DLP2023 Lab4 Report

Introduction

In this lab, we are going to implement a conditional vidoe prediction with ***Conditional Variational Auto Encoder (CVAE)*** . With a given frame, and a sequence of poses, we want our model able to generate video sequence that the person in the frame doing the similar pose as the the given poses.

Therefore, in training step, we input a sequence of video and correponding frame to make our model learn the how to generate the next frame. And because the model need to generate a continous sequence video, we use ***Variational Auto Encoder*** instead of ***Auto Encoder***, where VAE will add a noise to the model, which make VAE able to generate a distribution instead of discrete variables.

Implementation Details

Training protocol

I. Coarse version of training

In training protocol, there will 16 frames as a training video sequence, and 16 corresponding pose as conditional labels. Starting from the first frame (***current_frame***) and the second label (***next_pose***), we want the model learn that the frame distribution from the first frame but with the second pose. So first, we sent current_frame and next_pose to two encoders seperately, and get ***encoded_current_frame*** and ***encoded_next_pose***.

```
encoded_current_frame = self.frame_transformation(current_frame)
encoded_next_pose = self.label_transformation(next_pose)
```

Second, we sent encoded_current_frame, encoded_next_pose, and a param(***z***), which I will explain later, to a decoder, and get ***decoded_features***. Then we sent the decoded_features to the generator to generate the ***predicted_ next_frame***.

```
decoded_features = self.Decoder_Fusion(encoded_current_frame,
encoded_next_pose, z)
predicted_next_frame = self.Generator(decoded_features)
```

So, we get the ***predicted next frame*** generated by the model. Similarly, use the generated frame(***predicted next frame***) and the third pose as the new inputs to train the model, and so on.

II. Loss

To train the model, there are two losses to calculate the gradients:

The first one is ***MSE loss (Reconstruction loss)*** , which equals to calculate the MSE loss of ***predicted next frame*** and ***ground truth next frame***.

```
mse_loss += self.mse_criterion(predicted_next_frame, next_frame)
```

The second one is ***KL Divergence***, which wants the latent distribution the encoder learned is as close as prior distribtion. We get the KL Divergence by sending next_pose and next_frame to two encoders sperately, and get ***encoded_next_pose*** and ***encoded_next_frame***. Then, send them to a posterior predictor to get the ***mean*** and ***log variance***. And use mean and variance to calculate the KL Divergence.

```
encoded_next_frame = self.frame_transformation(next_frame)
encoded_next_pose = self.label_transformation(next_pose)
z, mu, logvar = self.Gaussian_Predictor(encoded_next_frame,
encoded_next_pose) # z from reparameterization trick
kl_loss += kl_criterion(mu, logvar, self.batch_size)
```

And we get our final loss, where beta is the ratio from kl_annealing:

```
beta = self.kl_annealing.get_beta()
loss = mse_loss + beta * kl_loss
```

for each batch, I use the loss to compute the gradient, then clean up the gradients.

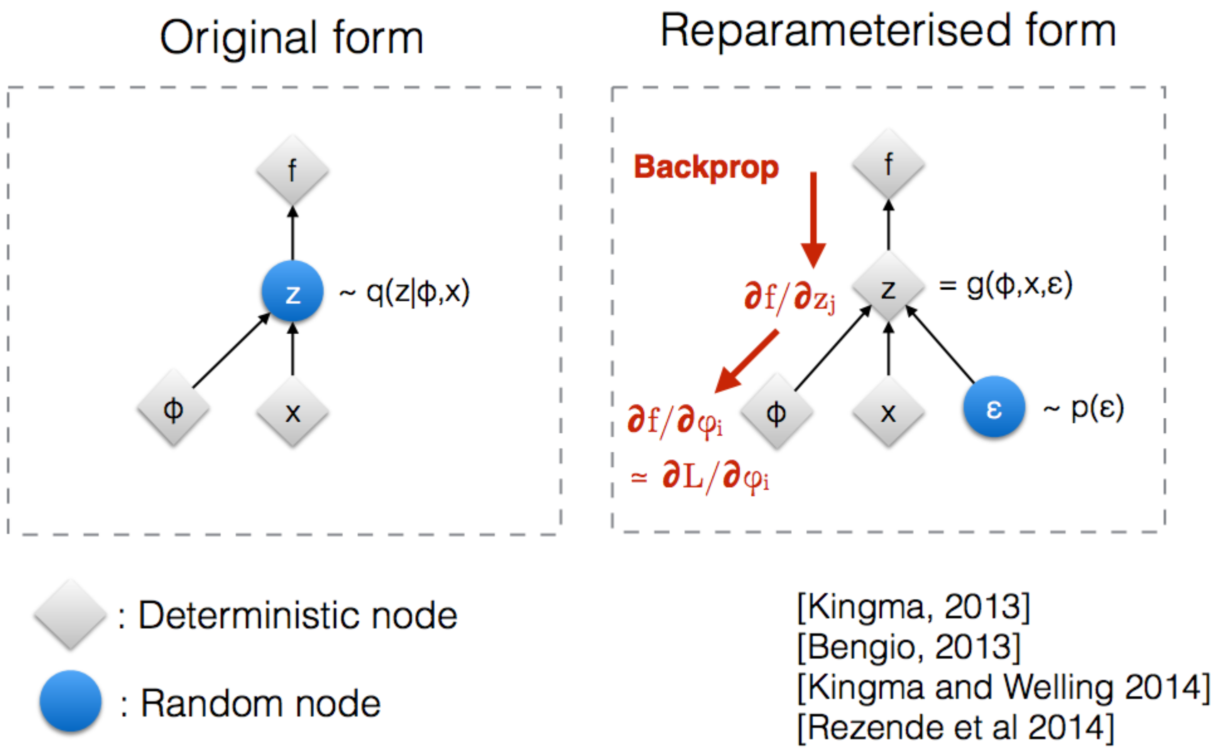
```
loss.backward()
self.optimizer_step()
self.optim.zero_grad()
```

Reparametrization tricks

Let's explain the param z. We calculate z by mean, standard deviation from posterior predictor, and a normal distribtion.

```
std = torch.exp(0.5 * logvar)
epsilon = torch.randn_like(std)
z = mu + std * epsilon
```

And why we need to do this parameteriztion trick? Originally, the latent variables takes sample from mean, and log variance. and we feeds this samples to the decoder. But the problem is we cannot run back propagation through a smapled latent variable. So, to train the model end-to-end, we represent the latent variables as: `z = mu + std * epsilon`, where epsilon is a normal distribution. So now, we dont have a whole stochastic latent variables, the only stochastic part is the epsilon, and we don't need to run backpropagation through it.



Alt text

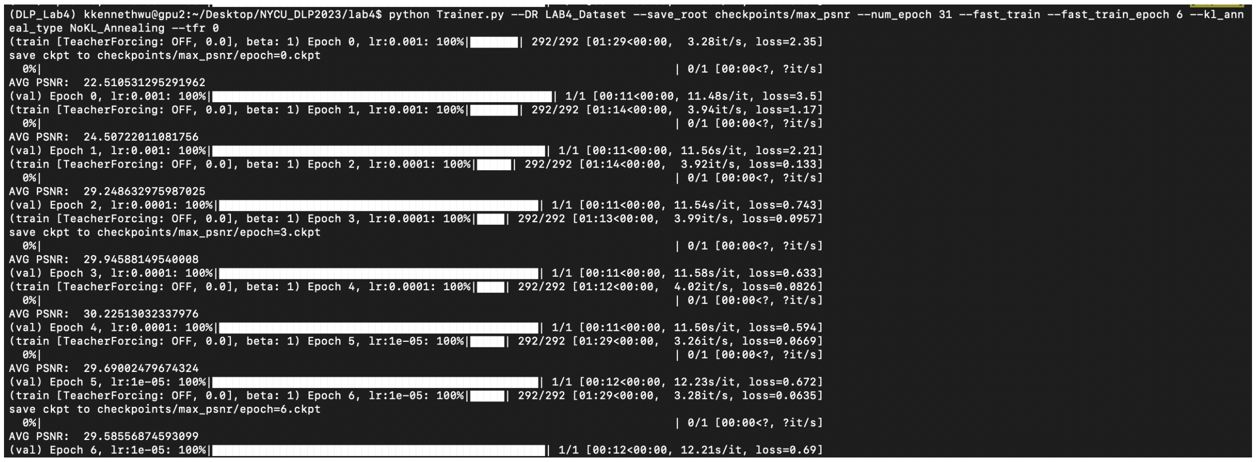
Setting of teacher forcing strategy

For different KL Annealing type, I use different teacher forcing strategy.

- Cyclical/Monotonic KL Annealing I set the teacher forcing ratio starting at 1, and start decaying when epoch meet sde, and gradually decay to 0 at the end of training. Without this strategy, I found taht the loss will be ***nan*** at first, becuase we don't have KL loss at first (beta=0). And this will make our model training failed.
- No KL Annealing In all training stage, I set the teacher forcing ratio = 0. This setting will make the validation/testing PSNR over 27 with just 6 epoch.

KL annealing ratio

My best val/test PSNR can meet 30 at just 4 epoch, when I don't use KL Annealoing i.e KL Annealing ratio equal to 1 all the training stage, and without teacher forcing strategy, and --fast-train. I thought the reason why this training strategy will get great score is that the encoder may learn the prior distribtion early and I also use --fast train.



```
(DLP_Lab4) kkennethwu@gpu2:~/Desktop/NYCU_DLP2023/lab4$ python Trainer.py --DR LAB4_Dataset --save_root checkpoints/max_psnr --num_epoch 31 --fast_train --fast_train_epoch 6 --kl_ann
eal_type NoKL_Annealing --tfr 0
(train [TeacherForcing: OFF, 0.0], beta: 1) Epoch 0, lr:0.001: 100%|███████| 292/292 [01:29<00:00,  3.28it/s, loss=2.35]
save ckpt to checkpoints/max_psnr/epoch=0.ckpt
  0%|                                                                                                 | 0/1 [00:00<?, ?it/s]
AVG PSNR:  22.510531295291962
(val) Epoch 0, lr:0.001: 100%|████████████████████████████████████████████████| 1/1 [00:11<00:00, 11.48s/it, loss=3.5]
(train [TeacherForcing: OFF, 0.0], beta: 1) Epoch 1, lr:0.001: 100%|███████| 292/292 [01:14<00:00,  3.94it/s, loss=1.17]
  0%|                                                                                                 | 0/1 [00:00<?, ?it/s]
AVG PSNR:  24.50722011081756
(val) Epoch 1, lr:0.001: 100%|███████████████████████████████████████████████| 1/1 [00:11<00:00, 11.56s/it, loss=2.21]
(train [TeacherForcing: OFF, 0.0], beta: 1) Epoch 2, lr:0.0001: 100%|█████| 292/292 [01:14<00:00,  3.92it/s, loss=0.133]
  0%|                                                                                                 | 0/1 [00:00<?, ?it/s]
AVG PSNR:  29.248632975987025
(val) Epoch 2, lr:0.0001: 100%|█████████████████████████████████████████████| 1/1 [00:11<00:00, 11.54s/it, loss=0.743]
(train [TeacherForcing: OFF, 0.0], beta: 1) Epoch 3, lr:0.0001: 100%|████| 292/292 [01:13<00:00,  3.99it/s, loss=0.0957]
save ckpt to checkpoints/max_psnr/epoch=3.ckpt
  0%|                                                                                                 | 0/1 [00:00<?, ?it/s]
AVG PSNR:  29.94588149540008
(val) Epoch 3, lr:0.0001: 100%|█████████████████████████████████████████████| 1/1 [00:11<00:00, 11.58s/it, loss=0.633]
(train [TeacherForcing: OFF, 0.0], beta: 1) Epoch 4, lr:0.0001: 100%|████| 292/292 [01:12<00:00,  4.02it/s, loss=0.0826]
  0%|                                                                                                 | 0/1 [00:00<?, ?it/s]
AVG PSNR:  30.22513032337976
(val) Epoch 4, lr:0.0001: 100%|█████████████████████████████████████████████| 1/1 [00:11<00:00, 11.50s/it, loss=0.594]
(train [TeacherForcing: OFF, 0.0], beta: 1) Epoch 5, lr:1e-05: 100%|█████| 292/292 [01:29<00:00,  3.26it/s, loss=0.0669]
  0%|                                                                                                 | 0/1 [00:00<?, ?it/s]
AVG PSNR:  29.69002479674324
(val) Epoch 5, lr:1e-05: 100%|██████████████████████████████████████████████| 1/1 [00:12<00:00, 12.23s/it, loss=0.672]
(train [TeacherForcing: OFF, 0.0], beta: 1) Epoch 6, lr:1e-05: 100%|█████| 292/292 [01:29<00:00,  3.28it/s, loss=0.0635]
save ckpt to checkpoints/max_psnr/epoch=6.ckpt
  0%|                                                                                                 | 0/1 [00:00<?, ?it/s]
AVG PSNR:  29.58556874593099
(val) Epoch 6, lr:1e-05: 100%|███████████████████████████████████████████████| 1/1 [00:12<00:00, 12.21s/it, loss=0.69]
```

Alt text

Because the model will converge earlier with

And because of --fast-train, when the model start training with whole training data, the PSNR will decrease. But gradually, the PSNR will back to 30 at epoch 17, and keep increasing.

Analysis & Discussion

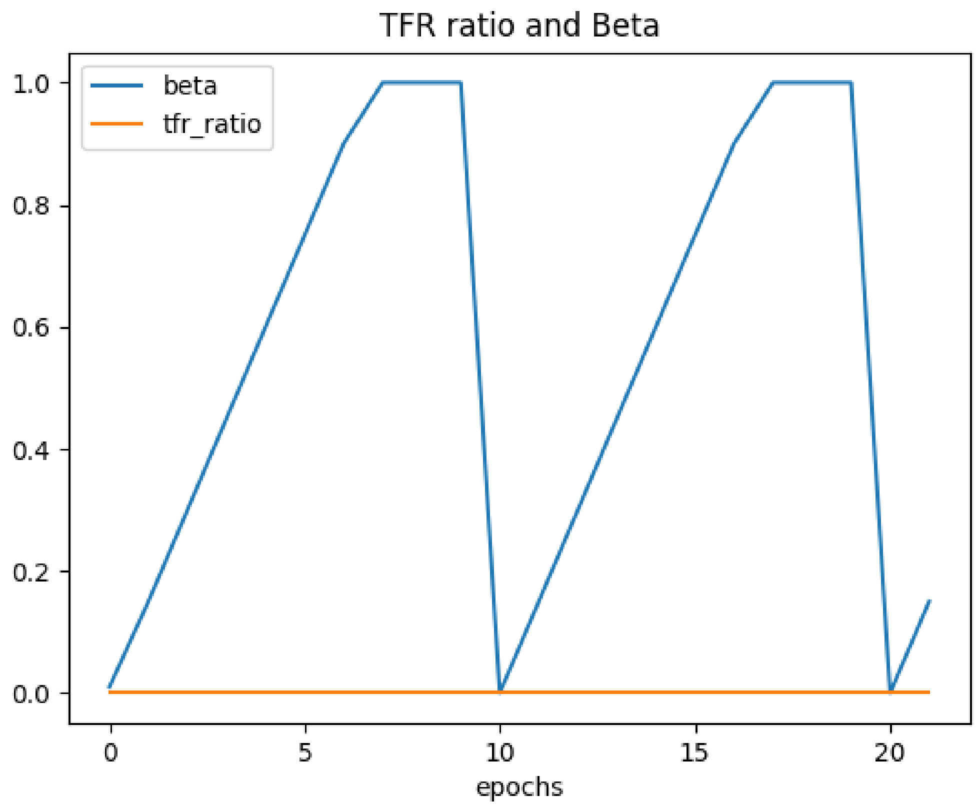
After a plenty of experiments, I found that the ratio of ***tfr ratio*** and ***beta*** plays a great role in training model. The tfr ratio and beta both represent the information from groundtruth label, because the tfr ratio represent how many the groundtruth frame feeding to the generator model, and the beta represent how much information from prior distribution adding to the loss. So, it can be thought that we cannot add to many ground truth information to the model. We also need to add some bad information (from predicted_next_frame mentioned before).

Plot teacher forcing ratio

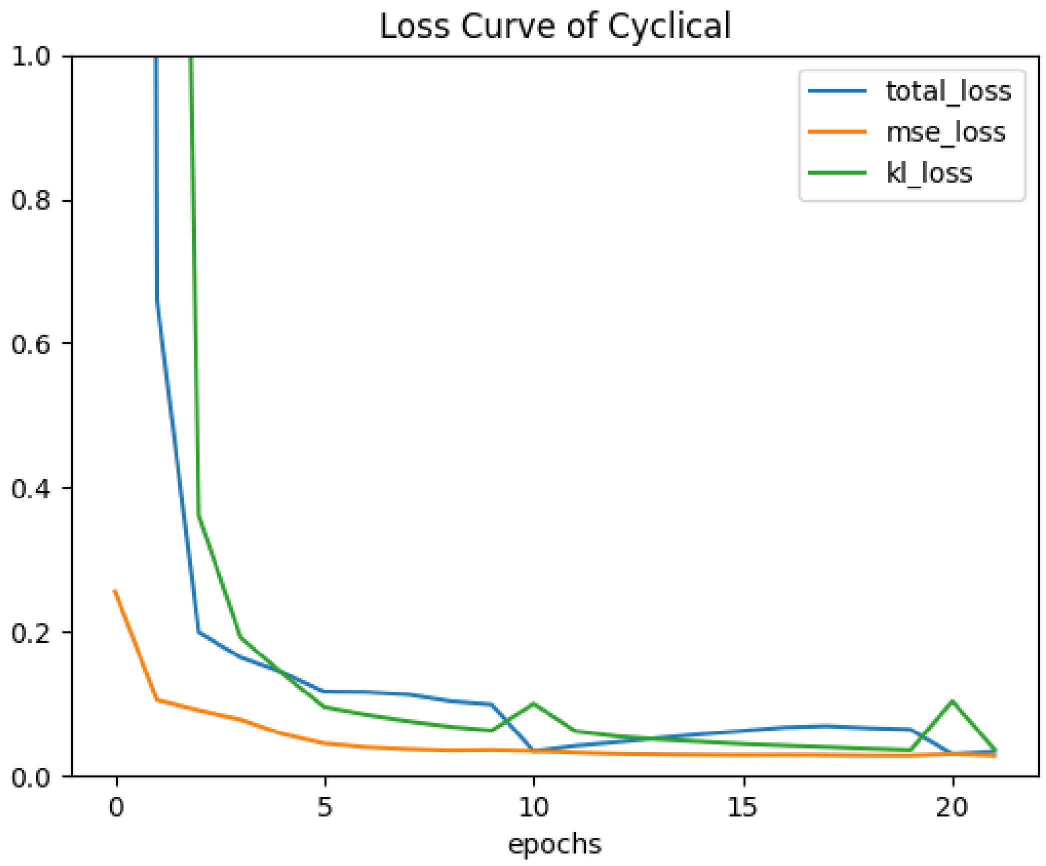
Here I use kl anneal type cyclical to experiment. I inital the beta=0.01 to make sure there are some KL loss for bad information at first. Otherwise, if the beta starting at 0, we will meet the gradient vanishing problem, and the loss will be nan. This will cause the model to fail to be trained for all subsequent epochs. Also, I use `--fast_tain` in the whole training stage to acclerate training, i.e just training with partial training data. And I use `--batch-size=4` to accelerate training, which will make the performance a little bit worse compared to `--batch_size=2`

- [Case A] tfr = 0

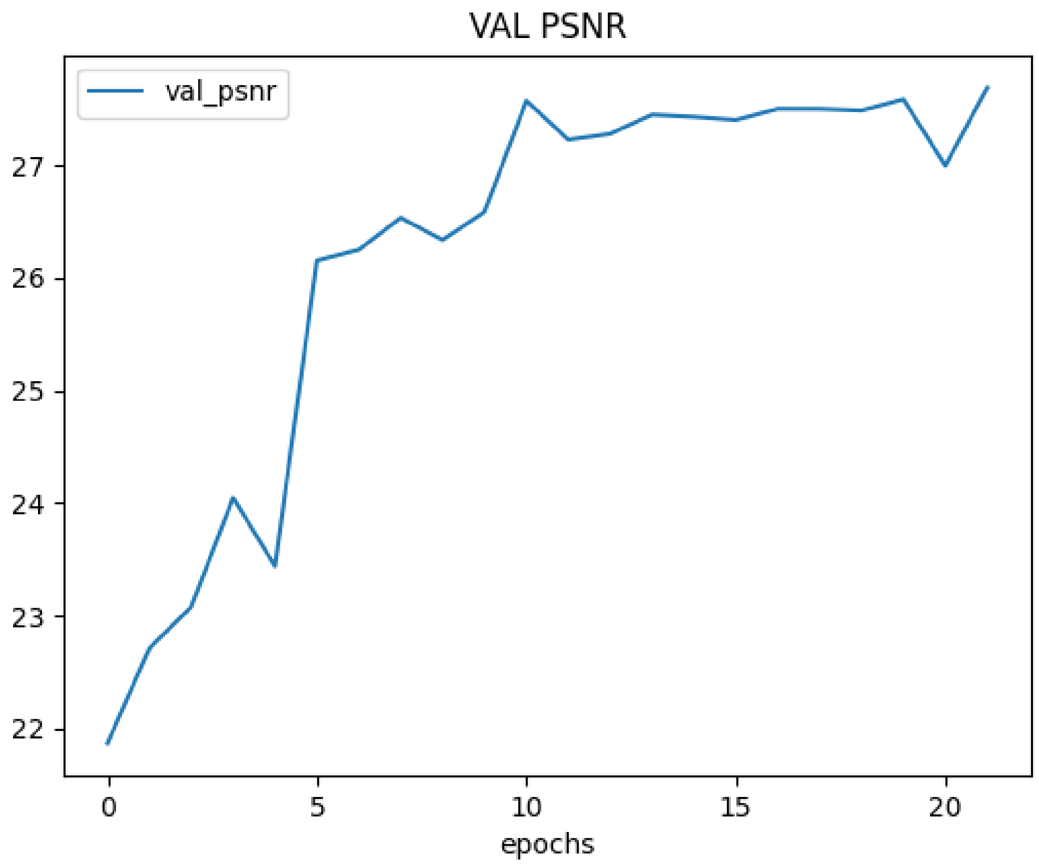
```
python Trainer.py --DR LAB4_Dataset/ --save_root
checkpoints/tfr_compare_tfr0_Cyclical --tfr 0 --tfr_sde
5 --kl_anneal_type Cyclical --kl_anneal_ratio 1.5
--num_epoch 22 --fast_train --fast_train_epoch 22
--batch_size 4
```



Alt text


Loss Curve of Cyclical
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
epochs
total_loss
mse_loss
kl_loss

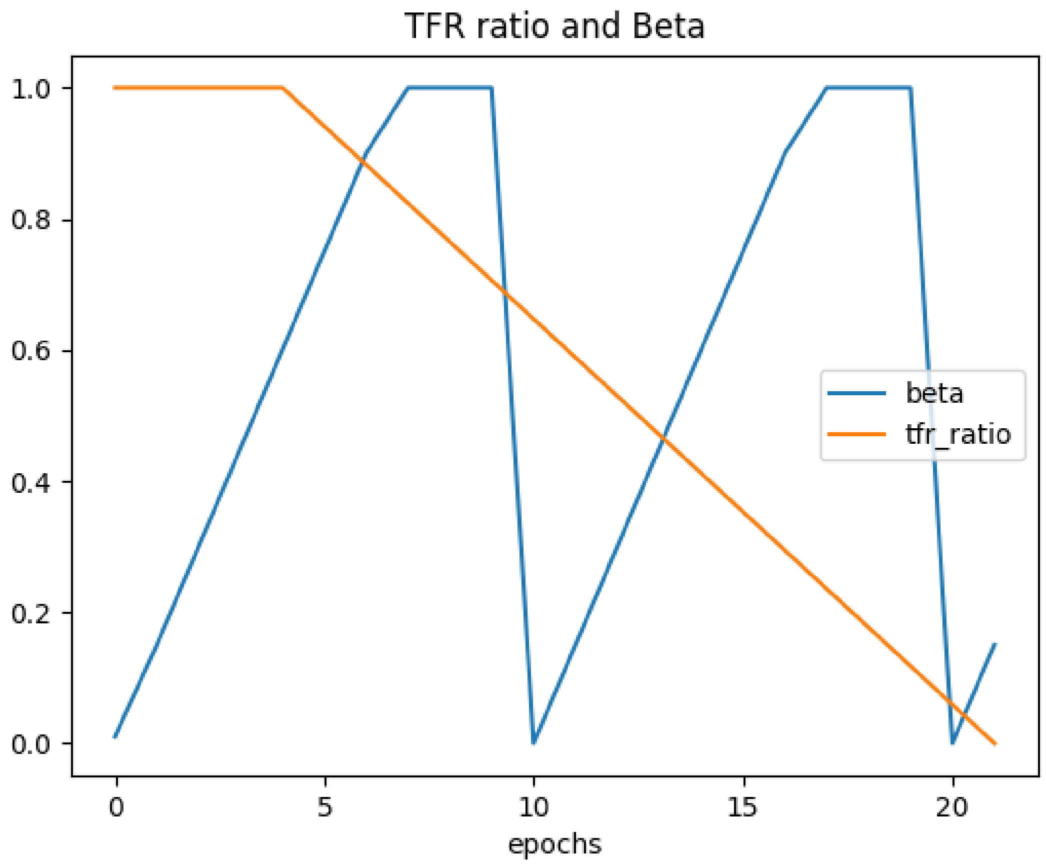
Alt text



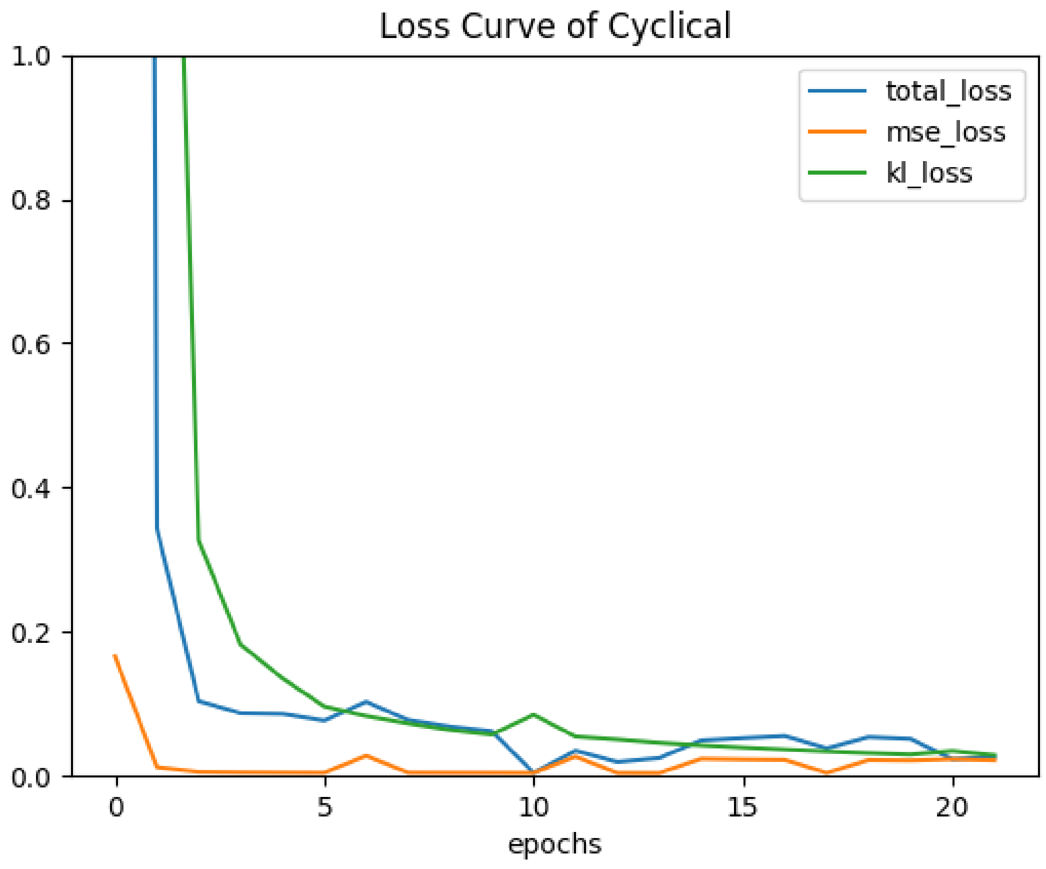
Alt text

- [CASE B] tfr = 1, and start decaying at epoch 5

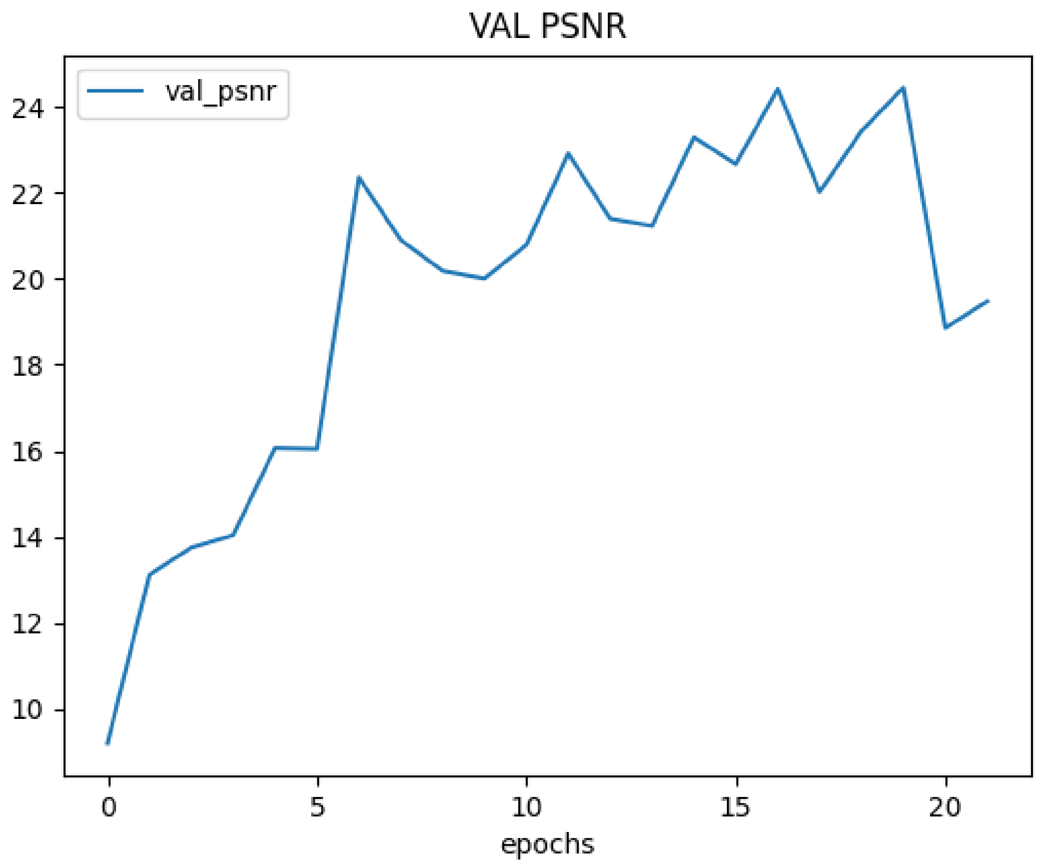
```
python Trainer.py --DR LAB4_Dataset/ --save_root
checkpoints/tfr_compare_tfr1_Cyclical --tfr 1 --tfr_sde
5 --kl_anneal_type Cyclical --kl_anneal_ratio 1.5
--num_epoch 22 --fast_train --fast_train_epoch 22
--batch_size 4
```


TFR ratio and Beta
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
epochs
beta
tfr_ratio

Alt text


Loss Curve of Cyclical
total_loss
mse_loss
kl_loss
epochs
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20

Alt text



Alt text

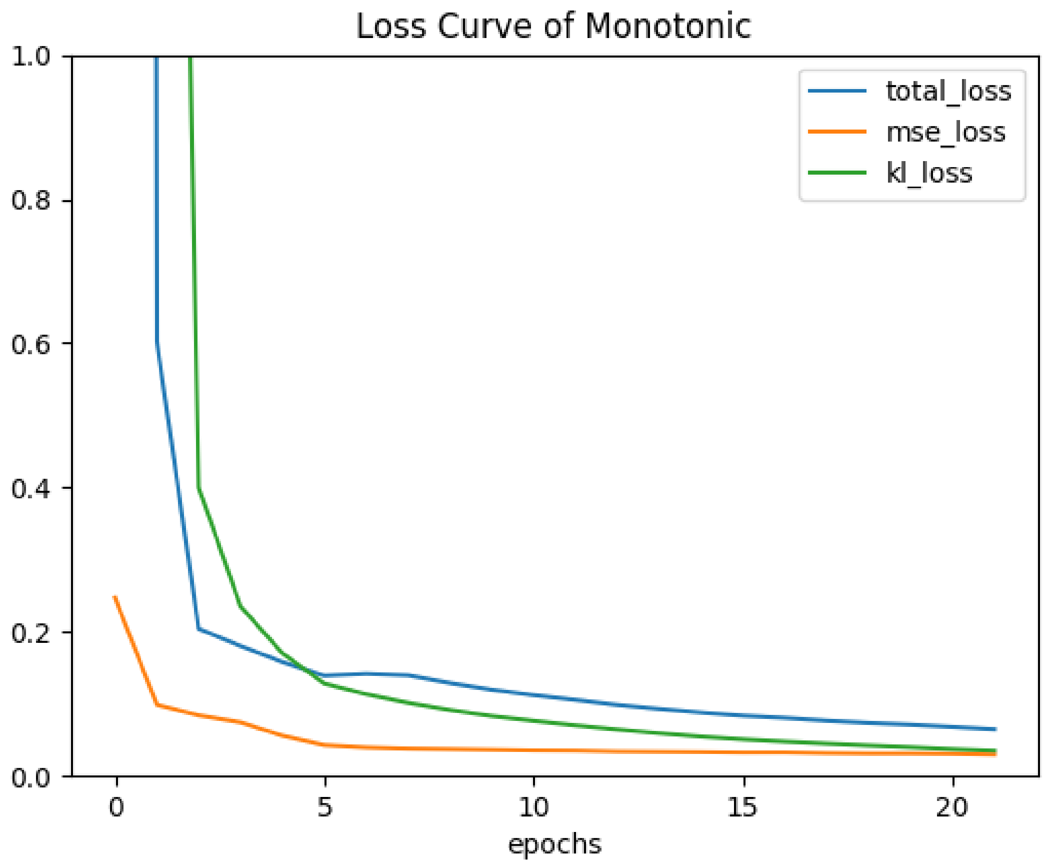
We can see that the [CASE A] has better performance becuase we feed good information and bad information to the model. On the other hand, [CASE B] have too grountruth information at begining, causing the bad performance.

Plot loss curve while training with different setting

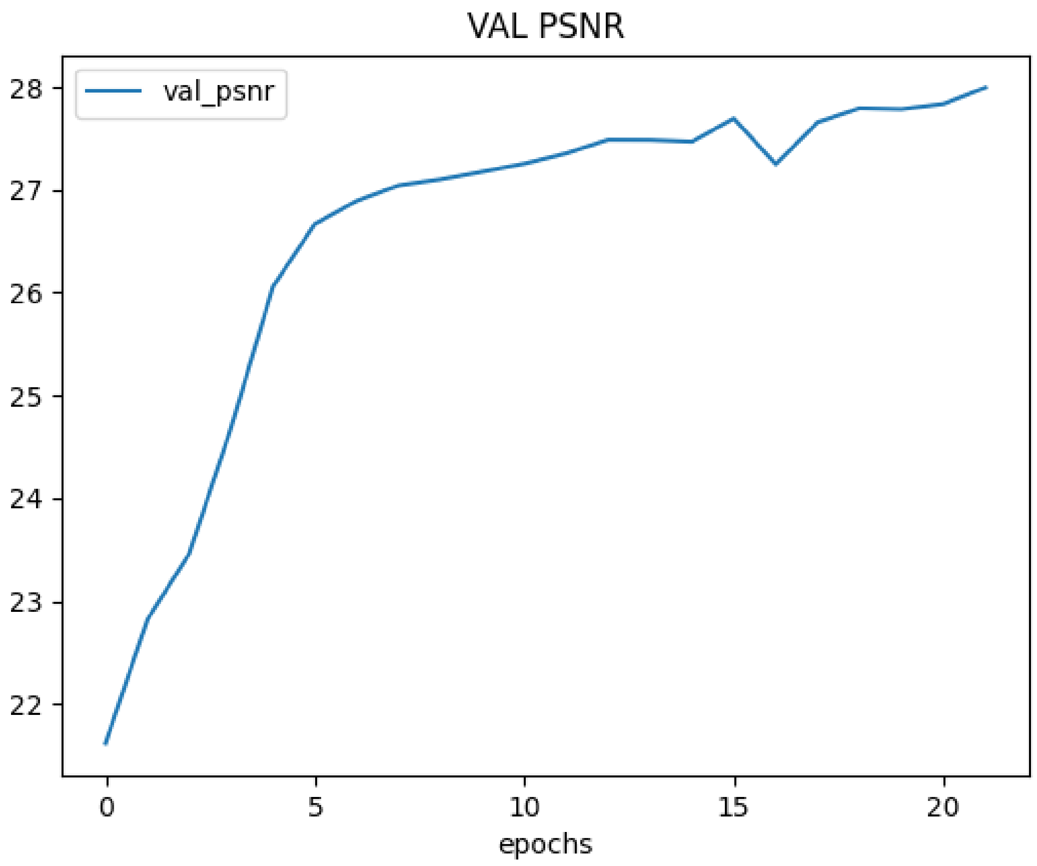
I test the performance of different kl anneal type with tfr=0. Also, I use `--fast_tain` in the whole training stage to acclerate training, i.e just training with partial training data. And I use `--batch-size=4` to accelerate training, which will make the performance a little bit worse compared to `--batch_size=2`

- With KL annealing (Monotonic), beta starting from 0.01

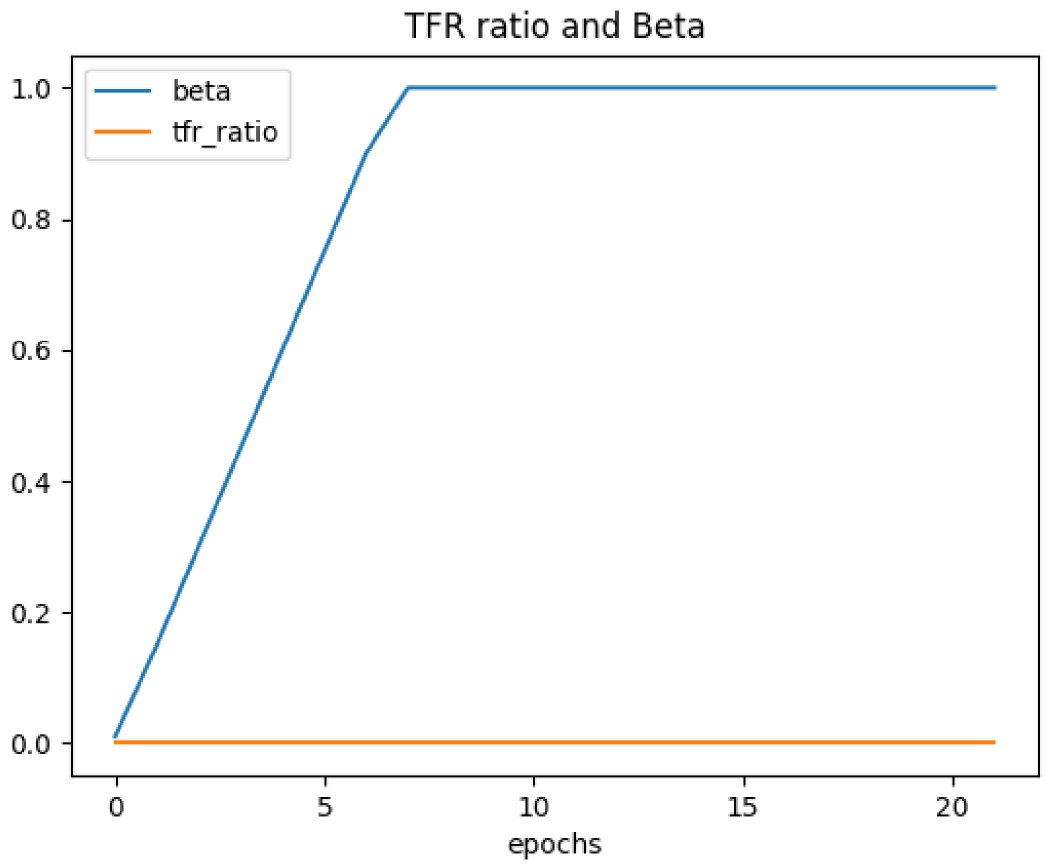
```
python Trainer.py --DR LAB4_Dataset/ --save_root
checkpoints/KL_compare_tfr0_Monotonic --tfr 0 --tfr_sde
5 --kl_anneal_type Monotonic --kl_anneal_ratio 1.5
--num_epoch 22 --fast_train --fast_train_epoch 22
--batch_size 4
```


Loss Curve of Monotonic
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
epochs
total_loss
mse_loss
kl_loss

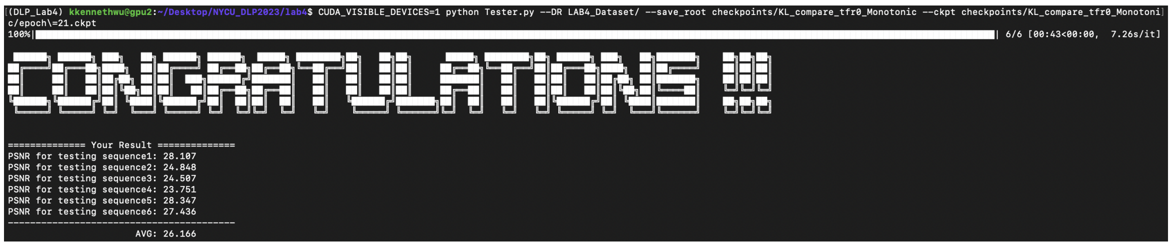
Alt text



Alt text



Alt text

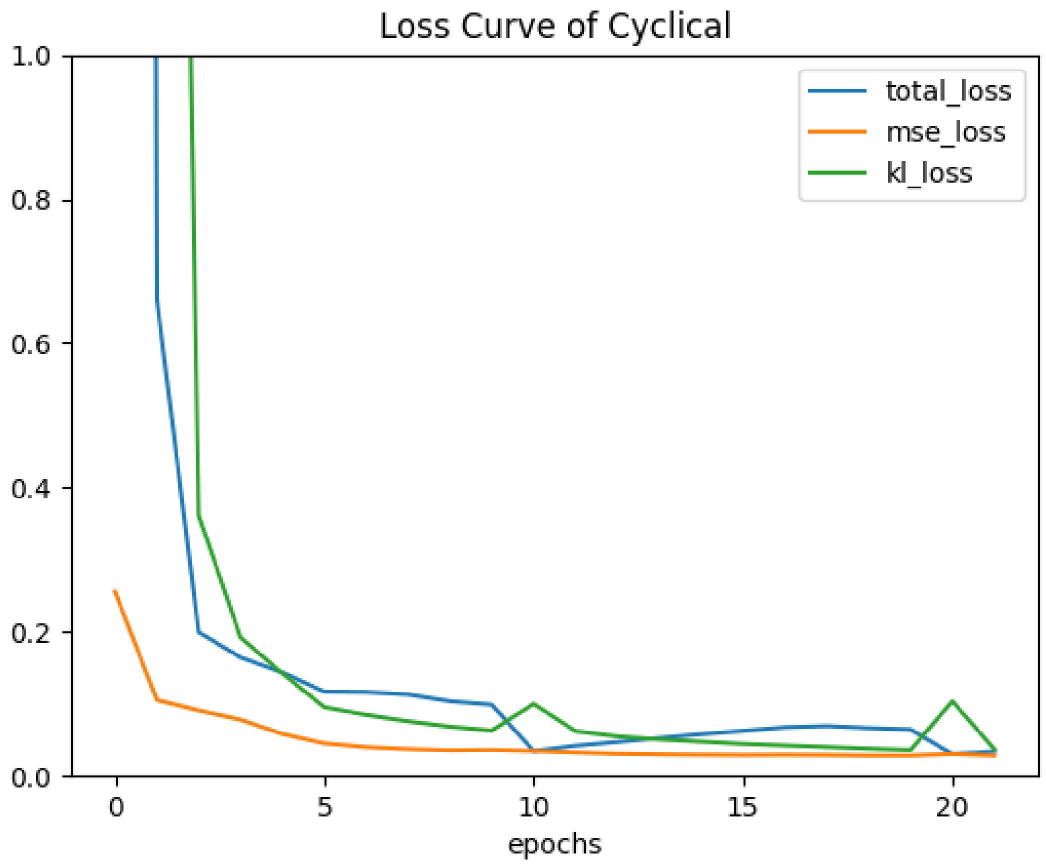


```
(DLP_Lab4) kkennethwu@gpu2:~/Desktop/NYCU_DLP2023/lab4$ CUDA_VISIBLE_DEVICES=1 python Tester.py --DR LAB4_Dataset/ --save_root checkpoints/KL_compare_tfr0_Monotonic --ckpt checkpoints/KL_compare_tfr0_Monotoni
c/epoch\=21.ckpt
100%|████████████████████████████████████████████████████████████| 6/6 [00:43<00:00,  7.26s/it]

CONGRATULATIONS !!!

============== Your Result ==============
PSNR for testing sequence1: 28.107
PSNR for testing sequence2: 24.848
PSNR for testing sequence3: 24.507
PSNR for testing sequence4: 23.751
PSNR for testing sequence5: 28.347
PSNR for testing sequence6: 27.436
-----------------------------------------
                       AVG: 26.166
```

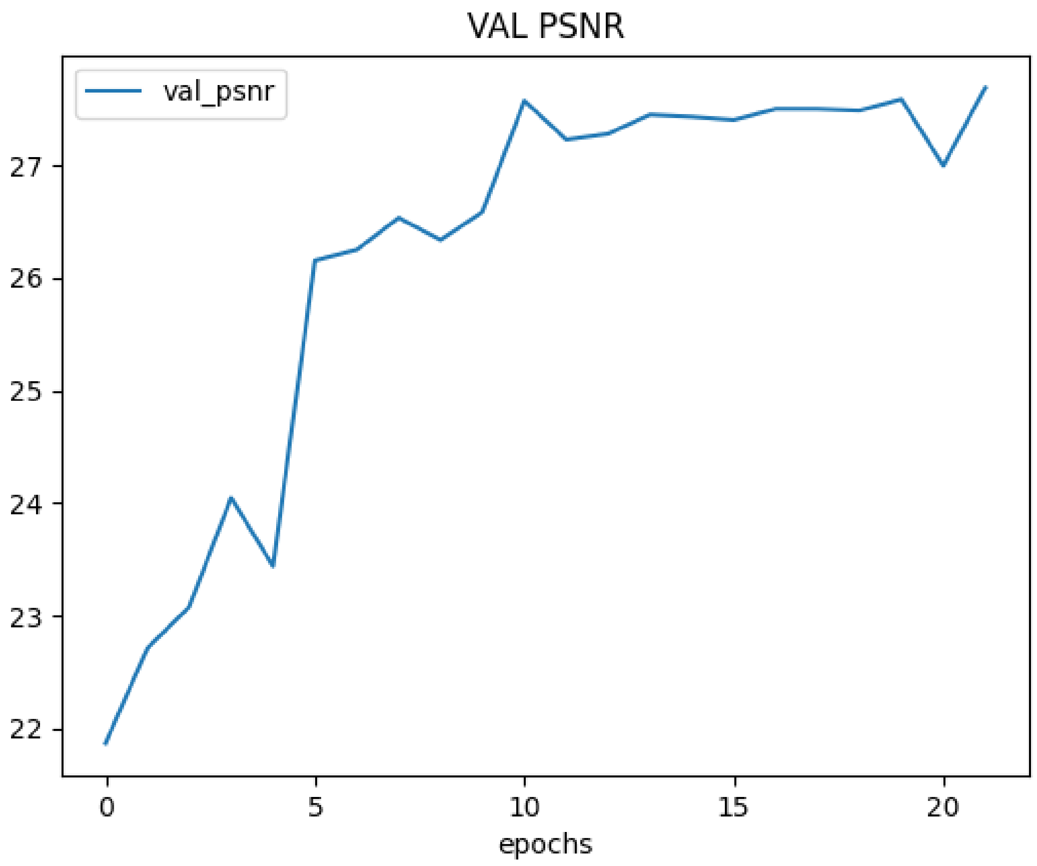
Alt text

- With KL annealing (Cyclical), beta startin from 0.01

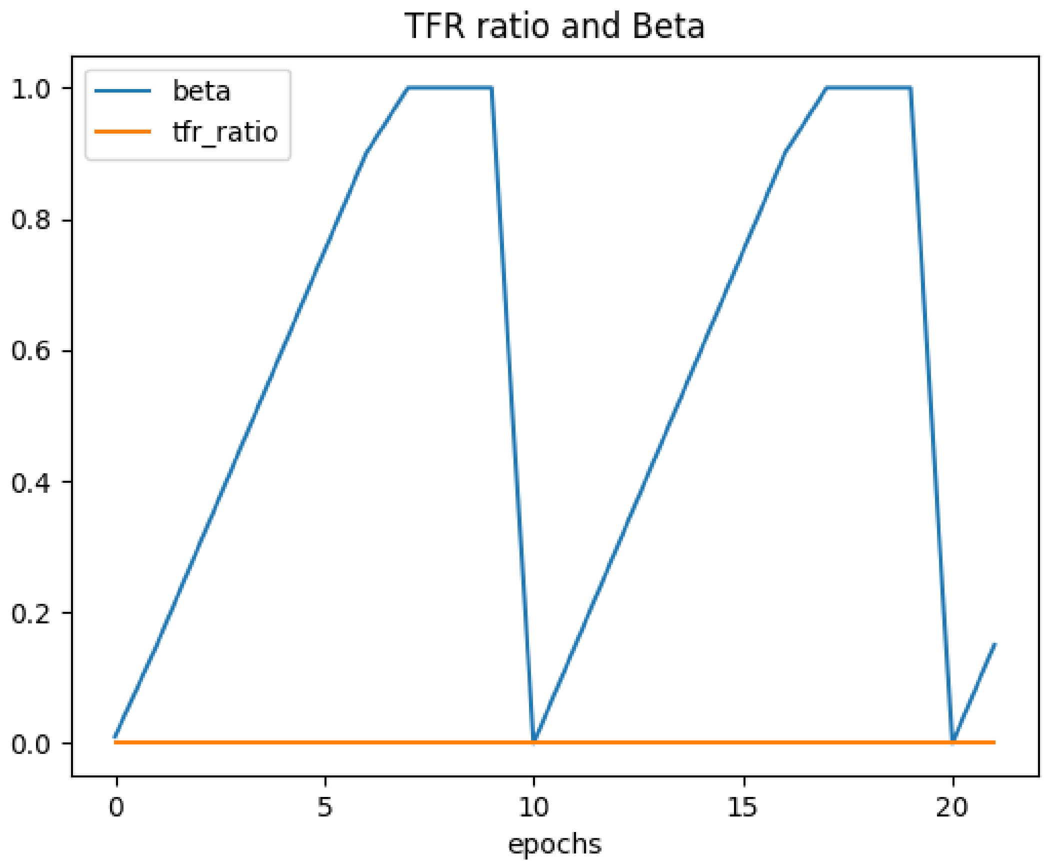
```
python Trainer.py --DR LAB4_Dataset/ --save_root
checkpoints/tfr_compare_tfr0_Cyclical --tfr 0 --tfr_sde
5 --kl_anneal_type Cyclical --kl_anneal_ratio 1.5
--num_epoch 22 --fast_train --fast_train_epoch 22
--batch_size 4
```


Loss Curve of Cyclical
total_loss
mse_loss
kl_loss
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
epochs

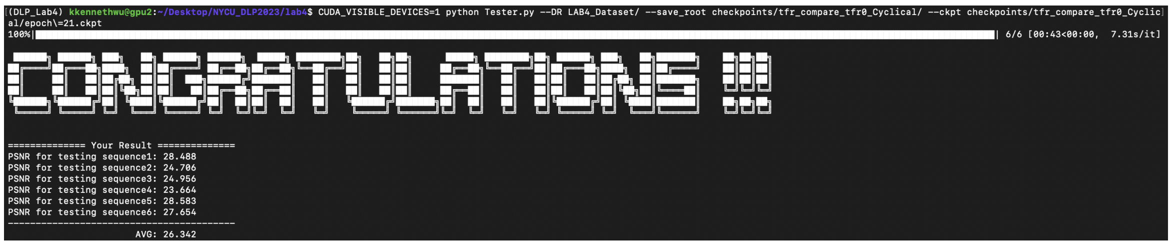
Alt text



Alt text



Alt text

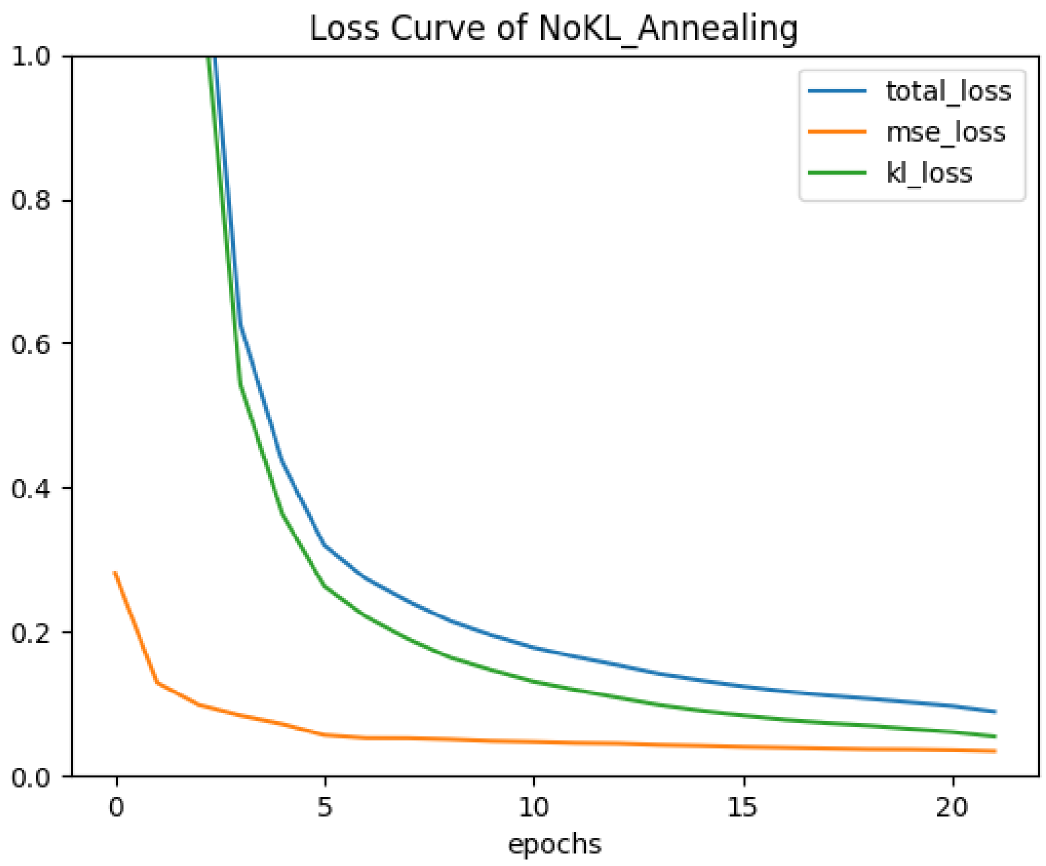


```
(DLP_Lab4) kkennethwu@gpu2:~/Desktop/NYCU_DLP2023/lab4$ CUDA_VISIBLE_DEVICES=1 python Tester.py --DR LAB4_Dataset/ --save_root checkpoints/tfr_compare_tfr0_Cyclical/ --ckpt checkpoints/tfr_compare_tfr0_Cyclic
al/epoch\=21.ckpt
100%|████████████████████████████████████████████████████████████████████████████| 6/6 [00:43<00:00,  7.31s/it]

CONGRATULATIONS !!!

============== Your Result ==============
PSNR for testing sequence1: 28.488
PSNR for testing sequence2: 24.706
PSNR for testing sequence3: 24.956
PSNR for testing sequence4: 23.664
PSNR for testing sequence5: 28.583
PSNR for testing sequence6: 27.654
-----------------------------------------
                     AVG: 26.342
```

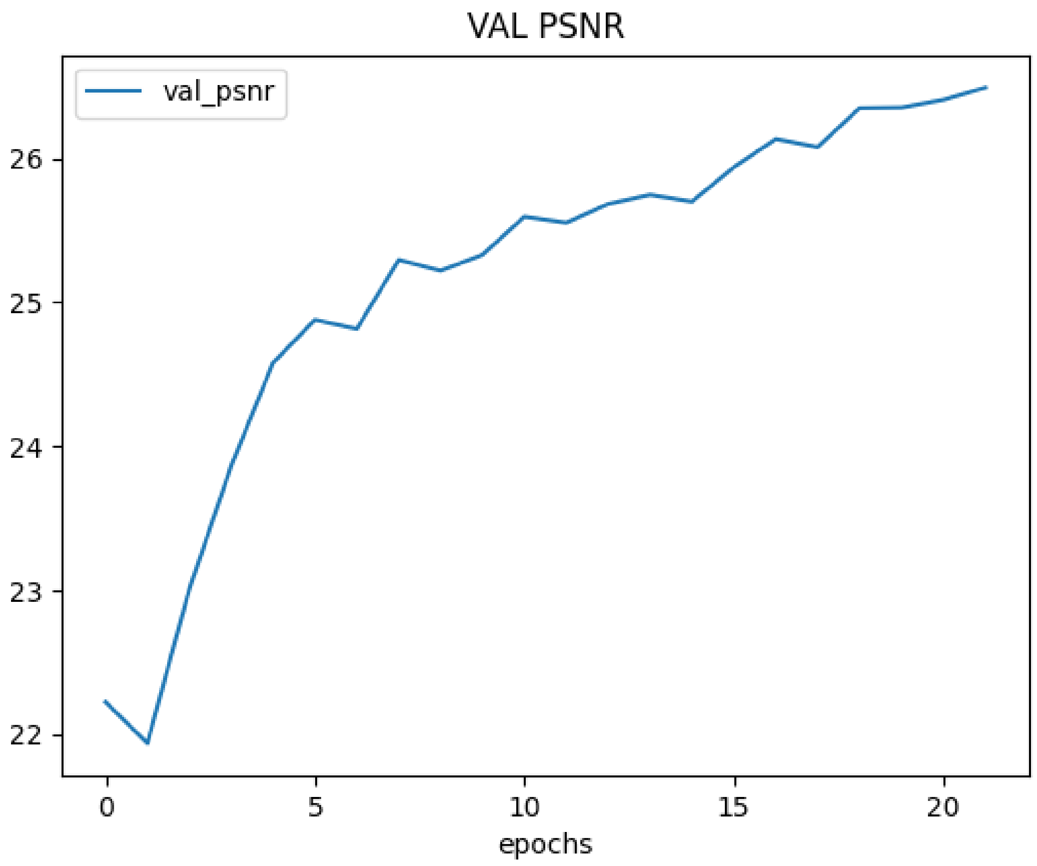
Alt text

- Without KL annealing, beta = 1

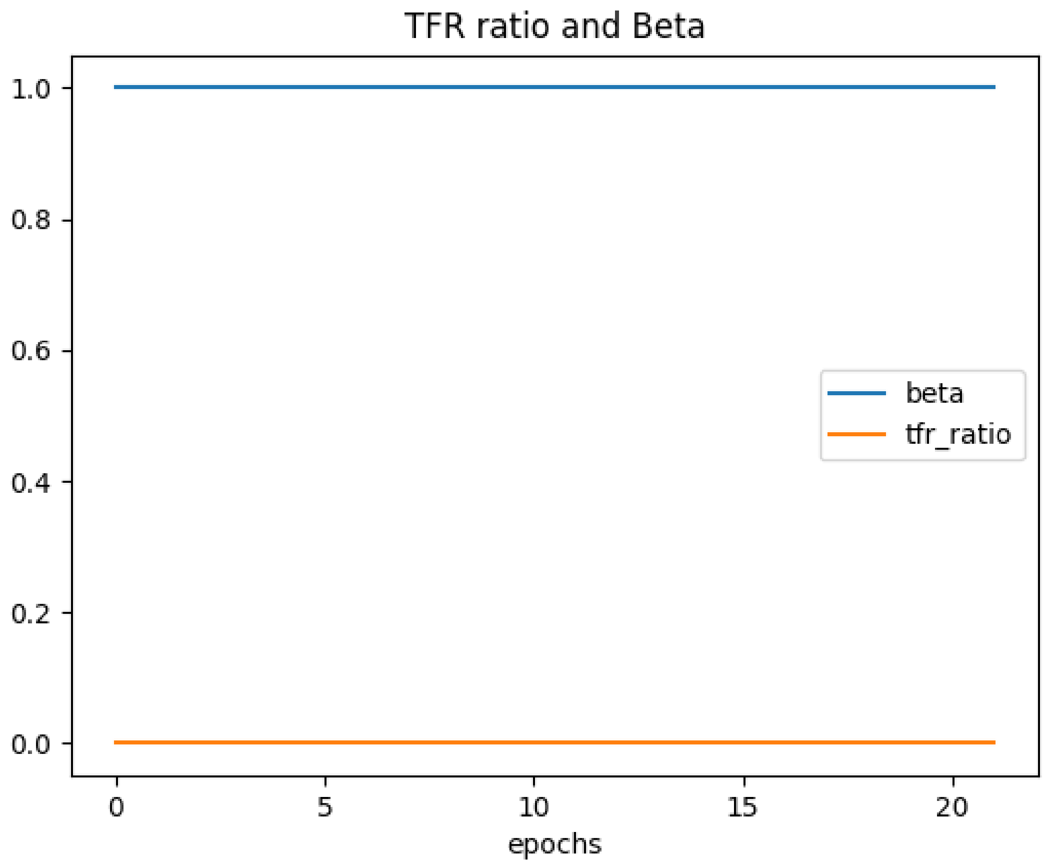
```
python Trainer.py --DR LAB4_Dataset/ --save_root
checkpoints/KL_compare_tfr0_NoKL --tfr 0 --tfr_sde 5
--kl_anneal_type NoKL_Annealing --kl_anneal_ratio 1.5
--num_epoch 22 --fast_train --fast_train_epoch 22
--batch_size 4
```


Loss Curve of NoKL_Annealing
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
epochs
total_loss
mse_loss
kl_loss

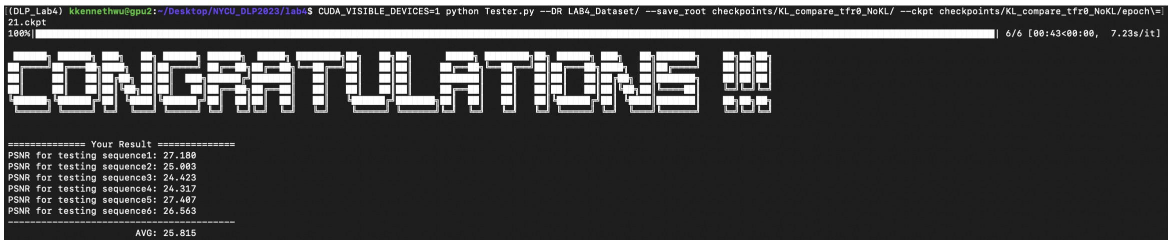
Alt text



Alt text



Alt text



```
(DLP_Lab4) kkennethwu@gpu2:~/Desktop/NYCU_DLP2023/lab4$ CUDA_VISIBLE_DEVICES=1 python Tester.py --DR LAB4_Dataset/ --save_root checkpoints/KL_compare_tfr0_NoKL/ --ckpt checkpoints/KL_compare_tfr0_NoKL/epoch\=
21.ckpt
100%|████████████████████████████████████████████████████████████████████████████████| 6/6 [00:43<00:00,  7.23s/it]

CONGRATULATIONS !!!

============== Your Result ==============
PSNR for testing sequence1: 27.180
PSNR for testing sequence2: 25.003
PSNR for testing sequence3: 24.423
PSNR for testing sequence4: 24.317
PSNR for testing sequence5: 27.407
PSNR for testing sequence6: 26.563
-----------------------------------------
                     AVG: 25.815
```

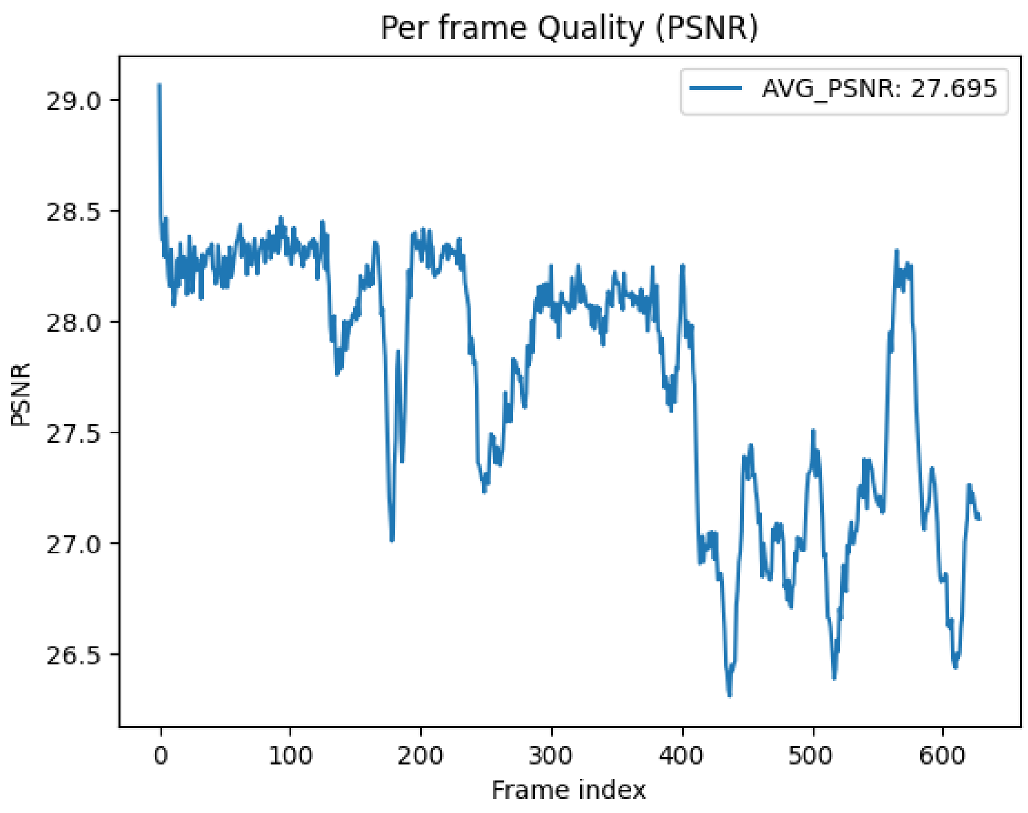
Alt text

For these three type of KL annealing, we starting from a similar validation PSNR because we both have a good portion of ground truth informations and bad informations. And we can see that the validation psnr will be not that smooth as the teacher forcing ratio will keep changing. They both reach a similar validation psnr. I thought the reason why they dont differ too much is the propotion of ground truth information and bad information is already good, so they can both reach a greate performance. So actually, I thought that the key role on the model performance is not what type of KL annealing we used, is that the propotion of groundtruth informations and bad informations we feed the model.

PSNR-per frame diagram in validation dataset

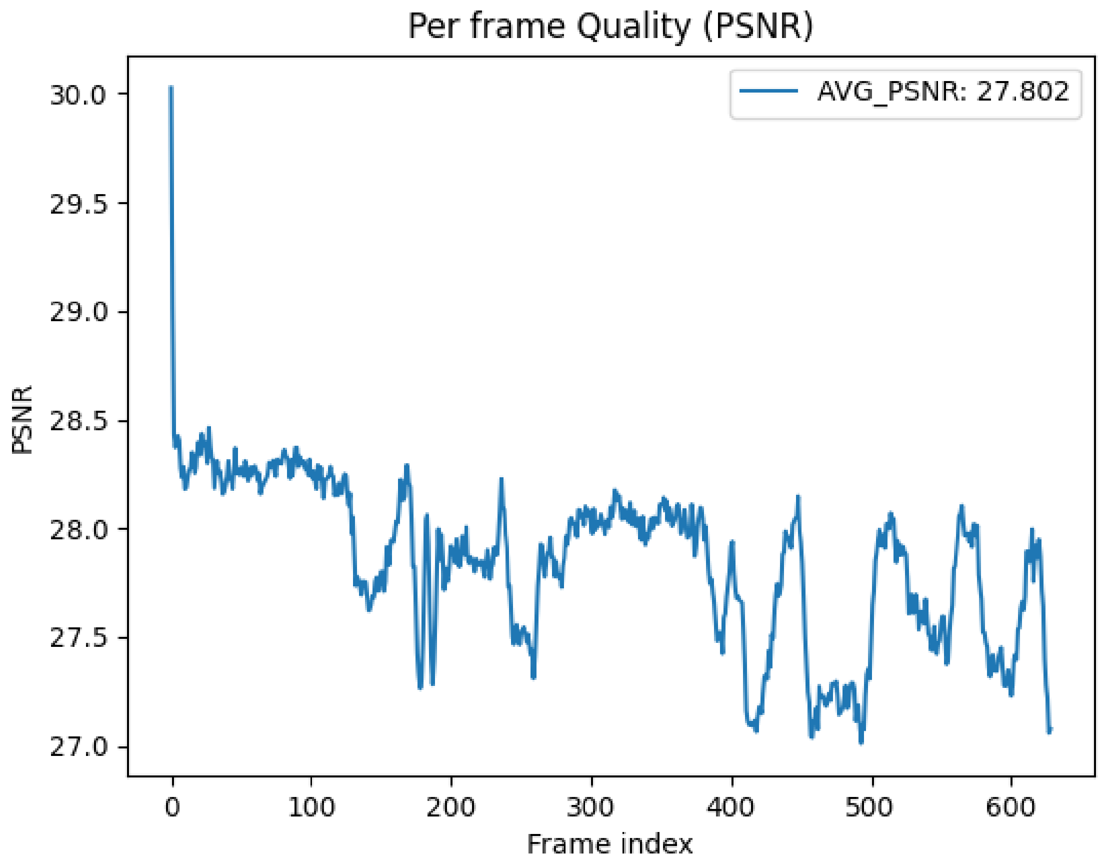
With 21 epoch and tfr = 0:

- With KL annealing (Monotonic), beta starting from 0.01



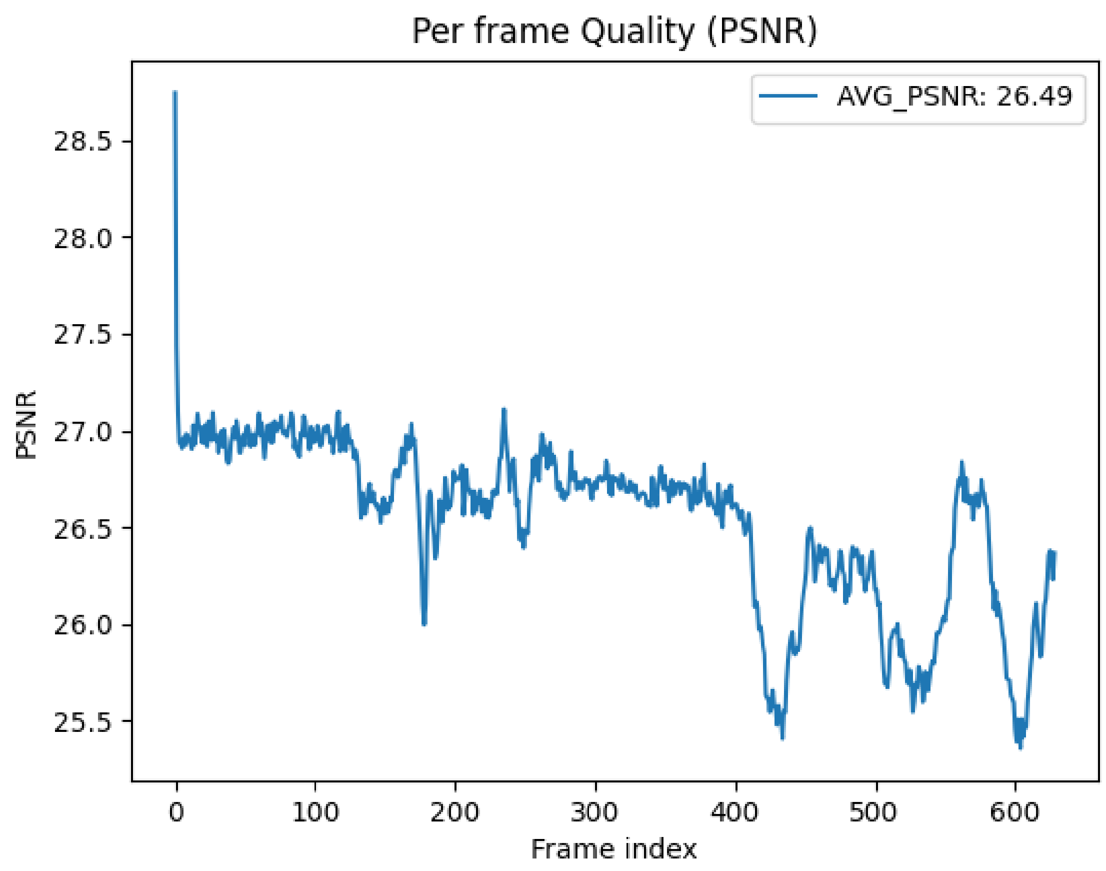
Alt text

- With KL annealing (Cyclical), beta startin from 0.01



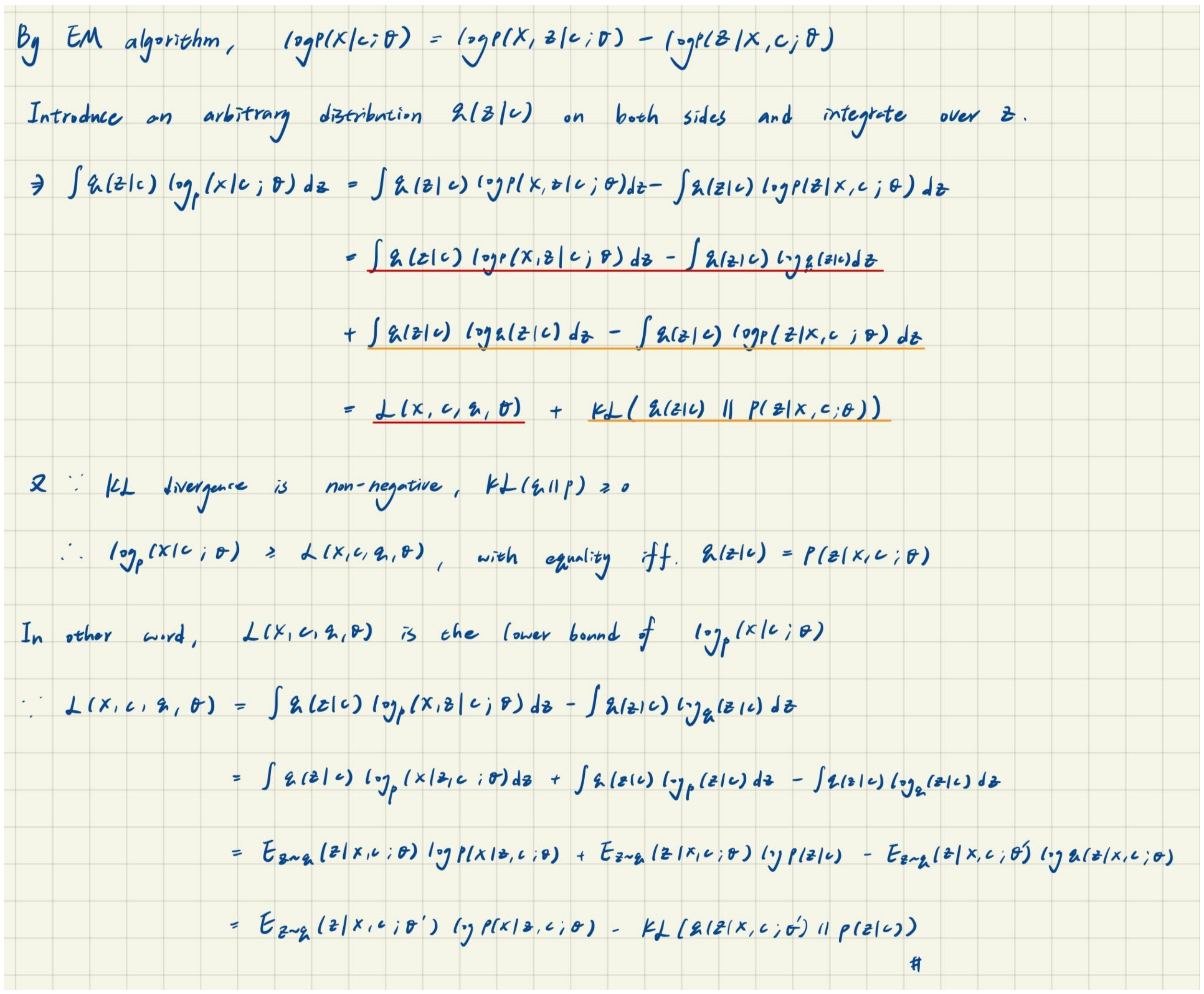
Alt text

- Without KL annealing, beta = 1



Alt text

Derivate conditional VAE formula (extra points 5%)



By EM algorithm, $\log p(x|c;\theta) = \log p(x,z|c;\theta) - \log p(z|x,c;\theta)$

Introduce an arbitrary distribution $q(z|c)$ on both sides and integrate over z.

$$\Rightarrow \int q(z|c)\log_p(x|c;\theta)\,dz = \int q(z|c)\log p(x,z|c;\theta)\,dz - \int q(z|c)\log p(z|x,c;\theta)\,dz$$

$$= \underline{\int q(z|c)\log p(x,z|c;\theta)\,dz - \int q(z|c)\log q(z|c)\,dz}$$

$$+ \underline{\int q(z|c)\log q(z|c)\,dz - \int q(z|c)\log p(z|x,c;\theta)\,dz}$$

$$= \underline{\mathcal{L}(x,c,q,\theta)} + \underline{KL(q(z|c)\,\|\,p(z|x,c;\theta))}$$

又 ∵ KL divergence is non-negative, $KL(q\|p) \geq 0$

∴ $\log_p(x|c;\theta) \geq \mathcal{L}(x,c,q,\theta)$, with equality iff. $q(z|c) = p(z|x,c;\theta)$

In other word, $\mathcal{L}(x,c,q,\theta)$ is the lower bound of $\log_p(x|c;\theta)$

∵ $\mathcal{L}(x,c,q,\theta) = \int q(z|c)\log_p(x,z|c;\theta)\,dz - \int q(z|c)\log_q(z|c)\,dz$

$$= \int q(z|c)\log_p(x|z,c;\theta)\,dz + \int q(z|c)\log_p(z|c)\,dz - \int q(z|c)\log_q(z|c)\,dz$$

$$= E_{z\sim q}(z|x,c;\theta)\log p(x|z,c;\theta) + E_{z\sim q}(z|x,c;\theta)\log p(z|c) - E_{z\sim q}(z|x,c;\theta')\log q(z|x,c;\theta)$$

$$= E_{z\sim q}(z|x,c;\theta')\log p(x|z,c;\theta) - KL(q(z|x,c;\theta')\,\|\,p(z|c))$$

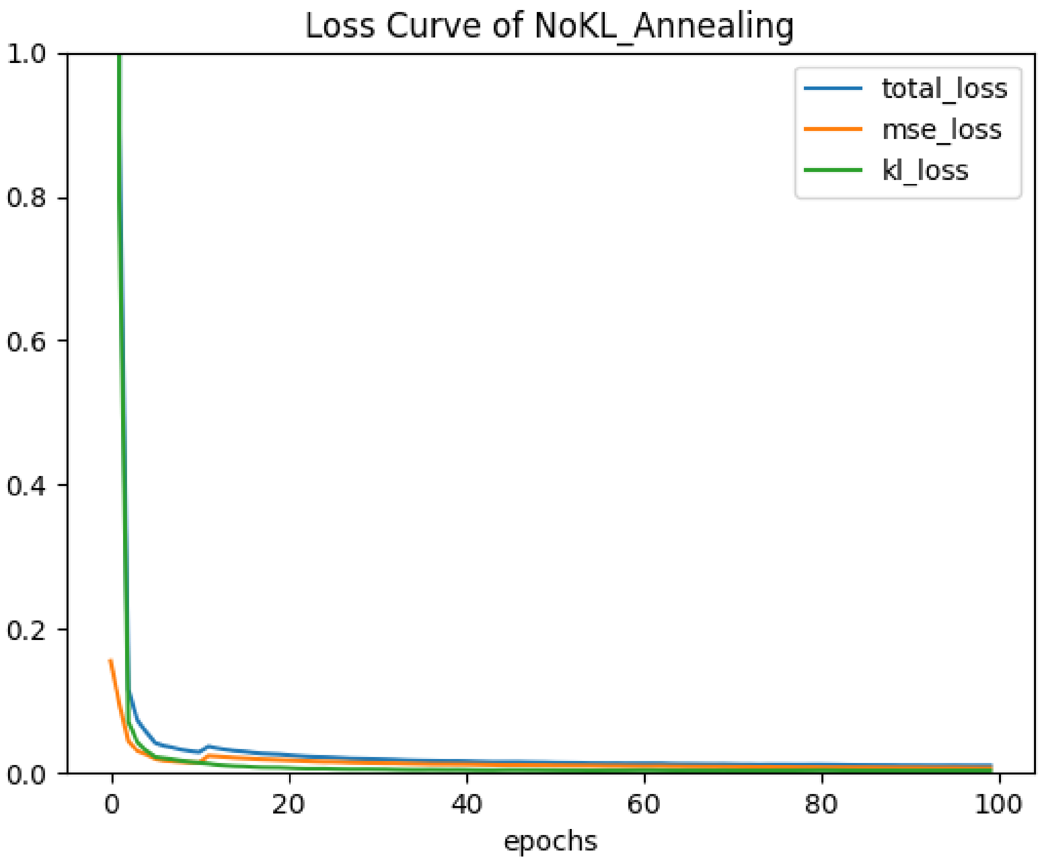
#

Alt text

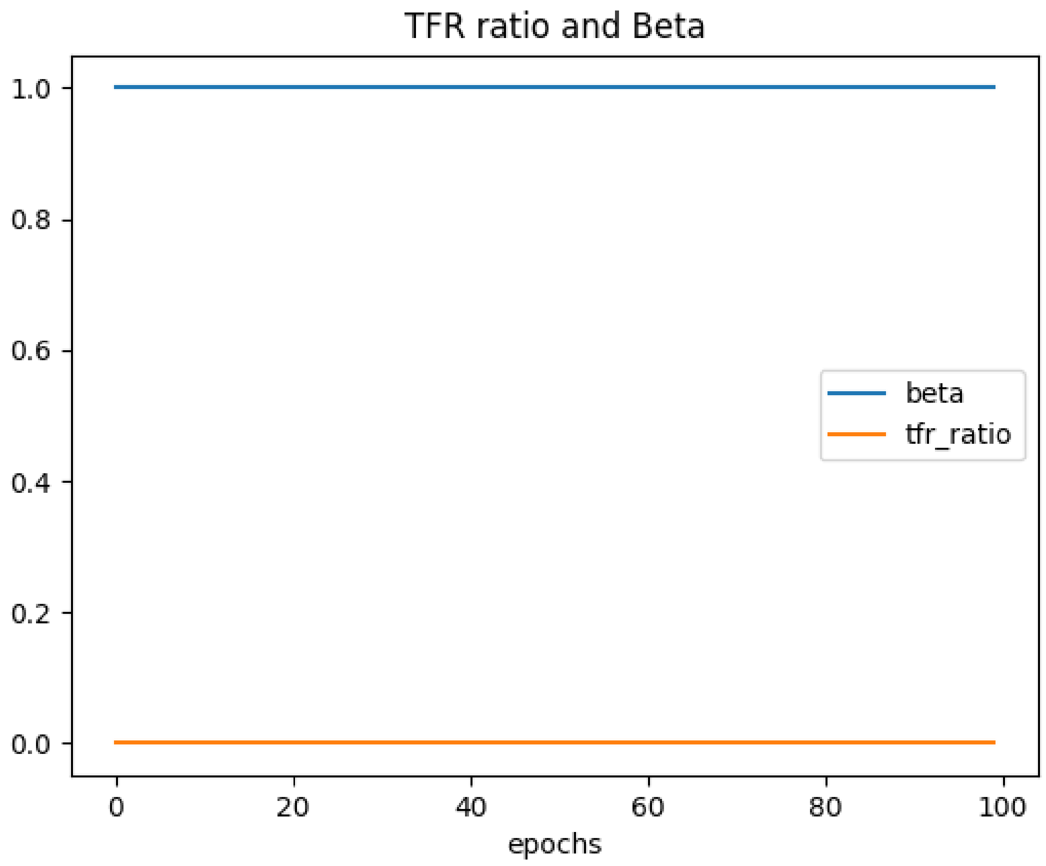
Highest PSNR

training

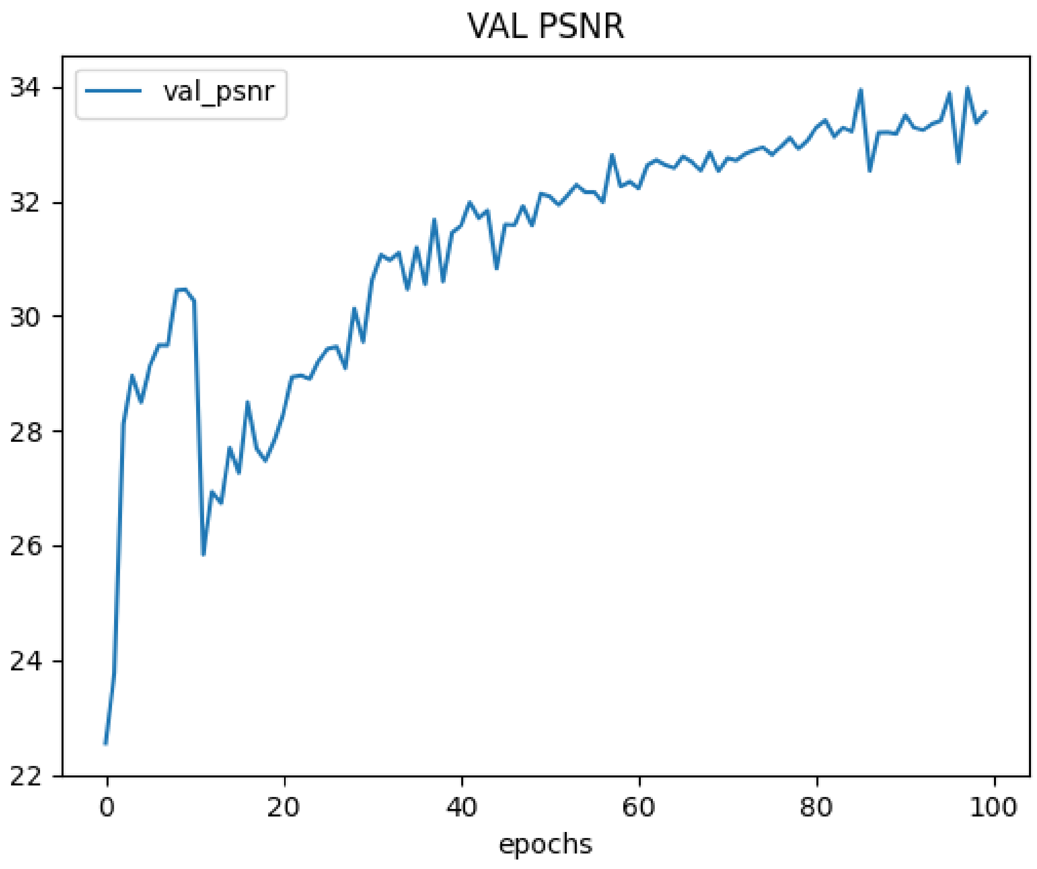
```
python Trainer.py --DR LAB4_Dataset/ --save_root
checkpoints/max_psnr/ --num_epoch 100 --fast_train
--fast_train_epoch 9 --kl_anneal_type NoKL_Annealing --tfr 0
```


Loss Curve of NoKL_Annealing
total_loss
mse_loss
kl_loss
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
epochs

Alt text


TFR ratio and Beta
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
epochs
beta
tfr_ratio

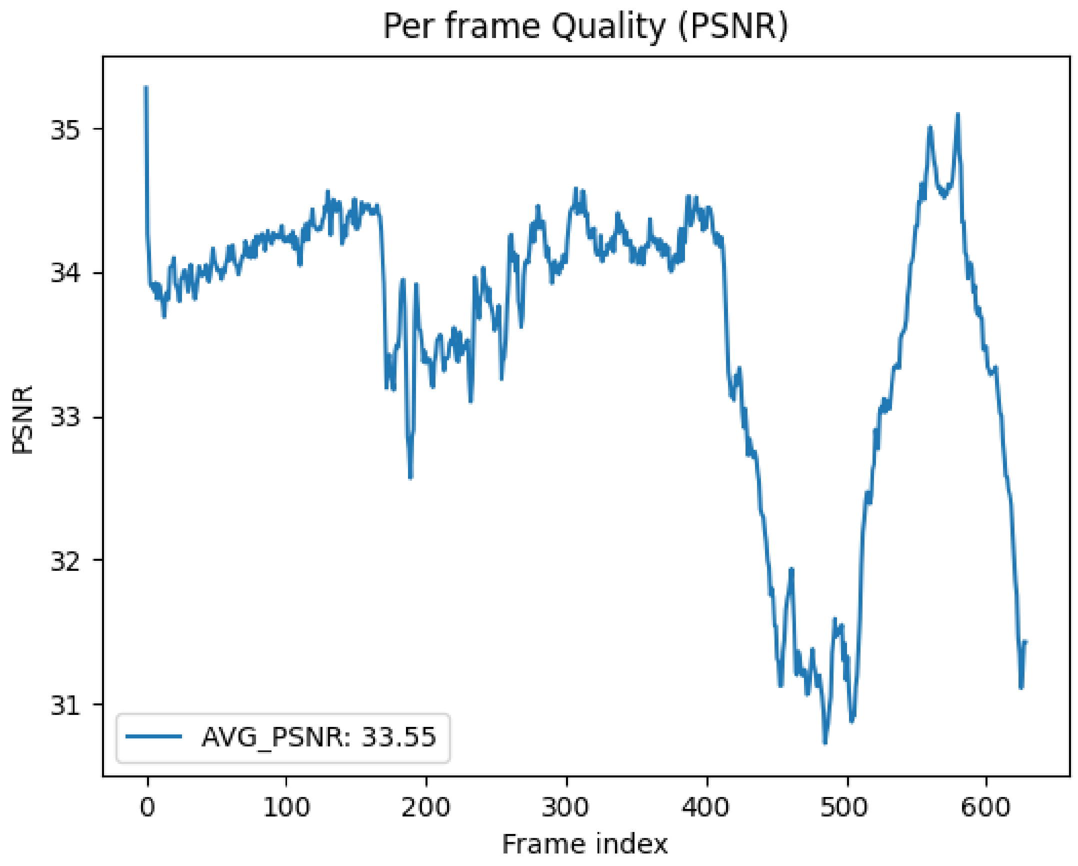
Alt text



Alt text

validation

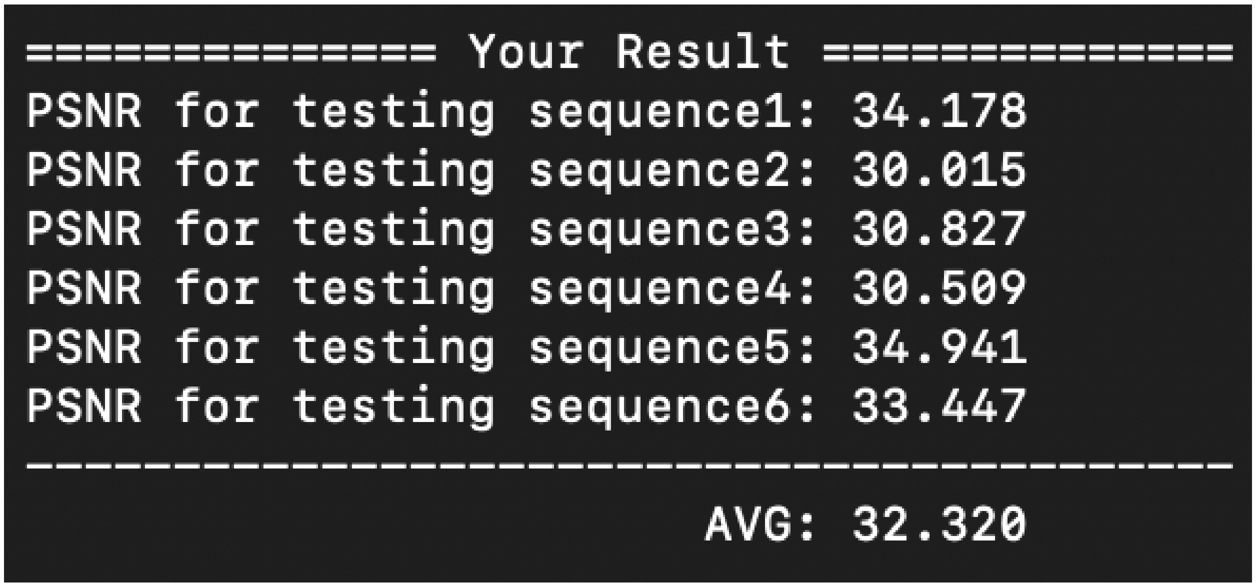
```
python Trainer.py --DR LAB4_Dataset/ --save_root
checkpoints/max_psnr/ --test --ckpt
checkpoints/max_psnr/epoch=99.ckpt
```



Alt text

testing

```
python Tester.py --DR LAB4_Dataset/ --save_root outputs/max_psnr/
--test --ckpt checkpoints/max_psnr/epoch=99.ckpt
```



```
============== Your Result ===============
PSNR for testing sequence1: 34.178
PSNR for testing sequence2: 30.015
PSNR for testing sequence3: 30.827
PSNR for testing sequence4: 30.509
PSNR for testing sequence5: 34.941
PSNR for testing sequence6: 33.447
-----------------------------------------
                      AVG: 32.320
```

Alt text

Reference

https://www.youtube.com/watch?v=9zKuYvjFFS8&t=6s&ab_channel=ArxivInsights

https://medium.com/mlearning-ai/a-must-have-training-trick-for-vae-variational-autoencoder-d28ff53b0023

https://www.youtube.com/watch?v=8zomhgKrsmQ&ab_channel=Hung-yiLee