(96, 96, 3, stride=2, padding=1, output_padding=1)
```

to perform upsampling. See Figure 12. We see no performance difference, but ”nn.Upsample” do not have learnable parameters and hence more efficient in forward computation and backpropgating. Hence, we adopt the default as ”nn.Upsample”.

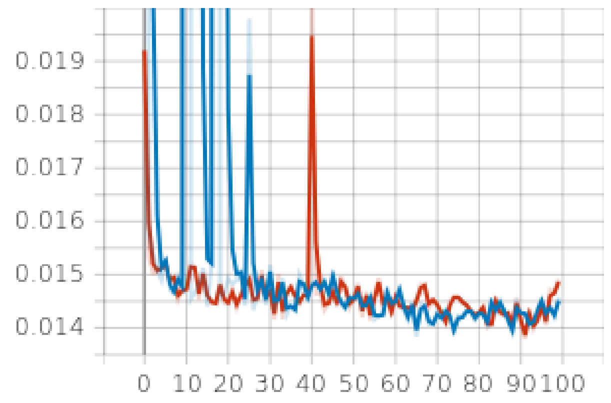

Figure 10: Training Loss

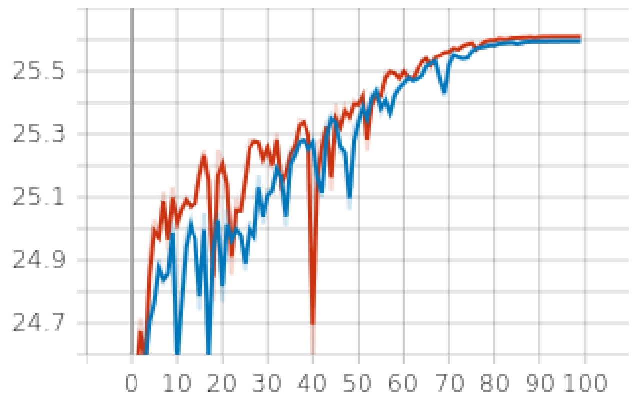

Figure 11: PSNR

Figure 12: Blue: Upsample. Red: ConvTranspose2d

3. Conclusion

Using default settings:

```
python model.py
```

We are able to train a model with PSNR ≈ 25.6.

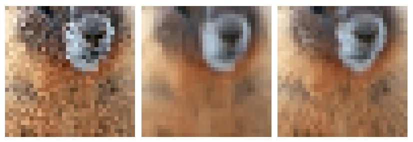

Figure 13: Example 1

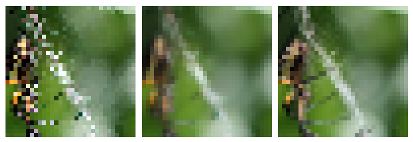

Figure 14: Example 2

Figure 15: Left: input. Middle: Denoised. Right: Target.

References

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention, pp. 234–241. Springer, 2015.