Deep Learning Lab4 **Student ID: 312554014 Name: 羅翊軒**

1. Introduction

The aim of this assignment is to explore the techniques and methodologies involved in training a VAE to predict future video frames, leveraging the contextual information provided by previous frames and associated labels. In the following sections of the report is about the implementation details of the VAE-based conditional video prediction model, such as the training protocol, reparameterization trick for efficient gradient computation, teacher forcing strategy for stable learning, and the choice of the KL annealing ratio to strike a balance between exploration and exploitation during training. Moreover, analyzing the impact of these methods through evaluations of different experiments, comparing different settings and strategies.

Environment:

The environment and Python packages of this lab are as follows.

Processor	Intel® Core™ i5-13400F / GeForce RTX 3060 Ti
Python version	3.9.17
Python packages	Pandas, Numpy, Pytorch, Matplotlib, imageio

2. Implementation Details

A. Training protocol

```python
def training_one_step(self, img, label, adapt_TeacherForcing):
    self.optim.zero_grad()

    reconstructed_frames, mu, logvar = self.forward(img, label, adapt_TeacherForcing)

    rec_loss = self.mse_criterion(reconstructed_frames, img)
    kld_loss = kl_criterion(mu, logvar, img.size(0))

    loss = rec_loss + kld_loss * self.kl_annealing.get_beta()

    loss.backward()
    self.optimizer_step()

    return loss
```

In `training_one_step` method, it takes a batch size of image as input, and the shape of parameters img and label is (batch size, sequence length, channels, height, width). In this function, it will first generate the reconstructed image

by `forward` method. Then, it will calculate the loss and update the model's parameters based on the reconstruction loss and the KL divergence loss. Besides, this function supports the use of teacher forcing, which means the previous ground truth frame is used as input for the next prediction or not. The implementation of generating reconstructed image is shown below.

```
def forward(self, img, label, adapt_TeacherForcing):

    # Initialize the initial frame for prediction
    reconstructed_frames = [img[:, 0]]

    for i in range(1, img.size(1)):
        if adapt_TeacherForcing:
            # Use the previous ground truth frame as input for the next prediction
            input_frame = img[:, i-1]
        else:
            # Use the previous reconstructed frame as input for the next prediction
            input_frame = reconstructed_frames[-1]

        input_frame_features = self.frame_transformation(input_frame)
        ground_truth_features = self.frame_transformation(img[:, i])
        label_features = self.label_transformation(label[:, i])

        z, mu, logvar = self.Gaussian_Predictor(ground_truth_features, label_features)
        reconstructed_frame = self.Generator(self.Decoder_Fusion(input_frame_features, label_features, z))
        reconstructed_frames.append(reconstructed_frame)

    reconstructed_frames = torch.stack(reconstructed_frames, dim=1)

    return reconstructed_frames, mu, logvar
```

This `forward` method performs the forward pass through the Variational Autoencoder model. It takes input video frames and label frames and generates reconstructed frames, along with mean (mu) and log variance (logvar) of the latent space. If the parameter `adapt_TeacherForcing` is True, model will take the previous ground truth image as the input to generate reconstructed image for the following 15 images; if the parameter `adapt_TeacherForcing` is False, then the model will take the previous reconstructed image as the input instead of the ground truth image. With the teacher forcing training technique, it can help improving the stability and convergence of the training process. The function will return reconstructed images and the mean and log variance generated by gaussian predictor.

B. Reparameterization tricks

```python
def reparameterize(self, mu, logvar):
    # todo
    std = torch.exp(logvar/2)
    eps = torch.randn_like(std)
    return mu + eps * std
```

Reparameterization trick is implemented within the `reparameterize` function. This function takes the mean (mu) and log variance (logvar) of a Gaussian distribution, which are outputted by the Gaussian Predictor in VAE model. It samples from the distribution by generating random noise (eps) from a standard Gaussian distribution and scales it by the standard deviation calculated from the log variance. Finally, the sampled latent variable is obtained by adding the scaled noise to the mean. This function enable the model to sample from a Gaussian distribution and supports gradient-based optimization and efficient training.

C. Teacher forcing strategy

```python
def teacher_forcing_ratio_update(self):
    if self.current_epoch >= self.tfr_sde:
        self.tfr = max(0.0, self.tfr - self.tfr_d_step)
```

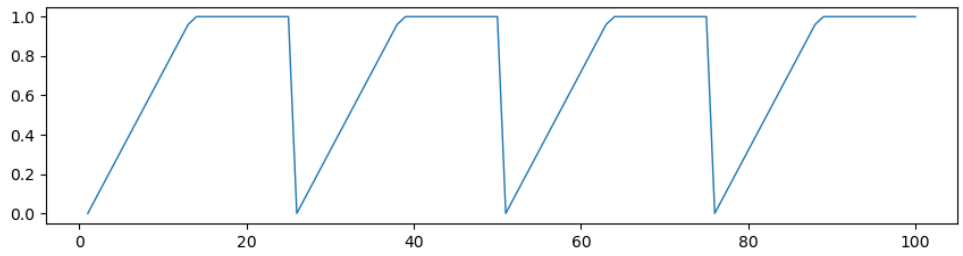
About teacher forcing strategy, there are three parameters can be adjusted, including teacher forcing ratio, epoch that teacher forcing ratio starts to decay, and decay step. For the parameter epoch that teacher forcing ratio starts to decay, it refers to the teacher ratio starting decay on which epoch intuitively, and about how much ratio to decay, it depends on the size of decay step. For the code that I implement, it will update the teacher forcing ratio only if the current epoch is greater than parameter epoch that teacher forcing ratio starts to decay, and the updated teacher forcing ratio will be teacher forcing ratio minus decay step.

D. KL annealing ratio

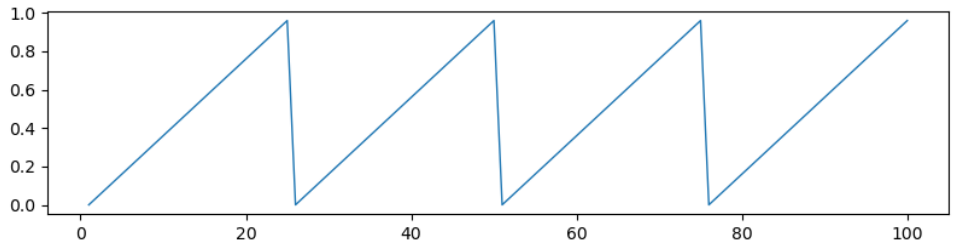
```python
class kl_annealing():
    def __init__(self, args, current_epoch=0):
        self.args = args
        self.beta = 1.0
        self.current_epoch = current_epoch
        self.kl_anneal_type = args.kl_anneal_type
        self.ratio = args.kl_anneal_ratio
        self.n_cycle = args.n_cycle

    def update(self):
        if self.kl_anneal_type == 'Cyclical':
            self.beta = self.frange_cycle_linear(self.args.num_epoch, start=0.0, stop=1.0, n_cycle=self.n_cycle, ratio=self.ratio)
        elif self.kl_anneal_type == 'Monotonic':
            self.beta = self.frange_cycle_linear(self.args.num_epoch, start=0.0, stop=1.0, n_cycle=1, ratio=0.25)
        elif self.kl_anneal_type == 'Without':
            pass
        else:
            raise ValueError(f"Unsupported kl_anneal_type: {self.kl_anneal_type}")
        self.current_epoch += 1

    def get_beta(self):
        return self.beta

    def frange_cycle_linear(self, n_iter, start=0.0, stop=1.0,  n_cycle=1, ratio=1):
        cycle_epoch_length = n_iter/n_cycle
        step = self.current_epoch % cycle_epoch_length
        start + step * (stop-start) / (cycle_epoch_length * ratio) if step < (cycle_epoch_length * ratio) else stop
```

For KL annealing beta, there are some parameters can be adjusted as well: KL annealing type, KL annealing ratio, and Kl annealing cycle. For KL annealing cycle, it controls how many times that the KL annealing beta should “iterate” during the training session. The meaning of “iterate” refers to that beta will start from zero and increases every epoch incrementally until reaching a specific value, and then remains constant for several epochs before starting to increase from zero again for the most cases. KL annealing ratio is the parameter which control the ratio between the situations of “increasing every epoch incrementally” and “remaining constant”. KL annealing has three methods. First one is “Cyclical”, where beta iterates multiple times. The second one is “Monotonic”, where beta only increases with the number of epochs and remains fixed after reaching a certain upper limit. The third one is “Without”, where beta remains unchanged and maintains a constant value, with a default of 1. There are some examples for different settings below:

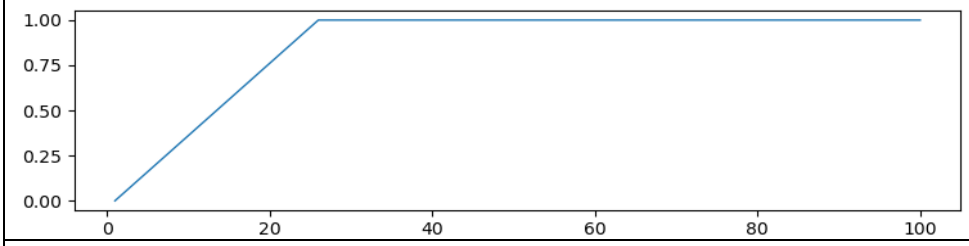
Cyclical: start=0.0, stop=1.0, cycle=4, ratio=0.5



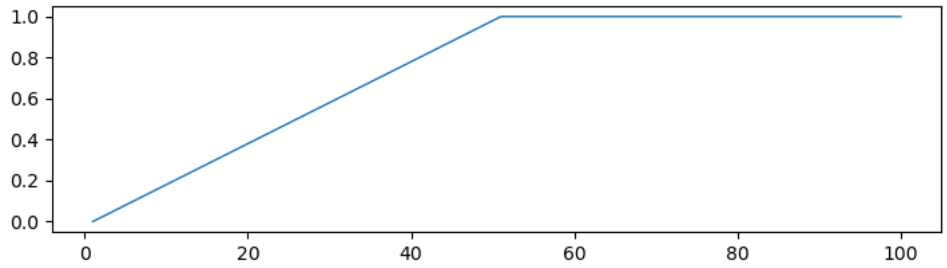
Cyclical: start=0.0, stop=1.0, cycle=4, ratio=1.0



Monotonic: start=0.0, stop=1.0, cycle=1, ratio=0.25



Monotonic: start=0.0, stop=1.0, cycle=1, ratio=0.5



3. Analysis & Discussion

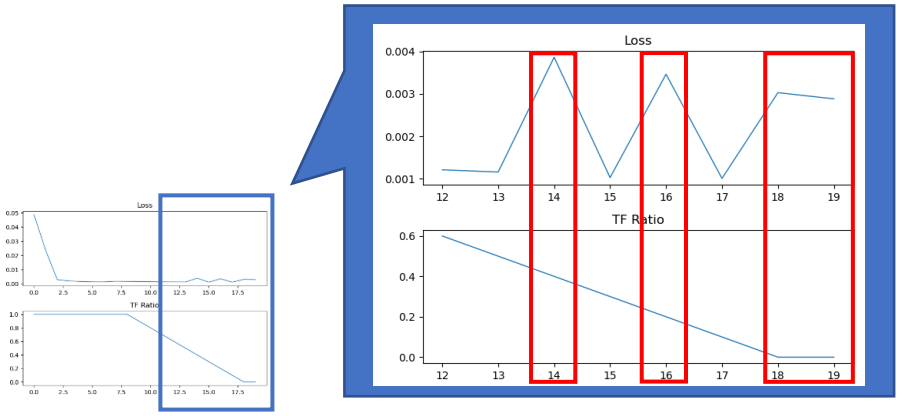
A. Teacher forcing ratio

◆ Analysis & compare with the loss curve

The table below is the table of teacher forcing ratio and teacher forcing method states of each epoch. Teacher forcing strategy turn off on 14th epoch, 16th epoch, and epochs after 18th. It is obvious that loss will

increase at the intersection of the epochs where teacher forcing strategy state from on to off.

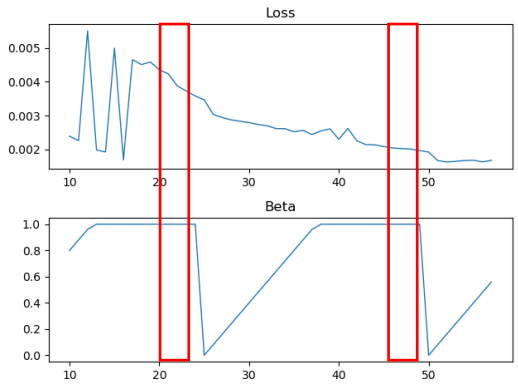
Epoch	**13**	**14**	**15**	**16**	**17**	**18**	**19**
Ratio	0.5	0.4	0.3	0.2	0.1	0	0
TF	ON	OFF	ON	OFF	ON	OFF	OFF



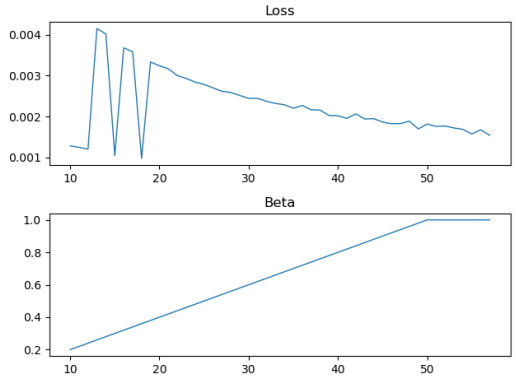
The figure above is the loss and teacher forcing ratio curve. It can see that loss turn high suddenly on 14th, 16th and 18th epoch. The sudden increase in loss may be attributed to the discontinuation of the Teacher Forcing mechanism. During model training step without teacher forcing mechanism, instead of using the ground truth images of the respective 15 frames to generate the reconstructed 15 frames, model will use the previous reconstructed frames as the input to generate the next-step reconstructed frame. As a result, the loss tends to increase.

B. Loss curve while training with different settings

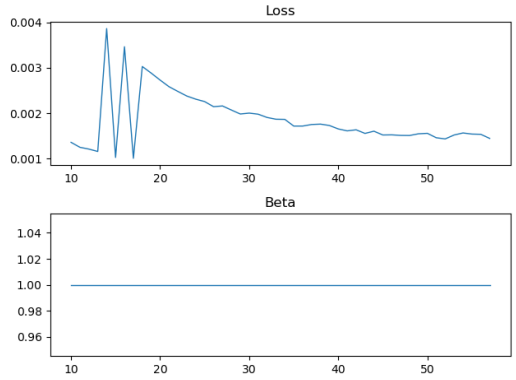
Cyclical: start=0.0, stop=1.0, cycle=4, ratio=0.5



Monotonic: start=0.0, stop=1.0, cycle=1, ratio=0.5

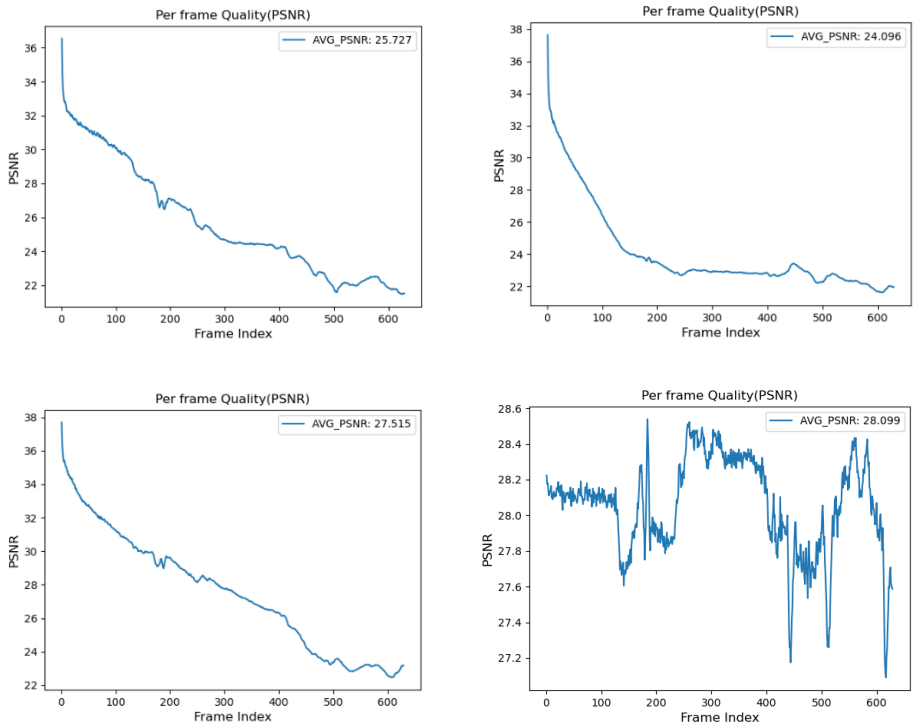


Without: kl_annealing_beta=1



From the “Cyclical” column in the table above, apart from the oscillations of loss curve before the first 20 epochs, it can be observed that when beta is starting a new cycle (the moment that beta from 1 to 0), there is a corresponding downward trend of the loss curve.

C. PSNR-per frame diagram in validation dataset



The figure above shows various experiments conducted with different settings. It is evident that in the majority of experiments, the PSNR values exhibit a rapid decline within the first 100 frames, with a decreasing trend as the distance from the first frame increases. This decline occurs because the error between the reconstructed images and the ground truth images tend to increase as moving towards the later frames.

In this Lab, the best result of my experiment is shown in the figure below:

D. Derivation of conditional VAE

$$\log P(x|c;\theta) = \log P(x,z|c;\theta) - \log P(z|x,c;\theta)$$

$$\int q(z|x,c)\log P(x,c)\,dz = \int q(z|x,c)\log\left(\frac{P(z,x|c)}{P(z|x,c)}\right)dz$$

$$= \int q(z|x,c)\log\left(\frac{P(z,x|c;\theta)}{q(z|x,c)}\,\frac{q(z|x,c)}{P(z|x,c;\theta)}\right)dz$$

$$= \underbrace{\int q(z|x,c)\log\left(\frac{P(z,x|c;\theta)}{q(z|x,c)}\right)dz}_{Lb(x,c,q,\theta)\cdots[1.1]} + \underbrace{\int q(z|x,c)\log\left(\frac{q(z|x,c)}{P(z|x,c;\theta)}\right)dz}_{KL(q(z|x,c)\,\|\,P(z|x,c,\theta))\cdots[1.2]}$$

∵ [1.2] is KL divergence term, and it's non-negative
$KL(q\|P)\geq 0$, it follows that $\log P(x|c;\theta) \geq Lb(x,c,q,\theta)$ with equality if and only if $q(z|c) = P(z|x,c;\theta)$
∴ [1.1] is a lower bound on $\log P(x|c;\theta)$

$$Lb(x,c,q,\theta) = \int q(z|x,c)\log\left(\frac{P(z|x,c;\theta)}{q(z|x,c)}\right)dz$$

$$= \int q(z|x,c)\log\left(\frac{P(x|z,c;\theta)P(z|c)}{q(z|x,c)}\right)dz$$

$$= \int q(z|x,c)\log\left(\frac{P(z|c)}{q(z|x,c)}\right)dz + \int q(z|x,c)\log P(x|z,c;\theta)\,dz$$

$$= \underbrace{-KL\left(q(z|x,c)\,\|\,P(z|c)\right) + \mathbb{E}_{q(z|x,c)}\log P(x|z,c;\theta)}_{\#}$$