

DL_Lab4: C-VAE for Video Prediction 313553024 多工一蘇柏叡

1. Report

1.1 Derivate conditional VAE formula

- 前提介紹
 $p(x|c;\theta)$ 為一個 conditional probability distribution 而 C-VAE 的目標
即是「maximize the log-likelihood of $p(x|c;\theta)$ 」
- $\max_{\theta} \int \log p(x|c;\theta) dx \Rightarrow$ 即為原式
也就是想辦法把 $KL(q(z)||p(z|x))$ 變得愈小愈好

推導過程

假設有一個 randomly 的 probability $q(z|c)$ 用來近似真實的後驗分佈 $p(z|x, c; \theta)$ 且 $\log p(x|c;\theta)$ 和 z 無關
故可以將 $\log p(x|c;\theta)$ 視為常數 $\Rightarrow \log p(x|c;\theta) \int q(z|c;\phi) dz$
 \hookrightarrow give it as 1 \Rightarrow 即為 $\log p(x|c;\theta)$

我們可將 $\log p(x|c;\theta)$ 寫成

$$\begin{aligned}\log p(x|c;\theta) &= \int \log p(x|c;\theta) q(z|c) dz \\ &= \int q(z|c) \log \frac{p(x,z|c;\theta)}{p(z|x,c;\theta)} dz \\ &= \int q(z|c) \log \left(\frac{p(x,z|c;\theta)}{q(z|c)} \right) \left(\frac{q(z|c)}{p(z|x,c;\theta)} \right) dz \\ &\quad \text{提出} \quad \checkmark \text{拆項} \\ &= \int q(z|c) \log \frac{p(x,z|c;\theta)}{q(z|c)} dz + \int q(z|c) \log \frac{q(z|c)}{p(z|x,c;\theta)} dz \\ &\quad \text{最後可寫成} \quad \xrightarrow{\text{先前提到之}} \\ &= \int q(z|c) \log \frac{p(x,z|c;\theta)}{q(z|c)} dz + [KL(q(z|c)||p(z|x,c;\theta))] \Delta\end{aligned}$$

此外又：不論是 $p(x|c;\theta)$ 或是 $q(z|c)$ 皆是 $\geq 0 \Rightarrow \underline{KL(p||q) \geq 0}$

i. 同理 $\int q(x|c) \log \frac{q(z|c)}{p(z|x,c;\theta)} dz \geq 0$ 换言之，我們可以說

$$\log p(x|c;\theta) \geq \boxed{\int q(x|c) \log \frac{q(z|c)}{p(z|x,c;\theta)} dz} \Rightarrow \text{即 } \square \text{部分為 } \log p(x|c;\theta) \text{ 的 lower bound}$$

$$\text{而 Loss}(x, c, q, \theta) = \int q(z|x, c; \phi) \log \frac{p(x|z, c; \theta) p(c|x; \theta)}{q(z|x, c; \phi)} dz$$

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

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

$$= -KL(q(z|x, c; \phi) || p(z|c; \theta)) + E_z q(z|x, c; \phi) [\log p(x|z, c; \theta)]$$

1.2 Introduction

在本次Lab中，我們使用C-VAE對影片進行預測，我們使用23410frames進行訓練，使用630frames的驗證集進行驗證，並且在5個包含630frames的測試序列中進行預測。此外本次實驗中亦引入了Teacher Forcing ratio(含Decay)、KL Annealing等技術針對模型訓練進行優化並比較，並且是以PSNR做為評估指標來衡量影片輸出品質。

1.3 Implementation details

1.3.1 How do you write your training/testing protocol

a. training_stage:

- i. 將各訓練、驗證損失、tfr、kl各建一個空list用於存值，並用於繪圖。
- ii. 將adapt_tfr該行放於迴圈內，以增加隨機性，否則整個training過程在teacher forcing部分將會全有全無。
- iii. 在處理val_loss、psnr_list、avg_psnr與繪圖時，關閉train模式
- iv. 使用avg_psnr作為學習率是否調整的依據，此外由於val_loss、avg_psnr對於後續測試皆有參考依據，故存取checkpoint時，會考慮兩者數據(畢竟有時候Loss低但PSNR高，或是相反)。
- v. 為避免訓練中斷造成結果難以呈現，因此將每10個epoch進行繪圖並存取。

```
def training_stage(self):  
    min_loss = float('inf')  
    max_psnr = 0  
    loss_list, val_loss_list, tf_list, kl_list = [], [], [], []  
    #adapt_TeacherForcing = random.random() < self.tfr  
    #為了增加隨機性，所以不在這裡定義，否則會一開始判斷是否<self.tfr，後面就不會再變動了  
    #若放在迴圈內，每次都會重新定義，就會有機會變動  
    for epoch in range(self.args.num_epoch):  
        train_loader = self.train_dataloader()  
        loss_temp = []  
        adapt_TeacherForcing = random.random() < self.tfr  
        for (img, label) in (pbar := tqdm(train_loader, ncols=120)):  
            img = img.to(self.device)  
            label = label.to(self.device)  
            loss = self.training_one_step(img, label, adapt_TeacherForcing)  
            loss_temp.append(loss.detach().cpu().item())  
  
            beta = self.kl_annealing.get_beta()  
            TF_status = 'ON' if adapt_TeacherForcing else 'OFF'  
            pbar.set_description(f'Epoch {epoch+1}/{self.args.num_epoch} [TeacherForcing: {TF_status} {self.tfr:.1f}, beta: {beta:.1f}, lr: {self.last_lr:.1e}]')  
  
[TeacherForcing: {TF_status} {self.tfr:.1f}, beta: {beta:.1f}, lr: {self.last_lr:.1e}]')
```

```

    avg_loss = np.mean(loss_temp)
    loss_list.append(avg_loss)
    kl_list.append(beta)
    tf_list.append(self.tfr)

    self.train(False) # 設置為驗證模式
    val_loss, avg_psnr, psnr_list = self.eval()
    val_loss_list.append(val_loss)
    self.plot_psnr(psnr_list, avg_psnr, epoch)
    self.train(True) # 設置為訓練模式

    if val_loss < min_loss or avg_psnr > max_psnr:
        min_loss = min(val_loss, min_loss)
        max_psnr = max(avg_psnr, max_psnr)
        self.save_checkpoint(val_loss, avg_psnr, epoch)

    # 學習率調整和其他更新
    self.scheduler.step(avg_psnr)
    self.update_teacher_forcing()
    self.kl_annealing.update()

    if (epoch + 1) % 10 == 0: ##預防中斷，每10個epoch存圖
        self.save_plot_loss([loss_list, val_loss_list, tf_list, kl_list])
    self.current_epoch += 1
    self.save_plot_loss([loss_list, val_loss_list, tf_list, kl_list])

```

b. training_one_step:

- i. 將計算loss、處理單一frame生成預測圖片、傳入的mu、logvar拆分成兩個獨立的function。
- ii. frame_loss包含了預測和groundtruth使用MSE後的loss，再加上kl_loss * beta value
- iii. train_process_frame則是將當前frame轉換為特徵，並透過高斯預測器產生潛在變數、分布參數。並將這些feature與label融合並產生最終的預測圖片。
- iv. 在每次梯度計算之前，先將優化器中的梯度歸零，以避免之前的梯度累積影響當前步驟。
- v. 若是平均損失值非nan值，先對其做梯度裁剪(**max_norm設2**)，若是出現nan則停止update

```

def calculate_frame_loss(self, pred_img, target_img, mu, logvar, beta):
    """Calculate the reconstruction and KL loss for a single frame."""
    frame_loss = self.mse_criterion(pred_img, target_img)
    kl_loss = kl_criterion(mu, logvar, self.batch_size)
    return frame_loss + beta * kl_loss

def train_process_frame(self, current_img, label_features):
    """Process a single frame to generate latent variables and the predicted image."""
    img_features = self.frame_transformation(current_img)
    z, mu, logvar = self.Gaussian_Predictor(img_features, label_features)
    fused_features = self.Decoder_Fusion(img_features, label_features, z)
    pred_img = self.Generator(fused_features)
    return pred_img, mu, logvar

```

```

def training_one_step(self, img: Tensor, Label: Tensor, adapt_TeacherForcing: bool):
    """Perform a single step of training."""
    pred_img = img[:, 0] # Start prediction from the first frame
    total_loss = torch.zeros(1, device=self.device)
    beta = self.kl_annealing.get_beta()
    self.optim.zero_grad()

    for i in range(1, self.train_vi_len):
        # Choose the correct previous frame based on whether teacher forcing is applied
        if adapt_TeacherForcing:
            current_img = img[:, i - 1]
        else:
            current_img = pred_img
        label_features = self.label_transformation(Label[:, i])
        # Process the frame and compute loss
        pred_img, mu, logvar = self.train_process_frame(current_img, label_features)
        loss = self.calculate_frame_loss(pred_img, img[:, i], mu, logvar, beta)
        total_loss += loss

```

```

# Average the loss over the length of the training sequence
average_loss = total_loss / (self.train_vi_len - 1)

if not torch.isnan(average_loss):
    average_loss.backward()
    nn.utils.clip_grad_norm_(self.parameters(), max_norm=2.0)
    self.optimizer_step()
    return average_loss
else:
    # Handle NaN loss
    print("Encountered NaN loss, skipping update.")
    return torch.tensor(float('inf'), device=self.device)

```

c. testing protocol (Demo時被問倒，故進行revise！)

- i. CVAE在訓練以及測試上有截然不同的z生成，因此在測試時的 z 並非是經由Gaussian_Predictor透過frame、label的feature計算而得，因為在training時CVAE通常使用條件c來學習生成的分佈。隱變量z是從條件潛在分佈 $P(z|c)$ 中抽樣得到的。因此，在訓練上z的抽樣是條件依賴的，且是學習得到的。然而在測試時，通常因為條件不可用，如僅有照片並無標註之label，所以通常在測試時z的生成會使用非條件分布，如直接使用常態分布去抽樣z。
- ii. decode_feature即是將z、frame、label的feature進行融合並傳入Generator生成下一幀，並且將新生成的幀添加到預測列表中，以便於後續的處理和評估。

```
# TODO
pred_img = [img[0].detach()]
for i in range(1, self.val_vi_len):#從1開始，因為第一張做為input
    current_img = pred_img[-1].detach()# 防止梯度回流

    # extract features and predict the next frame
    frame_features = self.frame_transformation(current_img)
    label_features = self.label_transformation(label[i])

    # z, mu, logvar = self.Gaussian_Predictor(frame_features, label_features)
    # Sampling z from a standard normal distribution

    z = torch.randn((1, self.args.N_dim, self.args.frame_H, self.args.frame_W), device=self.args.device)

    decoded_features = self.Decoder_Fusion(frame_features, label_features, z)
    new_pred_img = self.Generator(decoded_features)

    # Add the newly generated frame to the prediction list for further processing
    pred_img.append(new_pred_img)
    decoded_frame_list.append(new_pred_img.cpu())
    label_list.append(label[i].cpu())
```

1.3.2 How do you implement reparameterization tricks

log-variance—也就是 σ^2 再取log，而標準差即是先乘上0.5再對其取指數，而再隨機生成一個為常態分佈的random noise稱之為 ϵ (epsilon)，使得最後生成的z保有隨機性，且又能在training過程中進行梯度更新(換言之，就是要確保sample過程中，隨機性是可以和model分離的)。公式: $z = \mu + \sigma \cdot \epsilon$

```
def reparameterize(self, mu, Logvar):
    # TODO
    ...
    根據forward提示，model output為mu、logvar，透過reparameterize取得z
    重參數技巧主要是為了讓模型在訓練時能夠反向傳播
    eps會生成一個標準常態分佈的tensor，並且與std相乘，再加上mu
    ...
    std = torch.exp(0.5*Logvar)
    eps = torch.randn_like(std)
    z = mu + eps * std
    return z
```

1.3.3 How do you set your teacher forcing strategy

tfr_sde為第n個epoch起開始下降teacher forcing ratio，假設是第5個epoch開始下降tfr，那麼當current_epoch = 4時會觸發該條件，並且用當前tfr去減掉tfr_d_step，又因為teacher forcing ratio為[0,1]區間之實數，故觸發該條件式之後需要處理負數問題，那這邊拿[0, tfr]去取max則可以避免掉負數問題。

```
def update_teacher_forcing(self):
    if self.current_epoch+1 >= self.tfr_sde: #因為current_epoch是從0開始，所以要+1
        self.tfr -= self.tfr_d_step
        self.tfr = max(0.0, self.tfr)
```

1.3.4 How do you set your kl annealing ratio

分成Monotonic、Cycle、None(beta直接設為1)三種，在判斷type時，若非None則初始為0.0並傳入stop、ratio進行計算。而在frange_cycle_linear中，我們會計算cycle中結束的epoch以及beta要上升的斜率，接下來就是根據Monotonic、Cycle定義進行撰寫條件式，首先，Monotonic的定義是當index超過停止上升beta的epoch值時，beta會等於傳入的stop value，反之則是用斜率乘上當前index再加上初始值。接著，Cycle的定義是在抵達stopped_epoch前，就是斜率乘上當前index再加上初始值，反之，beta會等於傳入的stop value。

```

class kl_annealing():
    def __init__(self, args, current_epoch=0):
        self.args = args
        self.type = args.kl_anneal_type
        self.ratio = args.kl_anneal_ratio
        self.stop = args.kl_anneal_stop
        self.anneal_cycle = args.kl_anneal_cycle
        self.epochs = current_epoch
        self.beta = 0.0
        if self.type == 'None':
            self.beta = 1.0
    def update(self):
        # TODO
        self.epochs += 1
        if self.type != 'None':
            self.frange_cycle_linear(0.0, self.stop, self.ratio)
        else:
            self.beta = 1.0 #如果不需要annealing，則beta設為1.0

    def get_beta(self):
        # TODO
        return self.beta
    def frange_cycle_linear(self, start, stop, ratio):
        stopped_epoch = np.ceil(self.anneal_cycle * ratio)
        slope = (stop - start) / stopped_epoch

        if self.type == 'Monotonic': # Monotonic : 只增不減，一旦達到stop保持不變
            epoch_index = self.epochs
            if epoch_index >= stopped_epoch:
                self.beta = stop
            else:
                self.beta = start + epoch_index * slope

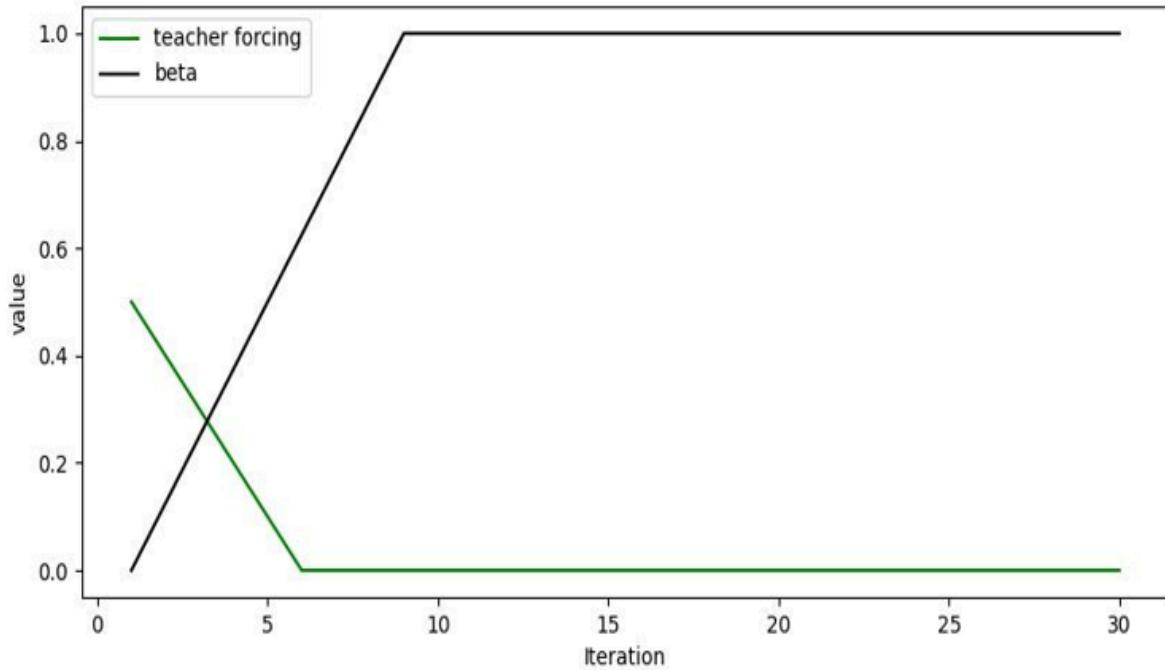
        elif self.type == 'Cyclical': # Cyclical : 週期性增減
            epoch_index = self.epochs % self.anneal_cycle
            if epoch_index >= stopped_epoch:
                self.beta = stop
            else:
                self.beta = start + epoch_index * slope
        else:
            # None : 不使用 KL annealing
            self.beta = 1.0

```

2. Analysis & Discussion

2.1.1 Plot Teacher forcing ratio

將初始tfr設定為0.5，並且從第二個epoch起開始下降，每次下降步長為0.1。故會在第6個epoch時，tfr會歸零。



本圖為使用Monotonic訓練30 epoch(第8個epoch時beta value會到達1.0)

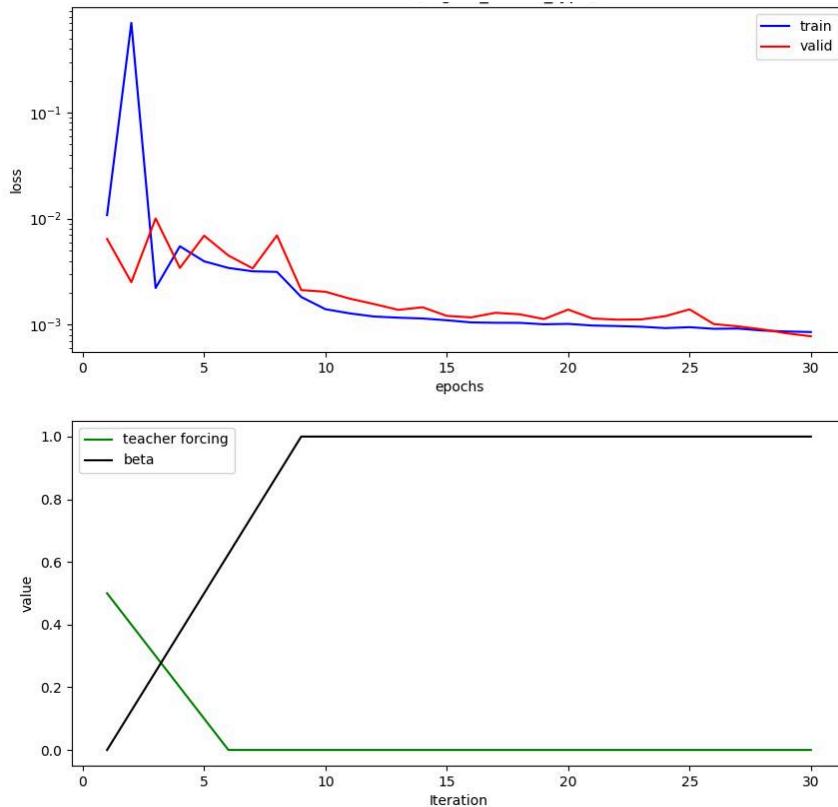
2.2 Plot the loss curve while training with different settings.

實驗參數設定：

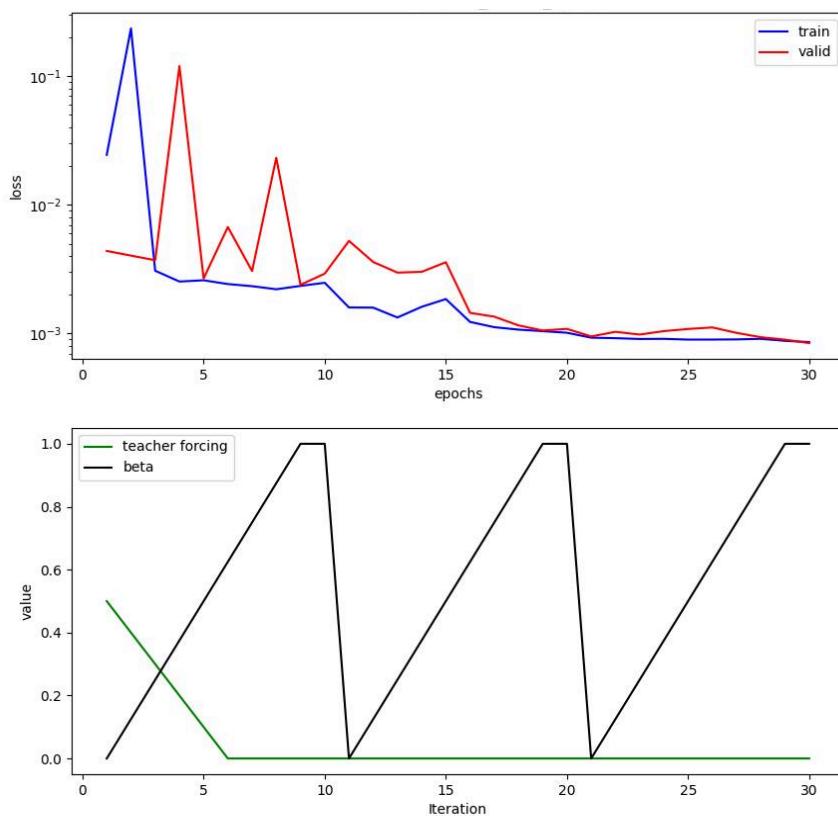
KL Annealing cycle = 10、kl_stop = 1.0、ratio = 0.8、lr = 0.001、epoch = 30
batch size = 2、tfr = 0.5、tfr_d_step = 0.1(第二個epoch開始下降)，並且搭配
Adam作為優化器並採用ReduceLROnPlateau，若是超過5個epoch PSNR值在Validation
沒有提升時，會降低學習率。

2.2.1 Compare & Analyze the difference between them

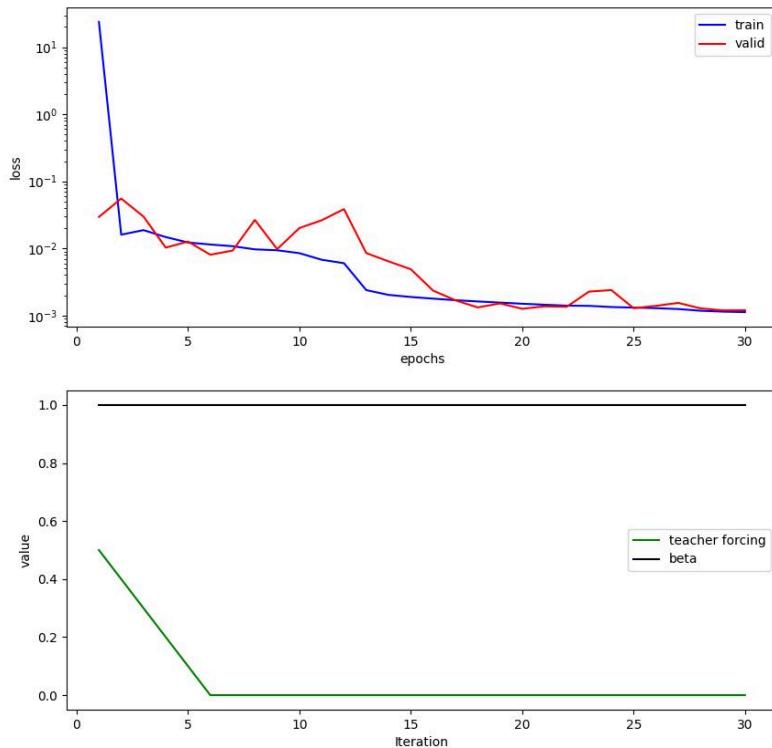
a. With KL annealing (Monotonic)



b. With KL annealing (Cyclical)

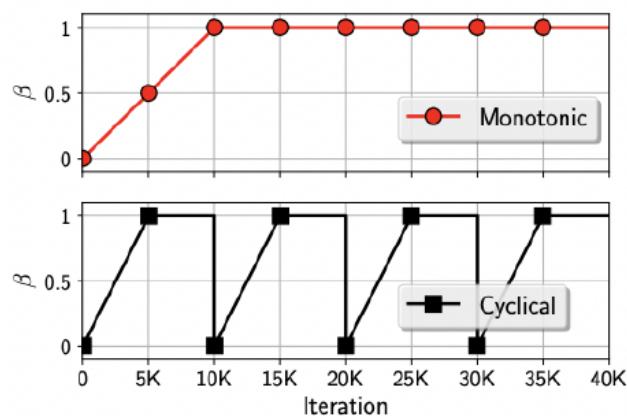


c. Without KL annealing



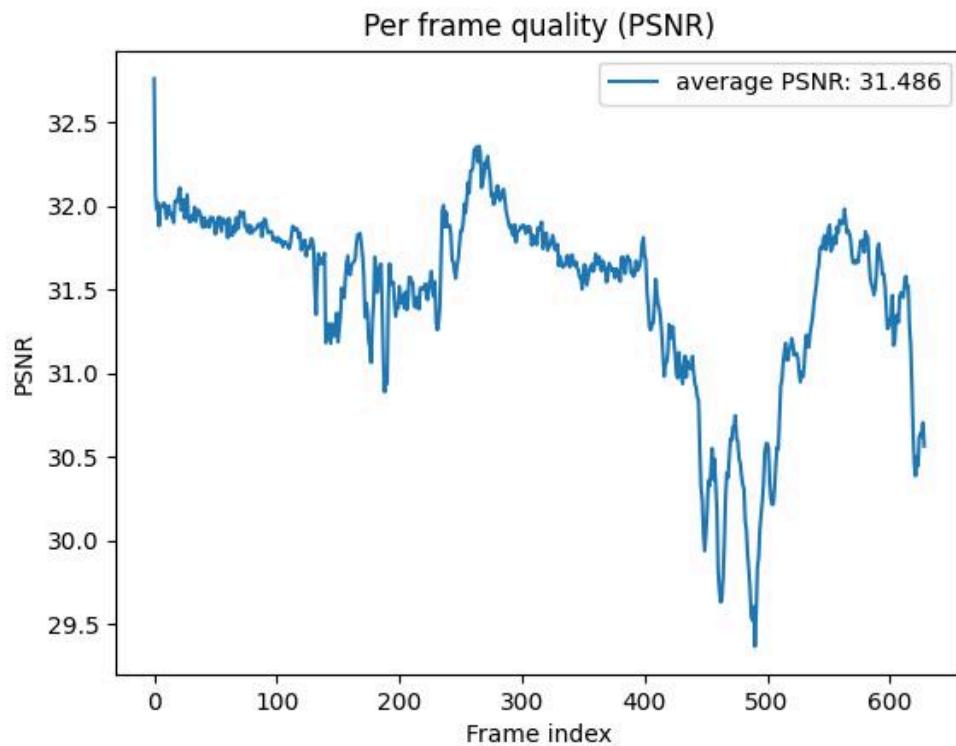
d. Compare

以這三種方法來看，Monotonic在前30圈中驗證集之損失函數擁有較為穩定的 loss curve，且也表現在2.3的數據中，是在前30Epoch裡面擁有較佳PSNR值的方法。然而這並不能夠武斷地推斷何者較優，畢竟如同Lab4 spec slide裡面出現的圖，可發現每次動態調整beta的幅度其實是挺小(如0.001)，但本lab因為設備、時間因素僅能先以30圈作為比較，故普遍調整beta步長較大(0.125)，故無法相提並論。不過仍可發現在Monotonic、Cycle兩個方法中，在進入第二個epoch時，training loss會爆升，但Without則是在第一圈時有極大的Loss value。此現象我們可由公式解釋，因為前兩者初始beta value為0，當beta不為0時 beta所乘上的KL即會和前一圈有極大的落差，而不使用前兩者的方法也是因為beta為1會造成第一圈的Loss值會比較大。

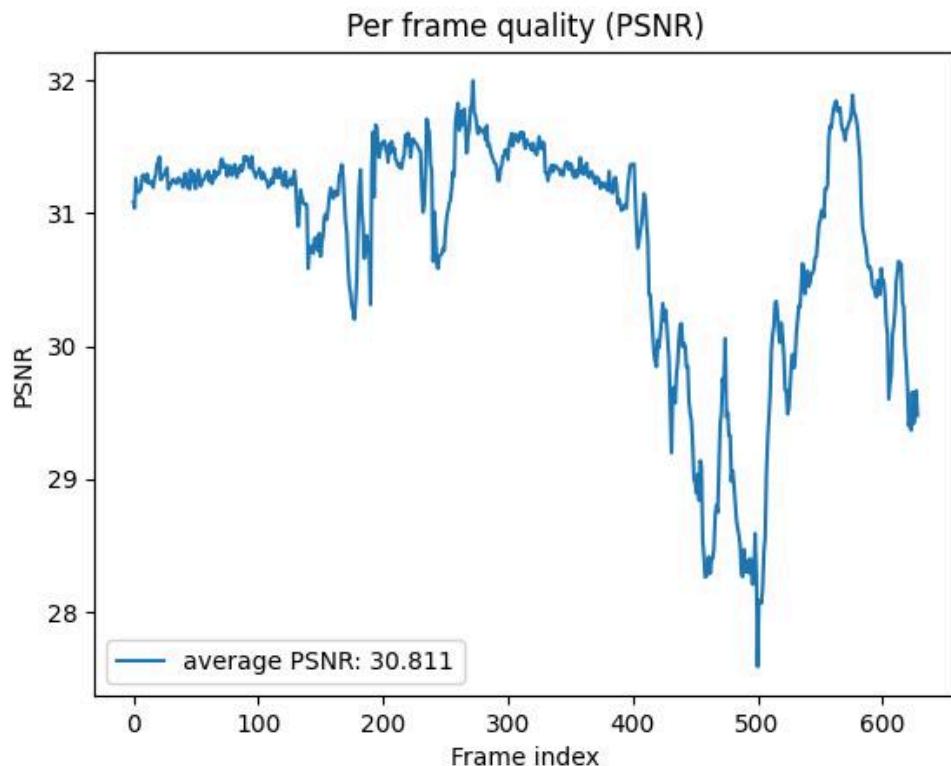


2.3 Plot the PSNR-per frame diagram in validation dataset

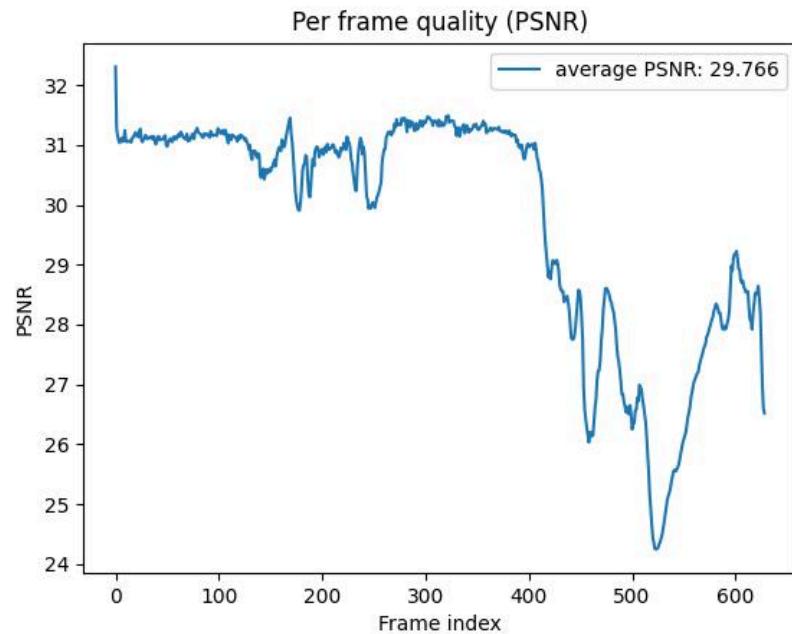
a. Monotonic Method (Epoch = 30時 , avg_psnr = 31.486)



b. Cycle Method (Epoch = 30時 , avg_psnr = 30.811)

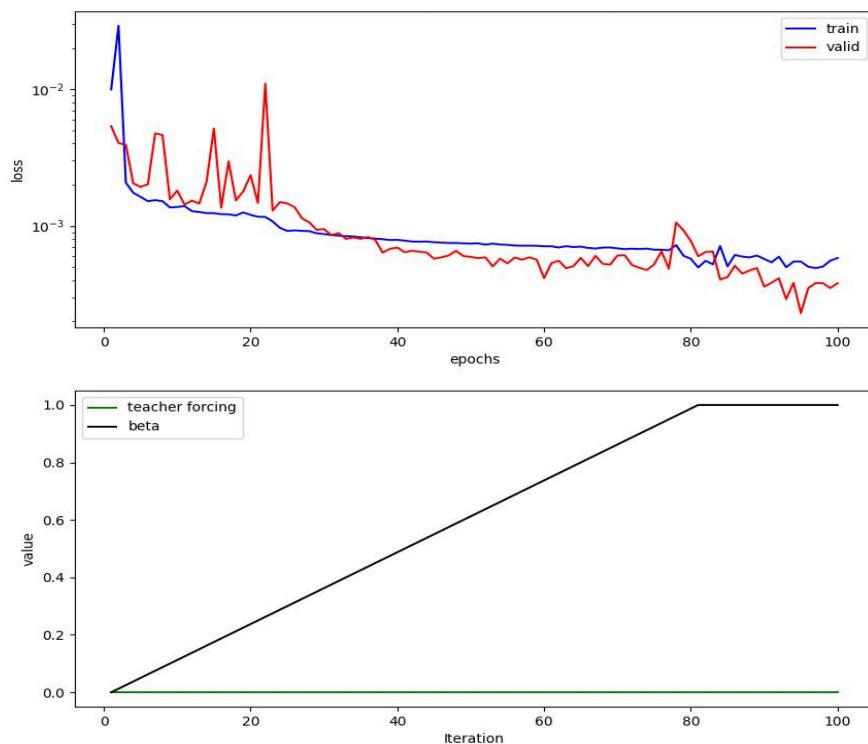


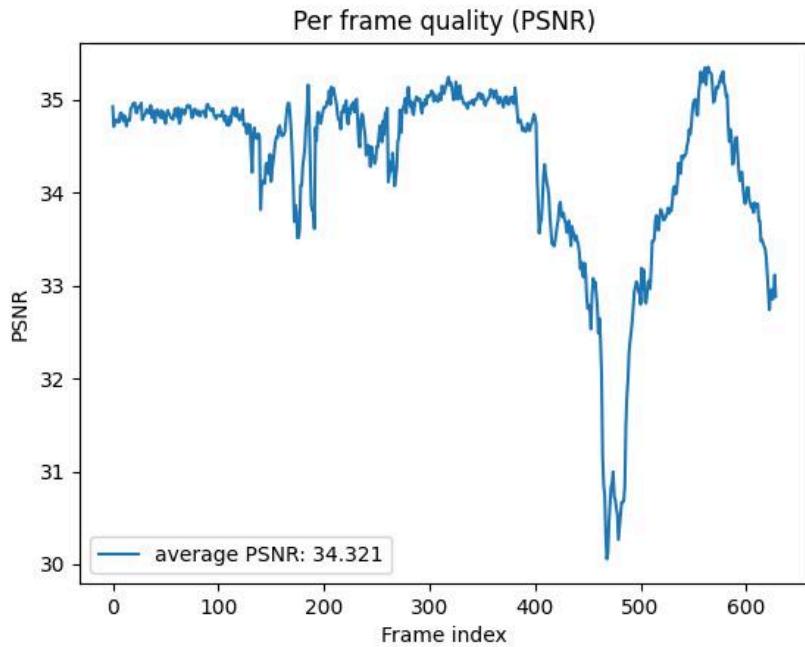
c. Without KL Annealing(Epoch = 20時 , avg_psnr = 29.766)



d. Best Presentation that I try (Cycle)

(維持先前設定，僅調整**cycle = 100**、**ratio = 0.8**，並設定random seed(699811706) to make the results can be reproduced，並於Epoch = 100時，avg_psnr = 34.321)





e. 發現：

由上面各種組合的PSNR值可看出，不論哪種方法，在400–500Frame Index區間中PSNR有明顯的暴跌，這我們從val資料集中發現在這區間內，跳舞者將手舉過頭頂(Y字形)，推斷是因為動作較大且是先前較無出現過的大動作，因此這段區間PSNR值有明顯暴跌。

