DL Lab4

i. Derivate conditional VAE formula

$$P(x|c;\theta) = \frac{P(x,z|c;\theta)}{P(z|x,c;\theta)}$$

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

introduce arbitrary distribution $q(z|x,c;\phi)$

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

($\log P(x|c;\theta)$ has no z)

$$\log P(x|c;\theta)\int q(z|x,c;\phi)\,dz = E_{z\sim q(z|x,c;\phi)}\log P(x,z|c;\theta) - E_{z\sim q(z|x,c;\phi)}\log q(z|x,c;\phi) + E_{z\sim q(z|x,c;\phi)}\log q(z|x,c;\phi) - E_{z\sim q(z|x,c;\phi)}\log P(z|x,c;\theta)$$

($\int q(z|x,c;\phi)\,dz = 1$; the last two terms form KL)

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

$KL \geq 0$

log likelihood (we want to maximize this) $\geq E_{z\sim q(z|x,c;\phi)}\log P(x,z|c;\theta) - E_{z\sim q(z|x,c;\phi)}\log q(z|x,c;\phi)$

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

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

#

ii. Introduction

In this lab, we are going to predict the video of the people given the image of the first frame and label on pose. We are going to leverage several training strategies like kl annealing and teachor forcing to help the process of training.

iii. Implementation details (25%)

1. How do you write your training protocol

   Given a batch, iterate from 0 to video length, compute the total loss in the batch and update the model.

```python
def training_one_step(self, img, label, adapt_TeacherForcing):
    # TODO
    loss = 0
    self.optim.zero_grad()
    last = img[:, 0]
    for i in range(1, self.train_vi_len):
        if adapt_TeacherForcing:
            out, mu, logvar = self.forward(last, img[:, i], label[:, i])
            last = img[:, i].detach()
        else:
            out, mu, logvar = self.forward(last, img[:, i], label[:, i])
            # temp = out.detach()
            # last = [self.histogram_specification(temp[i].unsqueeze(0), last[i].unsquee
            # last = torch.cat(last, axis=0)
            last = out.detach()
        mse = self.mse_criterion(out, img[:, i])
        kld = kl_criterion(mu, logvar, self.batch_size) * self.kl_annealing.get_beta()
        loss += mse + kld
    loss.backward()
    self.optimizer_step()
    return loss
```

2. How do you implement reparameterization tricks

```python
def reparameterize(self, mu, logvar):
    # TODO
    # do exponential to logvar
    var = torch.exp(logvar * 0.5)
    # create random number fron N(0,1)
    eps = torch.randn_like(var)
    # reparemeterize
    z = mu + eps * var
    return z
```

3. How do you set your teacher forcing strategy

   I tried several hyperparameters on tfr / tfr_step / tfr_d_step. At first, I thought having more teachor forcing value will guide the model towards the right direction and have a better result, but it turns out that tfr=0.8, tfr_sde=8, and tfr_d_step=0.1 will have good result. I have a small chance to do teacher forcing only in first 16 epoch, but the model performance is better than that of using higher teachor forcing epochs.

   I implemented teachor focing using the following code.

```python
if adapt_TeacherForcing:
    out, mu, logvar = self.forward(last, img[:, i], label[:, i])
    last = img[:, i].detach()
```

4. How do you set your kl annealing ratio

   I tried 2 kl annealing ratio, 1 and 2. Having a kl annealing ratio = 2 means that

the kl loss term will change fast in the first half of the cycle and be stable in the second half. This can help the take more considering more on kl loss, and the flat part in the second half might drive the model learn to minimize reconstruction loss. After some experiment, I set ratio to 1 and cycle 100, and it seems that having such a huge cycle did help stabilize the training, and the only disadvantage is that you have to train more epochs (I trained 2000 epoch on this set of hyper-parameters, and this is used as the final result in the kaggle submissions).

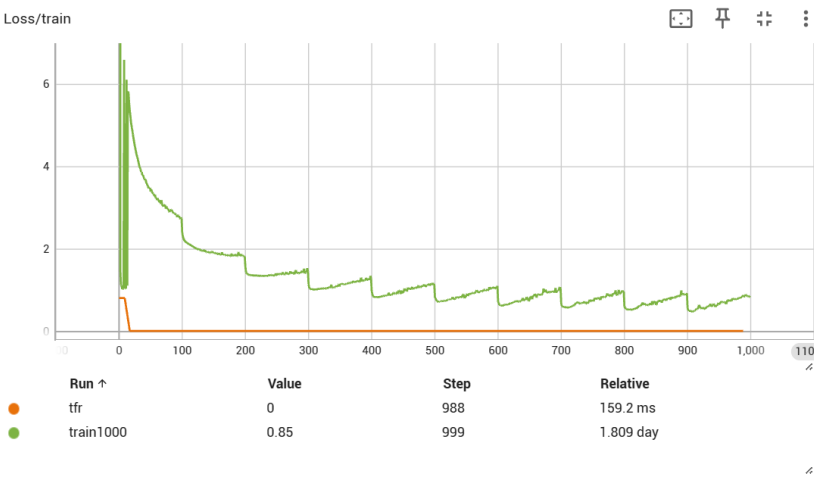
Implementation:

```python
def update(self):
    # TODO
    self.current_epoch += 1
    if self.kl_anneal_type == 'Cyclical':
        self.beta = self.frange_cycle_linear(start=0.0, stop=1.0)
    elif self.kl_anneal_type == 'Monotonic':
        self.beta = min(1.0, self.beta + self.kl_anneal_ratio * self.current_epoch / self.kl_anneal_cycle)
    else:
        raise NotImplementedError

def get_beta(self):
    # TODO
    return self.beta

def frange_cycle_linear(self, start=0.0, stop=1.0):
    # TODO
    return min(1.0, start + (stop - start) * (self.current_epoch % self.kl_anneal_cycle) / self.kl_anneal_cycle * self.kl_anneal_ratio)
```

iv. Analysis & Discussion

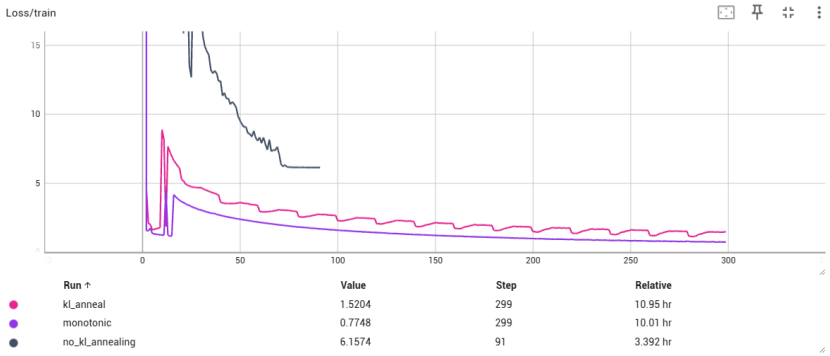
1. Plot Teacher forcing ratio

As described in iii. 3., I found that having so many epoch of teacher forcing is not a good thing. The cuve before tfr vibrates strongly because some of the epochs use teacher forcing and some do not. After the teacher forcing epochs, the curve go way smoother and drop gradually.

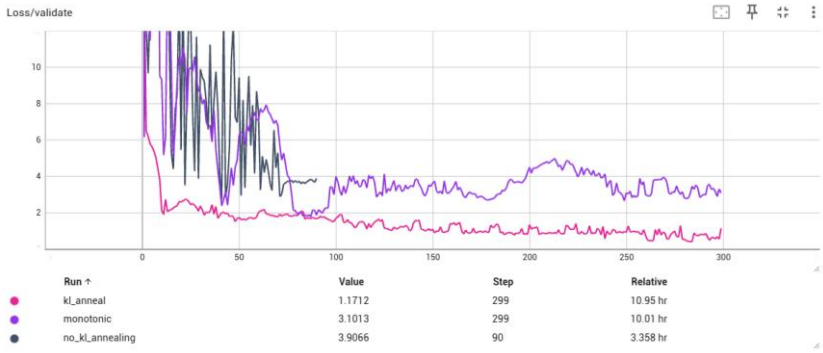


2. Plot the loss curve while training with different settings. Analyze the difference between them

Training loss:

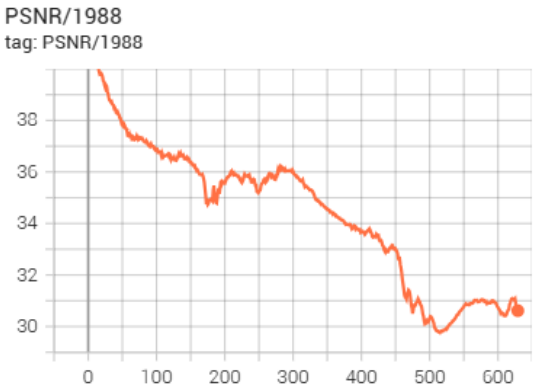


Validation loss

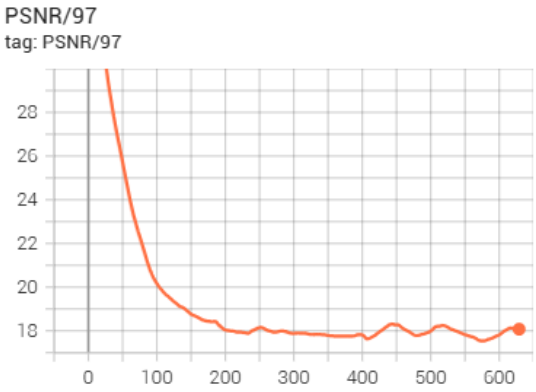


Unfortunately, I only trained the one without kl annealing for 100 epochs, but you can still tell the differences. The red curve is the one with kl anneal ratio = 2 & cycle = 20, and the purple one is monotonic. In the two graphs, you can tell that the training of kl anneal type monotonic is smooth and converges the fastest, but it's performance on validation set is worse than kl anneal type cycle (monotonic have twice the loss of cycle). Without kl annealling, the loss of kl term will always be one, making the model hard to reduce reconstruction loss at the very beginning. This is the reason why the model can not converge well in the beginning without kl annealling (as shown in graph).

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



The PSNR drops by each frame, and the PSNR sometimes recover after a drop, which shows that the model did learn how to recover from small pertubations. The model does not have the ability to recover the dropped value in the epoch 100.



4. Other training strategy analysis

- Histogram specification

  At first, I only trained 100 epoch on the task, and the result is not satisfying. The blackground looks grey. It is white in picture, showing that my model does not learn the lighting condition well, and it just kept getting darker and darker each frame. Hence, I do histogram specification on the next input of the dataset (preprocessing on next input) using the last input (I assume there won't be many changes in the lighting condition in the pixels). This approach is sort of like teacher forcing, but finetune the input image instead of using the correct image. This approach can have an average PSNR of 27 but can not deal with the frames that the people went outside the frame well (as shown

in the following image). This can help stabilize the training when teacher forcing ratio is 0 but must be disabled when the model learns well.

- Crop the image

  Observing that my model does not do when the people goes out of the frame, I tried to have the model learn this condition (there is a part in the testing set that the people walk towards you, and my model delete the man directly). After having a model learned something, I have the model trained on another set of videosmixed with 50% original data and 50% cropped image. The result shows that this approach does not help on the condition, but instead make the original result on valdition set worse. The training curve is as follows: