Lab4 313605019 方敏

A. Introduction

Lab4 實作基於 VAE（Variational AutoEncoder）的影像序列生成模型，透過輸入一張初始影像與後續每一幀對應的 label，逐步生成接下來的每一幀影像序列。並比較在不同的 KL Annealing（Monotonic、Cyclical、None）與 Teacher Forcing 調整下，觀察 Loss Curve、Teacher Forcing Ratio Curve、PSNR-per-frame 進行評估分析。

B. Implementation details

i. How do you write your training/testing protocol

在 Trainer.py 裡實作了 training_one_step 以及 val_one_step，training_one_step 負責在每個 frame 中，根據上一幀影像與當前標籤進行當前幀的預測，並計算 loss 後進行模型參數更新。

一開始先參考助教在 Tester.py 裡 val_one_step 的做法，先將資料調整維度，把影像與標籤從 [B, seq, C, H, W]調整為 [seq, B, C, H, W]，B 代表 batch size，seq 代表 frame 連續長度，C 代表圖片通道數 (RGB 為 3)，H 跟 W 代表圖片高度與寬度，轉換後讓 seq 在第一個，讓模型方便依時間順序處理每一幀。

因為是依據當前的幀與下張標籤來預測下張幀，因此我先設定第一個 frame 為 img[0]，迴圈內從 i = 1 開始，以符合因為是根據當前幀與下一張幀對應標籤來預測下一張幀，因此在一開始我將 img[0] 作為初始輸入的幀（第 0 幀），在接下來的迴圈中，從 i = 1 開始讀取第 i 幀標籤逐步向後進行，迴圈最後再將預測出來的幀作為下一 loop 輸入幀，這樣就能保證第 i - 1 幀及第 i 幀的對應標籤作為輸入，預測出第 i 幀的影像。

在迴圈內先將第 i-1 幀與第 i 幀對應標籤經過 RGB_Encoder 與 Label_Encoder 進行特徵轉換，再送入 Gaussian_Predictor 來預測潛在空間分布，得到 mu(平均值)、logvar(變異數)、透過 reparameterization trick 得到的 z(隨機向量)，以引入隨機性，讓模型具備生成能力；再將特徵轉換過的幀與標籤以及 z 送進 Decoder_Fusion 融合提取特徵，最後把特徵輸入到 Generator 後解碼即為預測的第 i 幀。

接下來判斷是否啟用 Teacher Forcing ，如果有就把 Ground Truth

的第 i 幀替代剛剛預測出的幀，可以有效減少誤差累積並讓模型更快收斂，然後計算 mse(均方誤差，預測與真實影像差異)、kl(潛在分佈與標準正態分佈的差異)、beta(從 KL Annealing 得到的 KL loss 權重係數 β)，總損失為 mse + beta * kl，加總到 total_loss 中。結束全部預測後，針對 total_loss 進行反向傳播，再呼叫 optimizer 進行梯度更新，並回傳平均損失。

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

    img = img.permute(1, 0, 2, 3, 4)   # [seq, B, C, H, W]
    label = label.permute(1, 0, 2, 3, 4)

    decoded_frame = img[0]
    total_loss = 0

    for i in range(1, self.train_vi_len):
        label_feature = self.label_transformation(label[i])
        frame_feature = self.frame_transformation(decoded_frame)

        z, mu, logvar = self.Gaussian_Predictor(frame_feature, label_feature)

        decode_feature = self.Decoder_Fusion(frame_feature, label_feature, z)
        decoded_frame = self.Generator(decode_feature)

        if adapt_TeacherForcing:
            decoded_frame = img[i]

        mse = self.mse_criterion(decoded_frame, img[i])
        kl = kl_criterion(mu, logvar, self.batch_size)
        beta = self.kl_annealing.get_beta()
        total_loss += mse + beta * kl

    total_loss.backward()
    self.optimizer_step()

    return total_loss / (self.train_vi_len - 1)
    # raise NotImplementedError
```

val_one_step 負責模型驗證階段的評估，邏輯與 training_one_step 相同，用第 i-1 幀及第 i 幀標籤預測出第 i 幀，不同的是驗證沒有做反向傳播跟參數更新，也沒有判斷是否使用 Teacher Forcing，直接使用預測出的幀，以及 z 不是透過高斯分佈參數抽樣，是從 randn 隨機產生的，還有為了要畫出 loss curve 與 PSNR_per_frame，除了回傳平均 mse 之外，還有回傳 psnr list(每一幀的重建品質)。

在 eval 中，我加入了若判斷為最後一個 epoch 則畫出 loss curve 與 PSNR_per_frame，畫出 teacher_forcing 則是加在 training_stage 中。

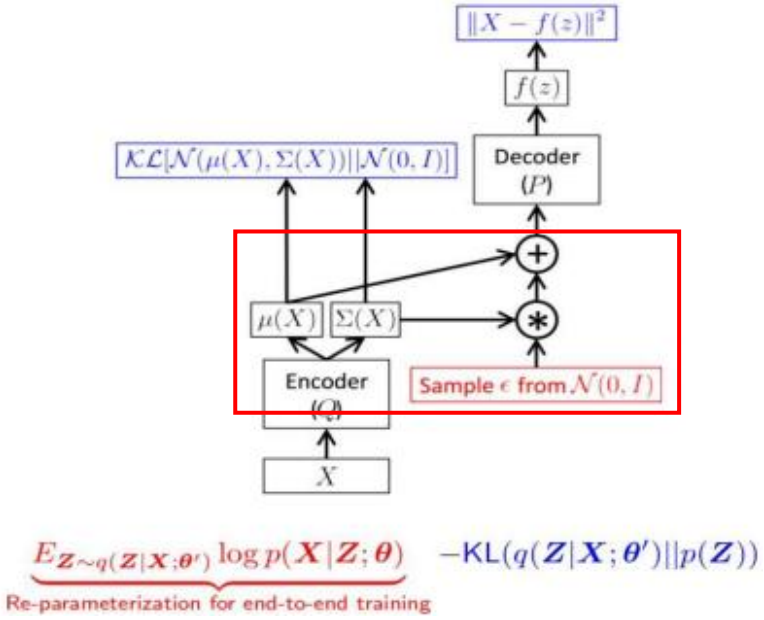
```python
def val_one_step(self, img, label):
    # self.eval()
    img = img.permute(1, 0, 2, 3, 4)   # [seq, B, C, H, W]
    label = label.permute(1, 0, 2, 3, 4)

    decoded_frame = img[0]
    total_loss = 0
    psnr_sum = 0
    psnr_list = []
    for i in range(1, self.val_vi_len):
        label_feature = self.label_transformation(label[i])
        frame_feature = self.frame_transformation(decoded_frame)

        # z, mu, logvar = self.Gaussian_Predictor(frame_feature, label_feature)
        z = torch.randn(1, self.args.N_dim, self.args.frame_H, self.args.frame_W).to(self.args.device)
        decode_feature = self.Decoder_Fusion(frame_feature, label_feature, z)
        decoded_frame = self.Generator(decode_feature)

        mse = self.mse_criterion(decoded_frame, img[i])
        # kl = kl_criterion(mu, logvar, self.batch_size)
        # beta = self.kl_annealing.get_beta()
        total_loss += mse
        psnr = Generate_PSNR(decoded_frame, img[i]).item()
        psnr_sum += psnr
        psnr_list.append(psnr)

    avg_loss = total_loss / (self.val_vi_len - 1)

    return avg_loss, psnr_list
    # raise NotImplementedError
```

在 Tester.py 中，實作了 val_one_step，負責執行測試階段的影像生成，跟 Trainer.py 的 val_one_step 大致相同，最後加上助教提供的程式碼把初始 frame 加上產生的 629 張影像組合成 gif。

```python
def val_one_step(self, img, label, idx=0):
    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)
    assert label.shape[0] == 630, "Testing pose seqence should be 630"
    assert img.shape[0] == 1, "Testing video seqence should be 1"

    # decoded_frame_list is used to store the predicted frame seq
    # label_list is used to store the label seq
    # Both list will be used to make gif
    decoded_frame_list = [img[0].cpu()]
    label_list = []

    decoded_frame = img[0]

    for i in range(1, 630):
        label_frame = label[i]

        label_feature = self.label_transformation(label_frame)
        frame_feature = self.frame_transformation(decoded_frame)

        z = torch.randn(
            (decoded_frame.size(0), self.args.N_dim, decoded_frame.size(2), decoded_frame.size(3)),
            device=self.args.device)

        decode_feature = self.Decoder_Fusion(frame_feature, label_feature, z)
        decoded_frame = self.Generator(decode_feature)
        decoded_frame_list.append(decoded_frame.cpu())

        label_list.append(label[i].cpu())

    print(f"Total frames generated: {len(decoded_frame_list)}")
    print(f"Total labels processed: {len(label_list)}")

    # Please do not modify this part, it is used for visulization
    generated_frame = stack(decoded_frame_list).permute(1, 0, 2, 3, 4)
    label_frame = stack(label_list).permute(1, 0, 2, 3, 4)

    assert generated_frame.shape == (1, 630, 3, 32, 64), f"The shape of output should be (1, 630, 3, 32, 64), but your output shape is {generated_frame.shape}"

    self.make_gif(generated_frame[0], os.path.join(self.args.save_root, f'pred_seq{idx}.gif'))

    # Reshape the generated frame to (630, 3 * 64 * 32)
    generated_frame = generated_frame.reshape(630, -1)
```

ii. How do you implement reparameterization tricks

這段程式碼實作 reparameterization trick，為了讓隨機採樣的過程可以被梯度反向傳播優化，如下圖紅框部分，先將 logvar 轉換為標準差，再乘上與 std 相同形狀的隨機噪音，最後加上 mu，將不可微的隨機採樣轉換成可微分的運算。




```python
def reparameterize(self, mu, logvar):
    # TODO
    std = torch.exp(0.5 * logvar)
    eps = torch.randn_like(std)
    return eps * std + mu
    # raise NotImplementedError
```

iii. How do you set your teacher forcing strategy

在 teacher_forcing_ratio_update 中，設定了 Teacher Forcing 的遞減策略，一開始模型訓練會使用 Ground Truth 而不是產生的預測幀作為輸入（預設 self.tfr = 1.0），當訓練經過設定的 epoch，self.tfr_sde（預設第 10 個 epoch 之後）就會開始逐步將 Teacher Forcing 的機率降低，每次下降一個固定的步長 self.tfr_d_step（預設 0.1），可以讓模型逐漸從依賴 Ground Truth 到完全自行生成輸出，是一種典型的 Scheduled Sampling 策略。

```python
def teacher_forcing_ratio_update(self):
    if self.current_epoch >= self.tfr_sde:
        self.tfr = max(0, self.tfr - self.tfr_d_step)
```

iv. How do you set your kl annealing ratio

kl_annealing 負責在訓練過程中調整 VAE 中 KL 損失的權重係數 β。這次實作了三種 KL Annealing 策略：

Cyclical 的 β 會隨著 epoch 週期性從 start(預設 0)緩慢上升到 stop(預設 1)，再降為 start，形成一 cycle，速度由 ratio 控制，若速度太快則 β 會在 1 保持 fixed 直到要下降的 epoch，先從 epoch 總數除以 cycle 數算出一個 cycle 有多少 epoch，再用 ratio 判斷哪些 epoch 的 β 要上升哪些不用，n_cycle 預設為 10，ratio 預設為 1。

Monotonic 的 β 會從 0 隨 epoch 線性成長直到 1 為止，之後固定為 1，速度由 ratio 控制。

None 的 β 從頭到尾固定為 1，不使用任何 KL Annealing。

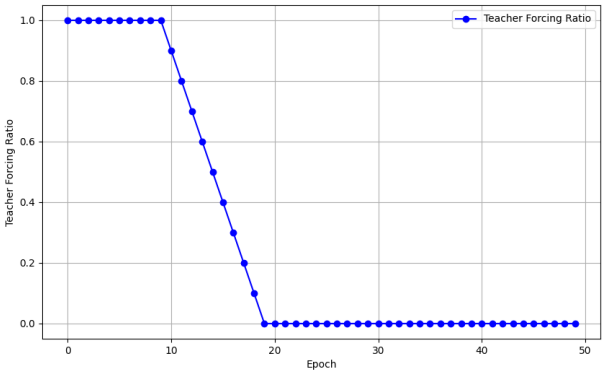
```python
class kl_annealing:
    def __init__(self, args, current_epoch=0):
        self.args = args
        self.current_epoch = current_epoch
        self.beta = 0.0

    def update(self):

        if self.args.kl_anneal_type == "Cyclical": # beta 週期性改變
            self.beta = self.frange_cycle_linear(n_iter=self.current_epoch,
                                                 start=0.0,
                                                 stop=1.0,
                                                 n_cycle=self.args.kl_anneal_cycle,
                                                 ratio=self.args.kl_anneal_ratio)

        elif self.args.kl_anneal_type == "Monotonic": # beta 增至 1
            self.beta = min(self.current_epoch * self.args.kl_anneal_ratio, 1.0)

        else: # beta 始終為 1
            self.beta = 1.0

        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):
        period = self.args.num_epoch / n_cycle
        cycle_progress = (n_iter % period) / period  # 計算當前週期位置
        step = (stop - start) / (period * ratio)  # 計算每步增加步長

        beta = min( cycle_progress * step, stop)
        return beta
```

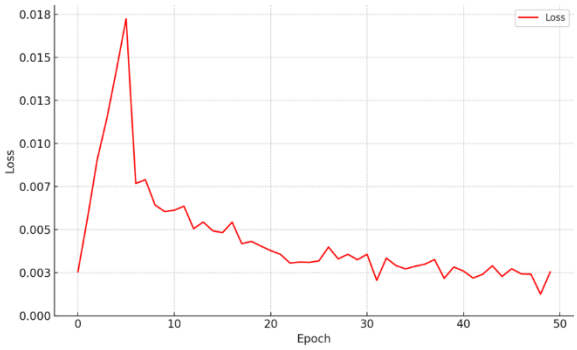
Plotting.py 負責畫圖

C. Analysis & Discussion

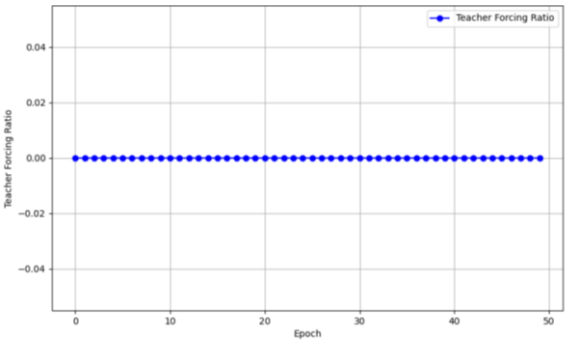
i. Plot Teacher forcing ratio

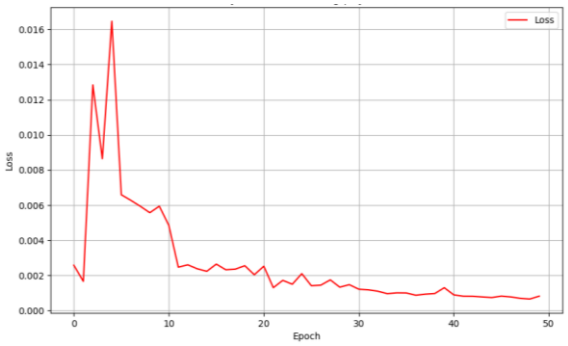
↓ Tfr：On | tfr = 1 | tfr_sde = 10 | tfr_d_step = 10





Tfr：Off

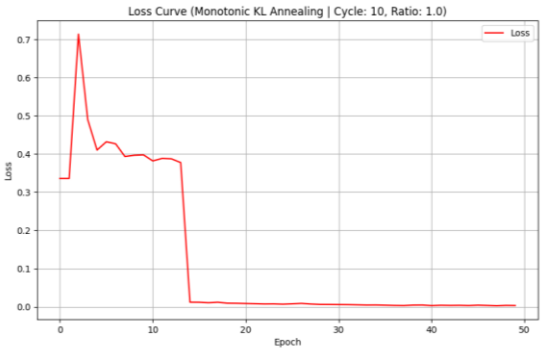


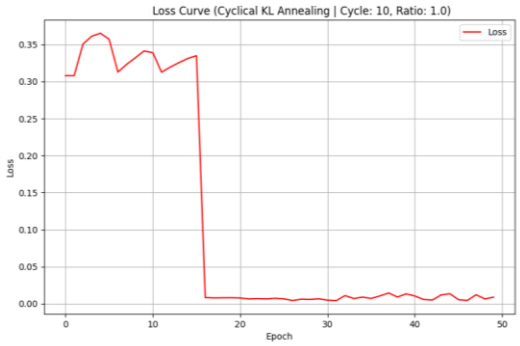


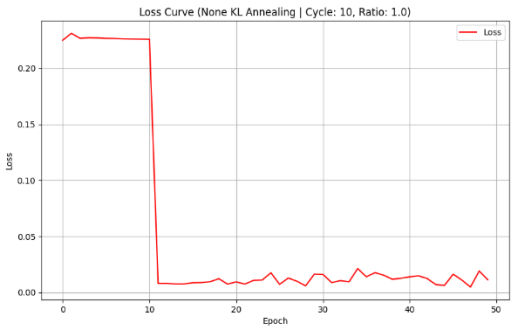
可以發現這次實驗中不使用 Tfr，效果會比較好，且在 kaggle 競賽中不使用 Tfr 的分數也比較高，可能是因為本次實驗中要生成的影片解析度跟內容都比較簡單，模型可以較容易學到 latent code，並有可能因為 distribution 造成 loss 下降效果差。

ii. Plot the loss curve while training with different settings.

Epoch = 50 | Cycle = 10 | Ratio = 1 | Tfr：On



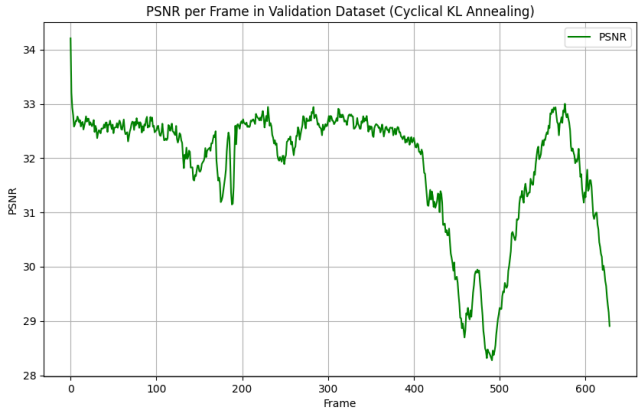




三種 KL Annealing Loss 在經過第 15 個 epoch 後都明顯降低，Monotonic 最穩定下降，其次是 Cyclical，最浮動的是不使用 KL Annealing，因為使用 KL loss 會幫助訓練時跳脫 local minimum。

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

Cyclical | Epoch = 50 | Cycle = 5 | Ratio = 0.5 | Tfr：Off



可以看到初期 PSNR，代表模型對初始幾張 frame 的預測相對容易，但中後期震盪幅度大，可能是因為預測當前幀是透過前一幀，因此錯誤也會被傳遞，導致 PSNR 越來越低。

改善方式：可以使用 better teacher forcing 策略。

iv. Other training strategy analysis

除了基本架構外，我還使用了學習率調整策略(Learning Rate Scheduling)，可以讓學習率在訓練過程中在指定的 epoch 降低(我是設乘上 0.1)，功能為當模型進入收斂階段時，適度降低學習率可以避免模型在 local minimum 附近震盪。

```
self.optim      = optim.Adam(self.parameters(), lr=self.args.lr)
self.scheduler  = optim.lr_scheduler.MultiStepLR(self.optim, milestones=[2, 5], gamma=0.1)
```

D. References

https://github.com/haofuml/cyclical_annealing/blob/master/plot/plot_schedules.ipynb

https://github.com/Bigpig4396/PyTorch-Variational-Autoencoder-VAE/blob/master/VAE.py

https://github.com/KJLdefeated/NYCU_DLP_2024/tree/main

https://github.com/steven112163/Deep-Learning-and-Practice

https://github.com/romanycc/NYCU_DLP_2023

https://github.com/cxyfer/NYCU_DLP

https://github.com/gyes00205/NYCU_DLP_2022

https://github.com/ray0727/Deep-Learning-and-Practice
https://github.com/cxyfer/NYCU_DLP/blob/main

https://github.com/youzhe0305/NYCU-DLP/blob/main