Lab 4 Conditional VAE for Video Prediction

林穎志 411551007

1. Introduction

This is a conditional video prediction task using VAE-based model. We implemented part of the 2 papers[1,2]. We train and predict the video clips in GAN model structure. By taking a pose image generated by a pre-trained pose estimation network and a previous frame, the model can generate the next consecutive frame. The example can be seen as below. The figure shows that taking the setected pose that is generated by network as an imput to generate the next consecutive frame which is the source to the target result.

2. Implementation details

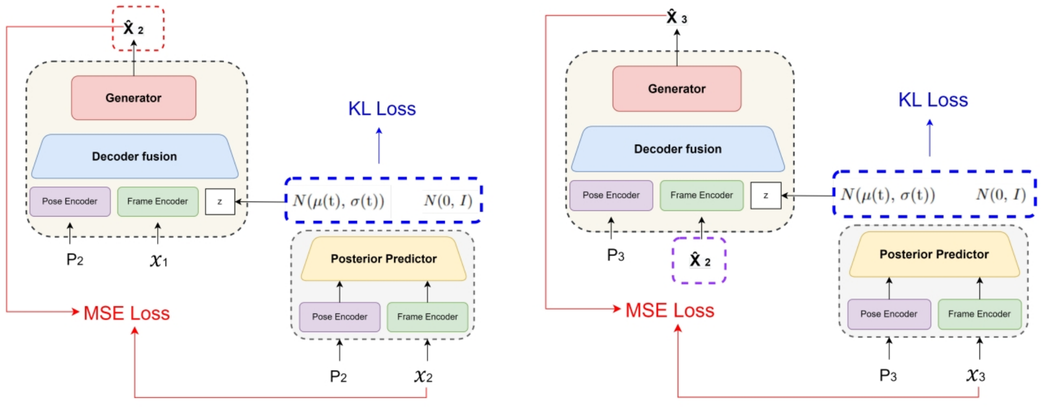
A. How do you write your training protocol

The figure below shows the main architecture of training, the model takes pose image, last predicted frame and sample Z from posterior predictor as input then generate the output by **Generator**. The **MSE loss** is calculated by the output and the label frame. Meanwhile, the label frame and the pose image are taken into **posterior predictor** as input to generate a distribution including latent variable(z), mean(μ), and standard error(σ). And the output of posterior predictor is used to calculate **KL divergence**. The total loss is defined as:

$$Loss = MSE + KL\ Divergence * \beta$$

- **KL annealing**
  - β is a coefficient to do KL annealing. We implement 3 kinds of KL annealing method and the result will be shown in the further section.
  - KL annealing is a technique used during the training of VAE. It malinly used to address the challenges posed by the Kullback-Leibler (KL) divergence term in the loss function. The KL divergence term encourages the learned latent space to match a predefined prior distribution, which typically is a Gaussian distribution.
  - KL annealing gradually increasing the weight or importance of the KL divergence term in the loss function during the early stages of training. This helps the model to initially focus on reconstructing the data accurately and then gradually shift its attention towards matching the desired prior distribution as training progresses.
- **Teacher forcing strategy**
  - Originally, the frame encoder takes the previous prediction as input to generate the next frame of prediction. In order to let the model learn more accurately or prevent the model input a bad prediction to generate a even more bad prediction of next frame, we implement teacher forcing strategy.
  - Once teacher forcing is performed, the frame encoder will take the previous label frame but not previous prediction as input. We assume this training strategy could train the model more stablely.
- **Reparameterization**
  - The reparameterization trick is used specifically to address the challenge of backpropagating gradients through the stochastic sampling process in VAEs. The core idea behind reparameterization is to separate the randomness introduced by sampling from the differentiable parts of the model, enabling gradients to flow through the model for learning.
- **Architecture and training**
  - This architecture is composited of 5 main models, we first initiate all these models and put all the corresponding input data into the models. Second, we calculate the loss and do backward propagation. Additionally, PSNR is calculated to monitor the performance.

PSNR	Optimizer	Learning rate	KL A type	KL A cycle	KL A ratio	TFR_step	Batc
24.72	AdamW	0.001	Cyclical	4	0.75	True	4



```
def training_one_step(self, img, label, adapt_TeacherForcing = True):
        # TODO
        # return loss

        self.frame_transformation.train().zero_grad()
        self.label_transformation.train().zero_grad()
        # Conduct Posterior prediction in Encoder
        self.Gaussian_Predictor.train().zero_grad()
        self.Decoder_Fusion.train().zero_grad()
        # Generative model
        self.Generator.train().zero_grad()

        img = img.permute(1, 0, 2, 3, 4) # change tensor into (seq, B, C, H, W)
        label = label.permute(1, 0, 2, 3, 4) # change tensor into (seq, B, C, H, W)


        train_mse = 0
        kld = 0
        PSNR_LIST = []
        for i in range(1, self.train_vi_len):

            x_t_1 = img[i-1]
            x_t = img[i]

            #pose encoder
            label_feat = self.label_transformation(label[i])
            #frame encoder
            if i == 1:
                human_feat_hat = self.frame_transformation(x_t_1)
            else:
                if adapt_TeacherForcing:
                    #put label into training
                    human_feat_hat = self.frame_transformation(x_t_1)
                else:
                    #put prediction into training
                    human_feat_hat = self.frame_transformation(self.pre_prediction)

            #put output
            #frame encoder
            frame_out = self.frame_transformation(x_t)
            z, mu_p, logvar_p = self.Gaussian_Predictor(frame_out, label_feat)

            #Put output of pose ecoder, frame encoder, z in decoder fusion
            parm = self.Decoder_Fusion(human_feat_hat, label_feat, z)
            #put output of decoder into generator
            x_hat_t = self.Generator(parm)
            self.pre_prediction = x_hat_t

            #loss
            #MSE
            mse_sub = self.mse_criterion(x_t, x_hat_t)
            train_mse += mse_sub

            #KL D
            kld += self.kl_criterion(mu_p, logvar_p, self.args.batch_size)

            #PSNR
            PSNR = self.Generate_PSNR(x_t, x_hat_t)
            PSNR_LIST.append(PSNR.item())


        self.beta = self.kl_annealing.get_beta()
        loss = train_mse + kld * self.beta
        avg_loss = loss / self.train_vi_len
        avg_PSNR = sum(PSNR_LIST)/(len(PSNR_LIST)-1)

        loss.backward()
        self.optim.step()

        return avg_loss, avg_PSNR
```

B. How do you implement reparameterization tricks

The steps are as following:

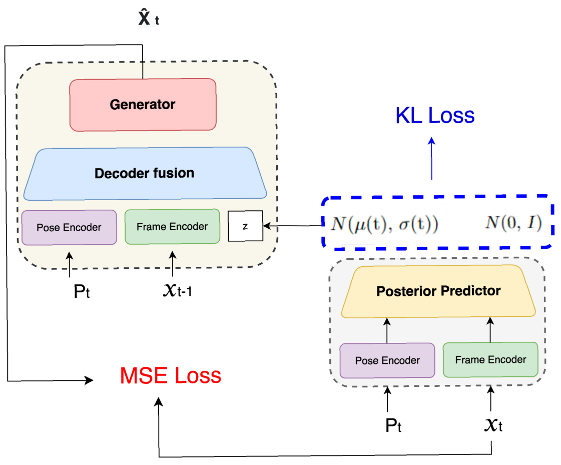
1. **Choose a Probabilistic Distribution**: In VAEs, the latent variables are assumed to follow a probabilistic distribution, often Gaussian (normal) distribution.
2. **Parameterize the Distribution**: The mean and standard deviation of the distribution are typically learned by the encoder network from the input data.
3. **Sample from a Noise Source**: During each training iteration, a noise sample ϵ is drawn from a simple noise distribution, usually a standard Gaussian distribution.
4. **Reparameterization Trick**: The latent variable z is computed by applying the reparameterization trick:

$$z = \mu + \sigma * \epsilon$$

```
def reparameterize(self, mu, logvar):
        # TODO
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return eps.mul(std).add_(mu)
```

C. How do you set your teacher forcing strategy

We set a Teacher Forcing Ratio(TFR) which set to be 1 initially and it will be descended as the current epoch grows. Every epoch it has a specific probability which is TFR to perform teacher forcing. That means it is more likely to do teacher focing at the begining of training, and it is less and less likely to do teacher forcing along with the epoch grows . We implement the different teacher forcing ratio step, the result will be shown in section 3.

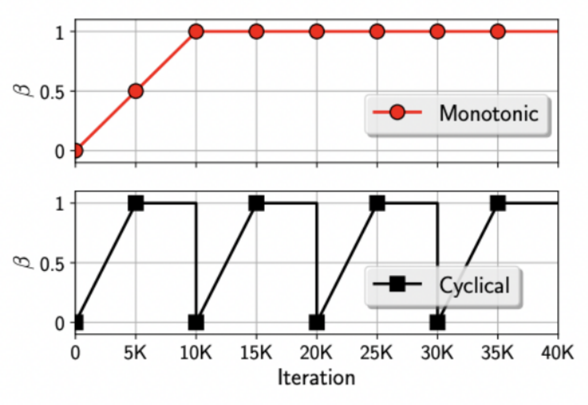


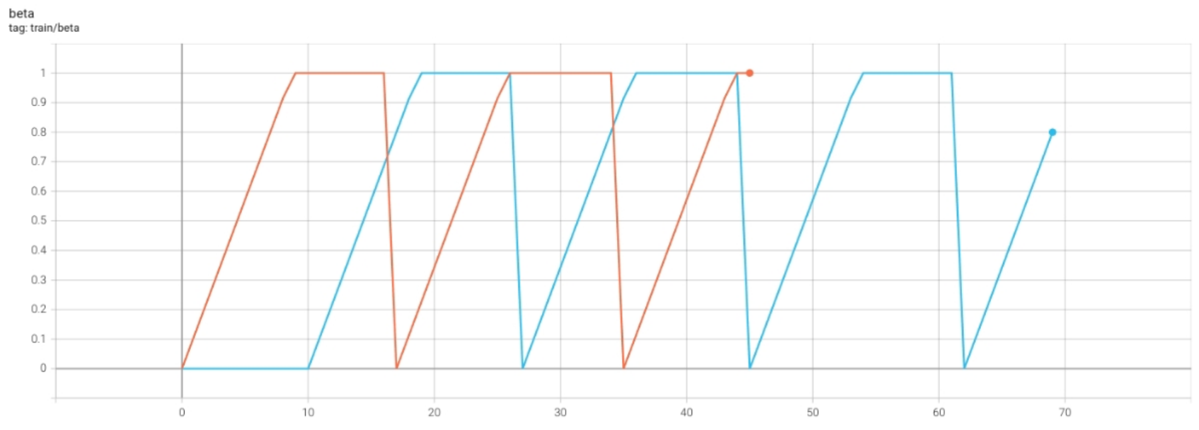
```
def teacher_forcing_ratio_update(self):

  if self.current_epoch >= args.tfr_sde:

    tfr = 1.0 - (self.current_epoch / args.num_epoch) * args.tfr_d_step
    self.tfr = tfr
    print(f'Teacher ratio: {self.tfr}')
```

D. How do you set your kl annealing ratio

Enforcing the KL divergence too early or the KL is too large in training can lead to issues such as posterior collapse, where the encoder ignores the data and only focuses on matching the prior.The KL annealing ratio β controls the rate at which the KL divergence term's weight is increased during training. We implement 3 kinds of annealing strategies including **Cyclical, Monotonic and Without KL annealing strategy**, the figure below shows how the β works along with the iteration.

We further do some experiments to let the model focusing on training the relationship between the previous frame and the later frame. We implement the KL annealing by **giving a late beta**, which means the beta in set to be 0 in the begining of the training, the beta comparison figure is shown as below. The blue line is how we implement this stategy.





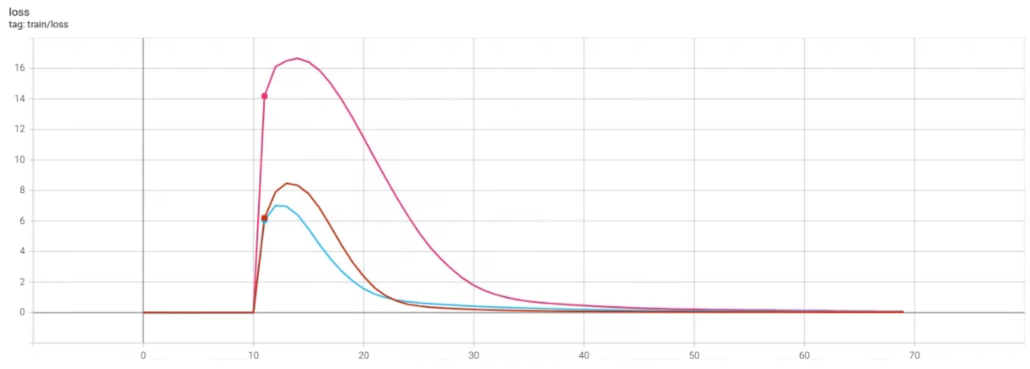
```
class kl_annealing():
    def __init__(self, args, current_epoch=0):
        # TODO
        self.current_epoch = current_epoch
        self.type = args.kl_anneal_type

        if self.type != 'None':
            self.t_beta = self.frange_cycle_linear(n_epoch=args.num_epoch, start_epoch=args.kl_anneal_start_epoch,
                                                   n_cycle=args.kl_anneal_cycle, ratio = args.kl_anneal_ratio)
        else:
            self.t_beta = np.ones(args.num_epoch)

        # raise NotImplementedError

    def update(self):
        # TODO
        self.current_epoch += 1


    def get_beta(self):
        # TODO
        beta = self.t_beta[self.current_epoch]
        return beta

    def frange_cycle_linear(self, n_epoch = 100, start_epoch = 10, start = 0, stop = 1,  n_cycle=4, ratio=0.5):
        # TODO
        if self.type == 'Cyclical':
            n_cycle = n_cycle
        elif self.type == 'Monotonic':
            n_cycle = 1

        L = np.ones(n_epoch)
        period = n_epoch/n_cycle
        step = (stop-start)/(period*ratio) # linear schedule

        for c in range(n_cycle):
            v , i = start , 0
            while v <= stop and (int(i+c*period) < n_epoch):
                if i < start_epoch:
                    L[i] = start
                else:
                    L[int(i+c*period)] = v
                    v += step
            i += 1
        return L
```

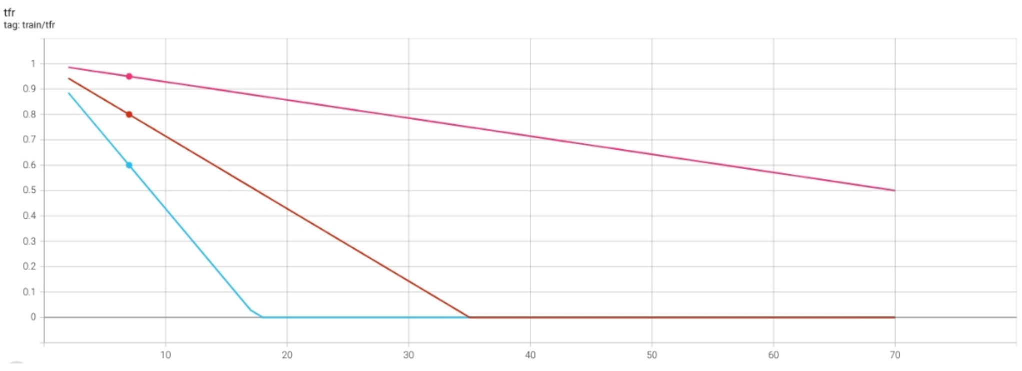
3. Analysis and Discussion

A. Plot Teacher forcing ratio

a. Analysis & compare with the loss curve

We implement different teacher forcing ratios to see how it influences the loss and PSNR. According to the comparison figures, the pink line, red line, blue line inidicate TFR_step = 1, 2, 4 respectively. The larger the TFR_step is, the teacher forcing ratio(TFR) will become smaller along the epoch grows. We can observe that the loss is larger when the ratio is larger, which means it could be more likely to do teacher forcing during training. We infer that it might be why the loss is larger when we have larger TFR , the performance is better.





	PSNR
TFR_step = 1	20.920
TFR_step = 2	19.944
TFR_step = 4	17.847

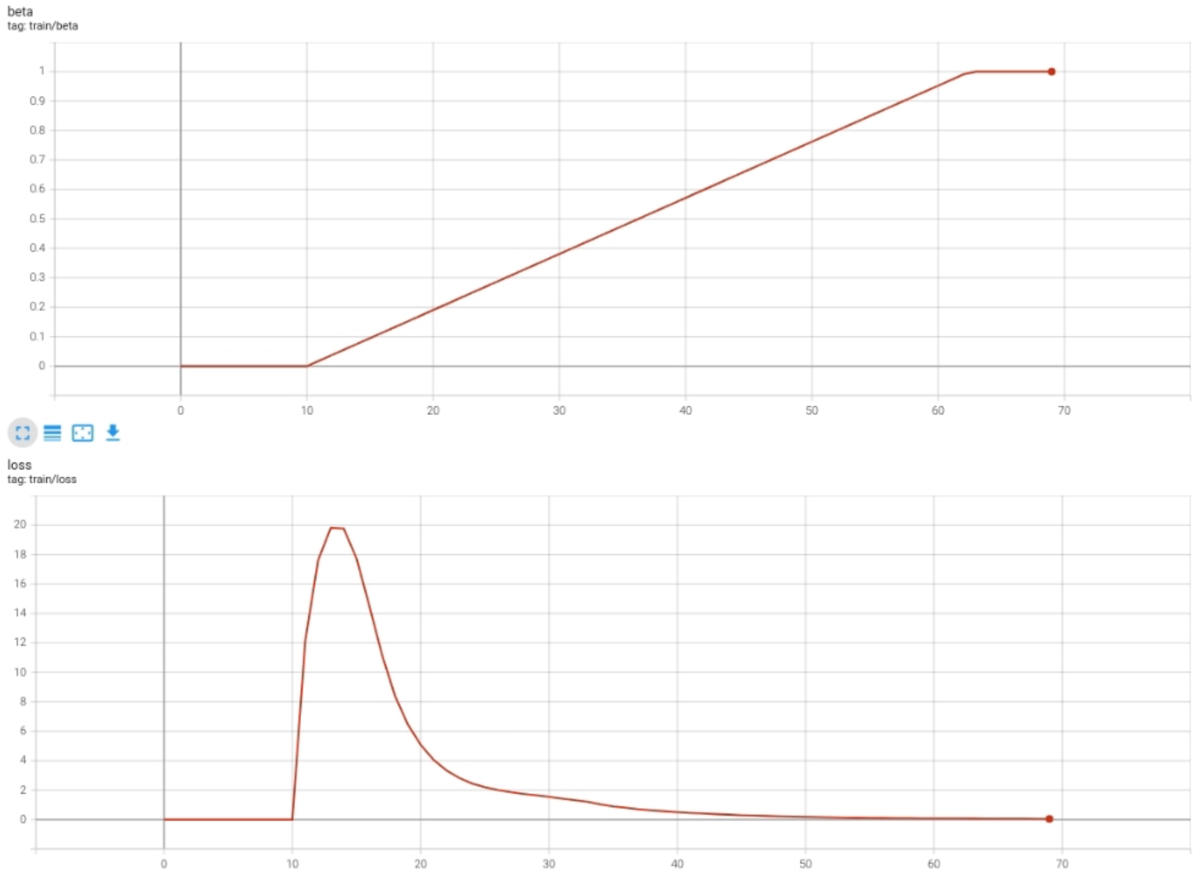
B. Plot the loss curve while training with different settings (Analyze)

In our experiment, we found that giving a late beta performed better while doing KL annealing. So we set the initial β as 0 and it starts to change at epoch 10. Further result will be shown at section d.

$$Loss = MSE + KL\ Divergence * \beta$$

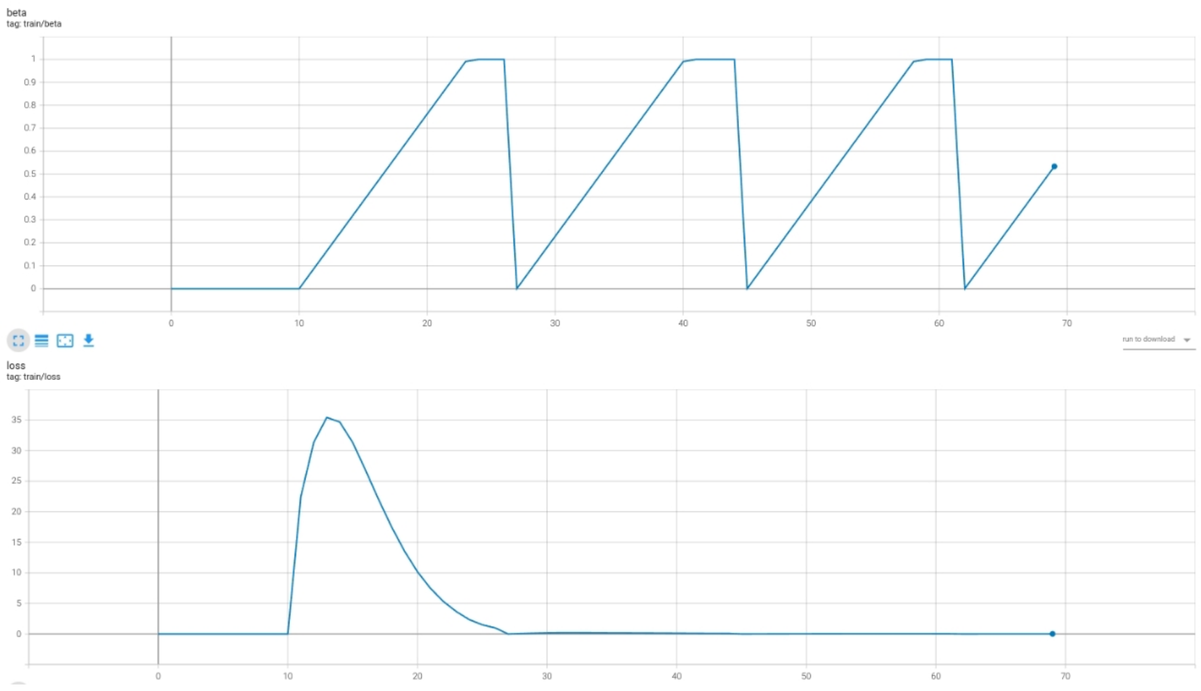
a. With KL annealing (Monotonic)

We can see obviously when the beta starts to grow, the loss becomes to be larger and then converges as the epoch grows. It's not hard to imagine how it happens. Due to the KL divergence starts to be contained in the loss calculation, the model must start to learn how to map the two distributions.



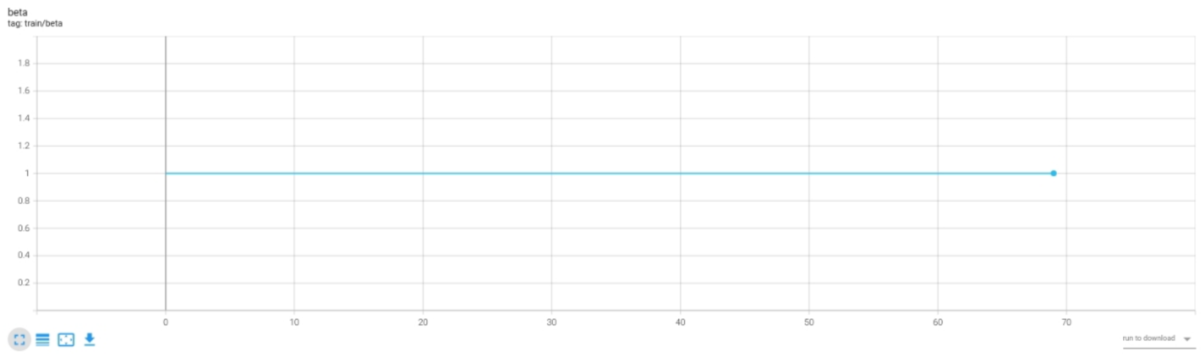
b. With KL annealing (Cyclical)

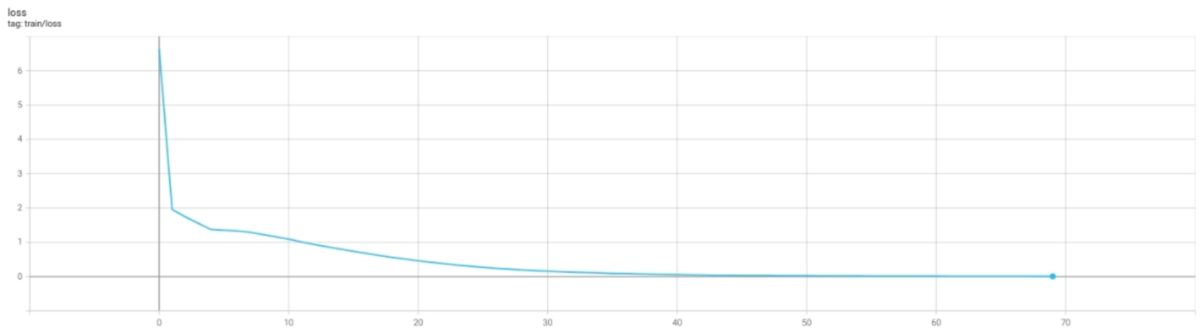
Same as the monotonic KL annealing strategy, we also can see obviously the beta is first contained in the loss term. When the model is trained to map the 2 distributions, the loss will not rise as the beta becomes larger during cycling.



c. Without KL annealing

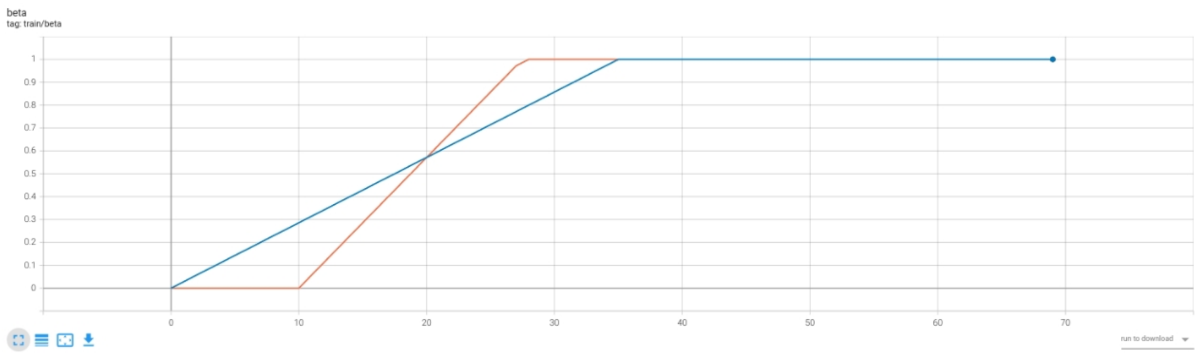
If we do not do KL annealing, the β is set to be 1 at whole epochs while training. The model is stricted to learn to map the 2 distributions at the begining. One can see that the loss is larger whe n the training starts, and then converges.

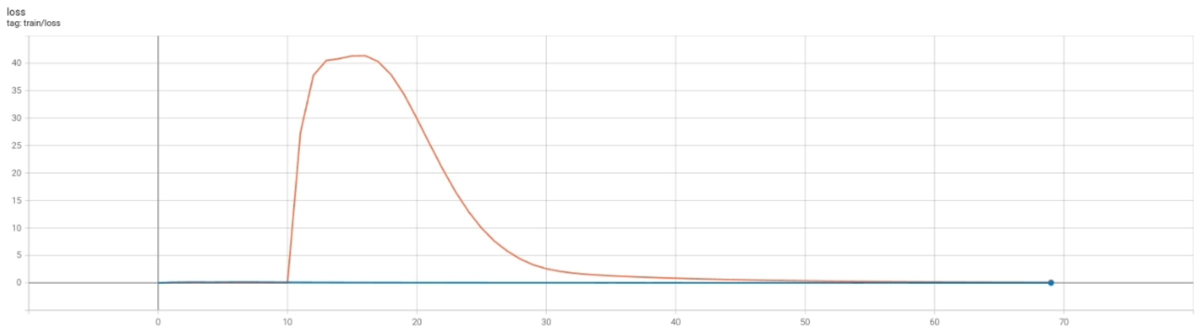




d. Extra experiment

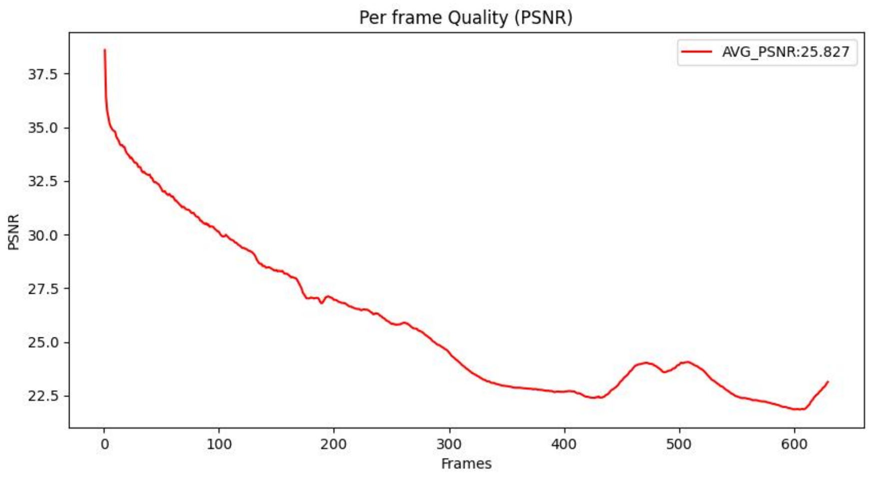
We implement the KL annealing by **giving a late beta**, which means the beta in set to be 0 in the begining of the training, and then grows as the setting before. We wish the model to focus on reconstructing the data at the begining and assume this strategy will have a better performance. The result shows that giving a late beta can improve PSNR.



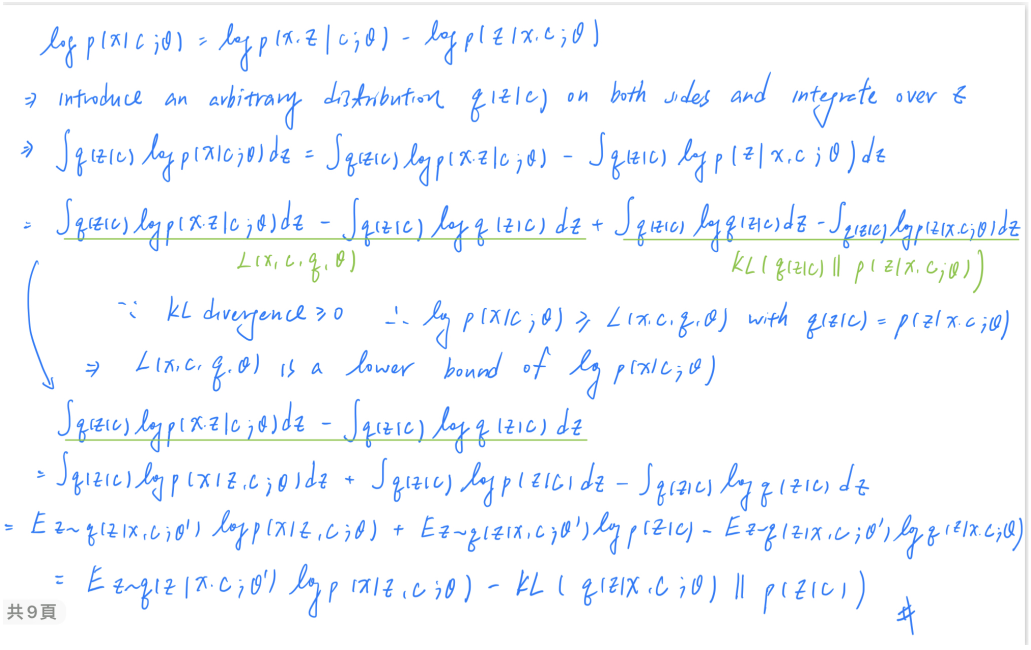


	PSNR
w\ Late Beta	21.4
w\o Late Beta	20.9

C. Plot the PSNR-per frame diagram in validation dataset



4. Extra - Derivate conditional VAE formula



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

$\Rightarrow$ introduce an arbitrary distribution $q(z|c)$ on both sides and integrate over z

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

$$= \underbrace{\int q(z|c)\log p(x,z|c;\theta)\,dz - \int q(z|c)\log q(z|c)\,dz}_{L(x,c,q,\theta)} + \underbrace{\int q(z|c)\log q(z|c)\,dz - \int q(z|c)\log p(z|x,c;\theta)\,dz}_{KL(q(z|c)\,\|\,p(z|x,c;\theta))}$$

$\because$ KL divergence ≥ 0 $\therefore \log p(x|c;\theta) \geq L(x,c,q,\theta)$ with $q(z|c) = p(z|x,c;\theta)$

$\Rightarrow L(x,c,q,\theta)$ is a lower bound of $\log p(x|c;\theta)$

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

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

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

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

#

共9頁

5. Reference

[1] C. Chan, et al., “Everybody Dance Now,” ICCV, 2019

[2] E. Denton, et al., “Stochastic Video Generation with a Learned Prior,” ICML, 2018