Lab 4 Conditional VAE for Video Prediction

314551022 廖洺玄

Introduction

在這個 Lab 中，我們實作了 VAE 的 cconditional video prediction generation 模型，主要任務是利用人體姿態 label 作為條件來生成連續的 frame。我們的模型架構參考了 ICML 2018 的論文"Stochastic Video Generation with a Learned Prior"，採用了多個關鍵模組：Gaussian_Predictor 進行後驗 distribution 預測以及 Generator 生成最終的視頻 frame。整個系統在訓練時採用 teacher forcing 策略，通過真實幀和預測幀的混合輸入來平衡訓練穩定性與生成多樣性。KL annealing 的策略也對訓練過程產生影響，我們實作了三種不同的 annealing 方式：Cyclical、Monotonic 和無 KL annealing，並分析它們對模型收斂性和生成品質的影響。透過 reparameterization trick，模型能夠有效在連續潛在空間中採樣。在這個 Lab 我也透過 LLM decode my code 和在利用 LLM 來教我原論文中的架構和想法還有 plot image 的 function 也是他協助幫我做的。

Implementation Details

Training Protocol

在 Training 的時候我會一個個 frame 的連續方式去根據上個 frame generate 下個，然後第一個會用 ground truth。當進行第 i frame 的預測時，如果用 teacher forcing，則使用真實的前一 frame 作為輸入；否則使用模型在前一步生成的 frame 作為輸入。每一步的 forward pass 過程包括將前一幀通過 RGB sampling，將當前姿勢 label 通過 encoder 提取特徵，然後將兩個特徵融合後送入 Gaussian Predictor 預測潛在變數的分佈參數，最後通過 Decoder Fusion 和 Generator 生成當前 frame。

訓練的損失函數結合了 MSE 和 KL 散度損失，其中 MSE 衡量生成 frame 與 ground truth 之間的差異，KL 散度損失則用來約束潛在變數的分佈接近先驗分佈。為了平衡兩個損失項，實作中採用不同 KL type 調整 KL 損失權重，並且加入 teacher forcing ratio 的衰減機制，讓模型在訓練初期依賴真實 frame，隨著訓練進行逐漸學會使用自己生成的 frame。此外，為了訓練穩定性，程式中還包含了梯度裁剪、數值穩定性檢查以及 NaN/Inf 值。

```python
def training_one_step(self, img, label, adapt_TeacherForcing):
    #  sequential training step
    self.optim.zero_grad()
    seq_len = img.size(1)
    generated_frames = []
    all_loss = 0
    for i in range(1, seq_len):
        if i == 1:
            prev_img = img[:, 0] # at the first step, use the first frame as previous frame
        else:
            if adapt_TeacherForcing:
                prev_img = img[:, i - 1] # use the true previous frame
            else:
                prev_img = generated_frames[-1].detach()

        current_label = label[:, i]
        target_frame = img[:, i]

        # generate the current frame
        output, mu, logvar = self.forward(prev_img, current_label, mode='train')
        generated_frames.append(output)
        # calculate the loss
        mse_loss = self.mse_criterion(output, target_frame)
        kl_loss = kl_criterion(mu, logvar, img.size(0))
        frame_loss = mse_loss + self.kl_annealing.get_beta() * kl_loss

        # Check for NaN in training loss
        if torch.isnan(frame_loss) or torch.isinf(frame_loss):
            print(f"Warning: NaN or Inf in training loss at epoch {self.current_epoch}, frame {i}")
            return torch.tensor(0.0, device=img.device, requires_grad=True)

    all_loss += frame_loss
```

Loss Function 這裡我用論文給的 mse+kl loss 去算出整個 frame 的 loss 後交給 optimizer 去做 backpropagation 給 model weight。

```python
# calculate the loss
mse_loss = self.mse_criterion(output, target_frame)
kl_loss = kl_criterion(mu, logvar, img.size(0))
frame_loss = mse_loss + self.kl_annealing.get_beta() * kl_loss
```

Testing Protocol

測試階段的任務是從單一初始 frame 預測出連續的 629 frame，總共生成 630 frame 的完整 video sequence。測試過程用 autoregressive 的方式，就是每次生成新 frame 時都用前一次的結果當 input，這樣可以測試 model 在長序列生成的穩定性。 生成過程中，model 先把初始 frame 加到結果 list 裡，然後跑 loop 生成剩下的 629 frame。每次 iteration 會用當前的 pose label 和前一 frame 作為 input，跑 feature extraction、latent sampling 和 decode 這些步驟來產生新 frame。跟 training 不一樣的地方是，測試時 latent variable z 直接從 standard normal distribution 採樣，這樣生成結果會比較多樣化。

```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 = []  # Include the initial frame in predictions
    label_list = []

    # Add the initial frame first
    decoded_frame_list.append(img[0].cpu())

    # Generate 629 consecutive frames (predict remaining frames)
    for i in range(0, 629):
        # Get current label
        current_label = label[i]

        # Encode current label
        label_feature = self.label_transformation(current_label)

        # Encode previous frame (first frame is provided, then use generated frames)
        if i == 0:
            prev_frame = img[0]  # Use provided first frame for first prediction
        else:
            prev_frame = decoded_frame_list[-1].to(self.args.device)  # Use last generated frame

        frame_feature = self.frame_transformation(prev_frame)

        # Use learned distribution parameters instead of random noise
        z, mu, logvar = self.Gaussian_Predictor(frame_feature, label_feature)

        # Fuse features
        fused_feature = self.Decoder_Fusion(frame_feature, label_feature, z)

        # Generate new frame
```

Reparameterization tricks

我的 Reparameterization tricks 是用標準公式 $z = \mu + \sigma * \varepsilon$，$\sigma = \exp(\text{logvar}/2)$，$\varepsilon$ 從 N(0,1) 採樣。這樣 randomness 被分離到 ε 上面，gradient 可以正常透過 μ 和 σ 做 backprop，又保持 VAE 的隨機性。 用 log variance 而不是直接 output variance 是因為可以確保 variance 永遠正，數值也較穩定。我在 train 的時候發現沒做這個限制 loss 會炸掉，所以這個很重要。

```python
    def reparameterize(self, mu, logvar):
        # todo makes the model and use differentable to gradient descent
        # z = mu + torch.exp(logvar / 2) * torch.randn_like(mu)
        std = torch.exp(logvar / 2)
        eps = torch.randn_like(std)
        return mu + eps * std
```

Teacher forcing strategy

我實作中，Teacher Forcing 的控制用動態衰減的比例來做，一開始 Teacher Forcing ratio 設成 1.0，代表 training 初期完全用真實的前一 frame 當 input 來預測當前 frame。
隨著 training 進行，從第 50 個 epoch 開始，Teacher Forcing ratio 會每個 epoch 減 0.1，一直衰減到 0.0，這時 model 就完全靠自己生成的前一 frame 來預測。
這種漸進式策略讓 model 可以平滑地從 supervised learning 過渡到自主生成。Training 的每一步，程式會根據當前 Teacher Forcing ratio 隨機決定要不要啟用，如果啟用就用真實前一 frame，不然就用 model 前一步生成的 frame。這樣設計避免 training 和 testing 階段分佈差異的問題，因為 model 在 training 後期已經習慣用自己生成的結果當 input，提高長序列生成的穩定性。我在調這個參數的時候發現如果衰減太快 model 會不穩定，太慢又學不好，所以這個 schedule 很重要。

```python
def training_stage(self):
    for i in range(self.args.num_epoch):
        train_loader = self.train_dataloader()
        epoch_train_losses = []

        for (img, label) in (pbar := tqdm(train_loader, ncols=160)):
            img = img.to(self.args.device)
            label = label.to(self.args.device)
            adapt_TeacherForcing = True if random.random() < self.tfr else False
            loss = self.training_one_step(img, label, adapt_TeacherForcing)
            epoch_train_losses.append(loss.detach().cpu().item())

            beta = self.kl_annealing.get_beta()
            if adapt_TeacherForcing:
                self.tqdm_bar('train [TeacherForcing: ON, {:.1f}], beta: {}'.format(self.tfr, beta), pbar, loss.detach().cpu(), lr=self.sc
            else:
                self.tqdm_bar('train [TeacherForcing: OFF, {:.1f}], beta: {}'.format(self.tfr, beta), pbar, loss.detach().cpu(), lr=self.s

        # Store epoch metrics
        avg_train_loss = sum(epoch_train_losses) / len(epoch_train_losses)
        self.train_loss_history.append(avg_train_loss)
        self.tfr_history.append(self.tfr)
        self.beta_history.append(self.kl_annealing.get_beta())

        if self.current_epoch % self.args.per_save == 0:
            self.save(os.path.join(self.args.save_root, f"epoch={self.current_epoch}.ckpt"))

        val_loss = self.eval()
        self.val_loss_history.append(val_loss)

        # Save training history
        self.save_training_history()

        self.current_epoch += 1
        self.scheduler.step()
        self.teacher_forcing_ratio_update()
        self.kl_annealing.update()
```

```python
def teacher_forcing_ratio_update(self):
    # 從第 tfr_sde 個epoch開始減少teacher forcing ratio
    if self.current_epoch >= self.tfr_sde:
        # Decay teacher forcing ratio
        self.tfr = max(0.0, self.tfr - self.tfr_d_step)
        print(f"Update teacher forcing ratio to {self.tfr}")
```

KL Annealing Strategy

我的 KL anneling 有循環退火和線性退火兩種策略。循環退火把 training 過程分成多個 cycle，每個 cycle 長度是 20 個 epoch，在每個 cycle 的前半段（annealing ratio 設 0.5）KL weight 從 0 線性增加到 1，後半段保持在 1。這種讓 model 可以週期性專注於 generation 品質，然後再逐步加強對 latent space distribution 的約束，有助於避免 posterior collapse 問題並提高生成多樣性。 線性退火比較簡單直接，KL weight 從 0 開始隨 training epoch 線性增加，到 1.0 後就保持不變。兩種策略各有優勢：循環退火讓 model 在 training 過程中多次探索重建和 regularization 的平衡點，特別適合複雜的生成任務；線性退火提供更穩定的 training 過程，適合需要快速收斂的場景。所以我表現最好的 model 是先用 linear 在用 cycle。

Cyclical Annealing:

```
def frange_cycle_linear(self, start=0.0, stop=1.0):
    # TODO
    cycle_progress = (self.current_epoch % self.kl_anneal_cycle) / self.kl_anneal_cycle
    # Apply ratio to determine how much of the cycle to use for annealing
    if cycle_progress <= self.kl_anneal_ratio:
        # Scale progress to full range within the annealing portion
        scaled_progress = cycle_progress / self.kl_anneal_ratio
        return start + (stop - start) * scaled_progress
    else:
        # Stay at max value for remaining portion of cycle
        return stop
```

Linear Annealing:

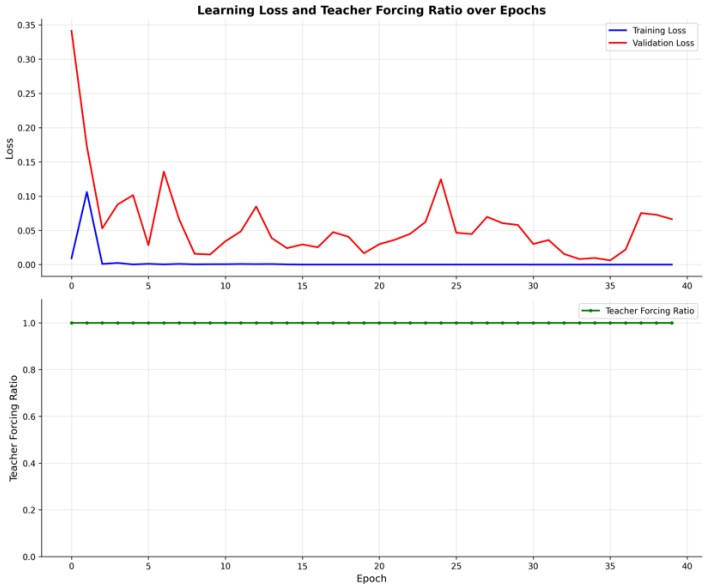
```
    elif self.kl_anneal_type == 'Linear':
        self.beta = min(1.0, self.beta + self.kl_anneal_ratio * self.current_epoch / self.kl_anneal_cycle)
```

Analysis & Discussion

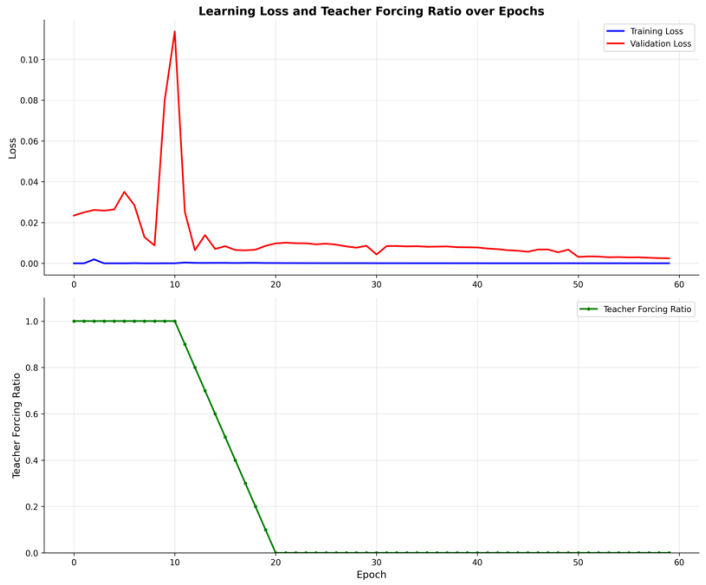
Plot Teacher forcing ratio

從這三個 stage 的 loss curve 可以看出 model 的訓練進展。Stage 1 時 Teacher Forcing ratio 保持 1.0，training loss 快速收斂但 validation loss 震盪較大（0.02-0.14）。Stage 2 是關鍵轉折點，Teacher Forcing ratio 從 1.0 線性下降到 0，validation loss 出現 spike 到 0.12 左右，這是 model 開始用自己生成的 frame 當 input 造成 distribution shift，但隨後快速穩定到 0.005。Stage 3 時 Teacher Forcing ratio 保持 0，validation loss 穩定在 0.001-0.003 之間，雖然有週期性震盪（可能是 KL annealing 造成），但顯示 model 已經學會穩定的自主生成，證明我的 Teacher Forcing schedule 設計有效。

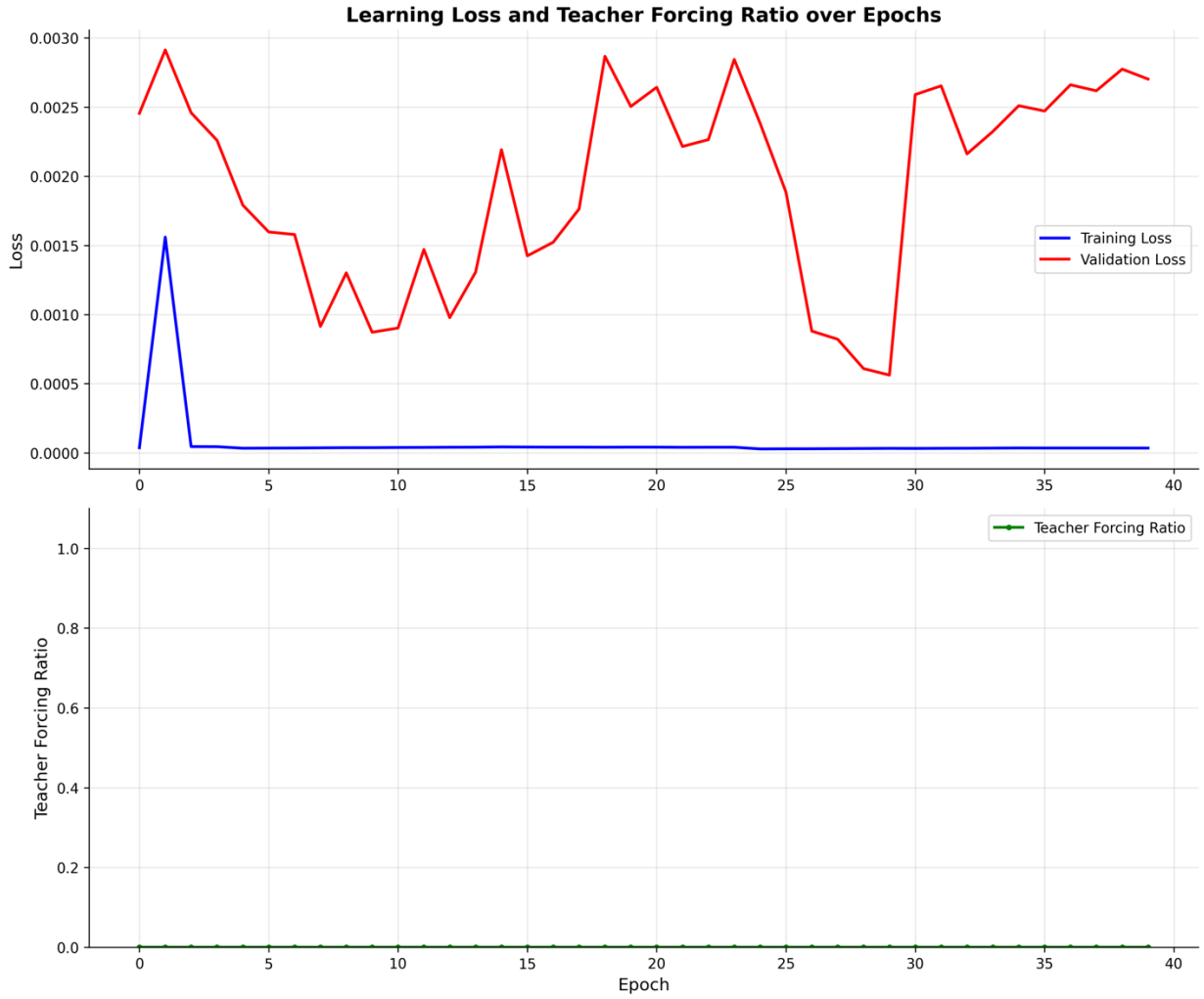
Stage1



Stage2



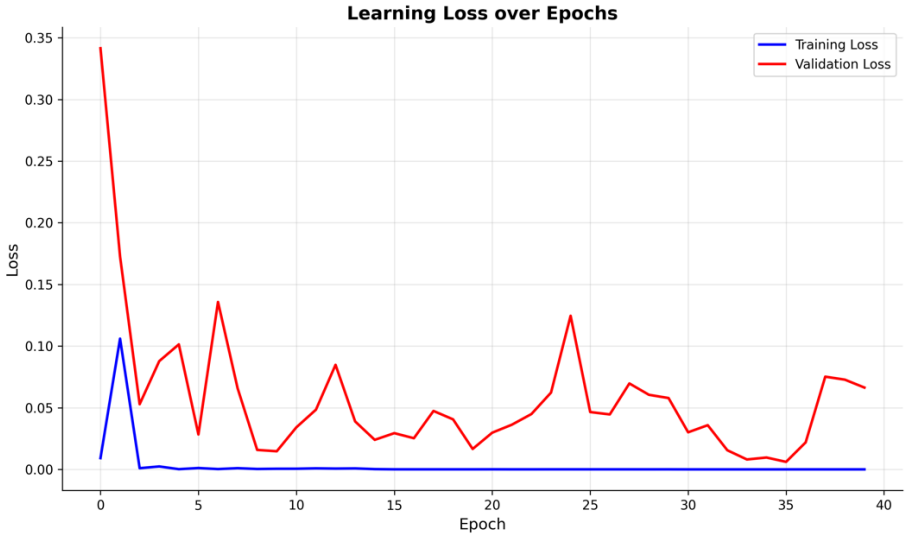
Stage3



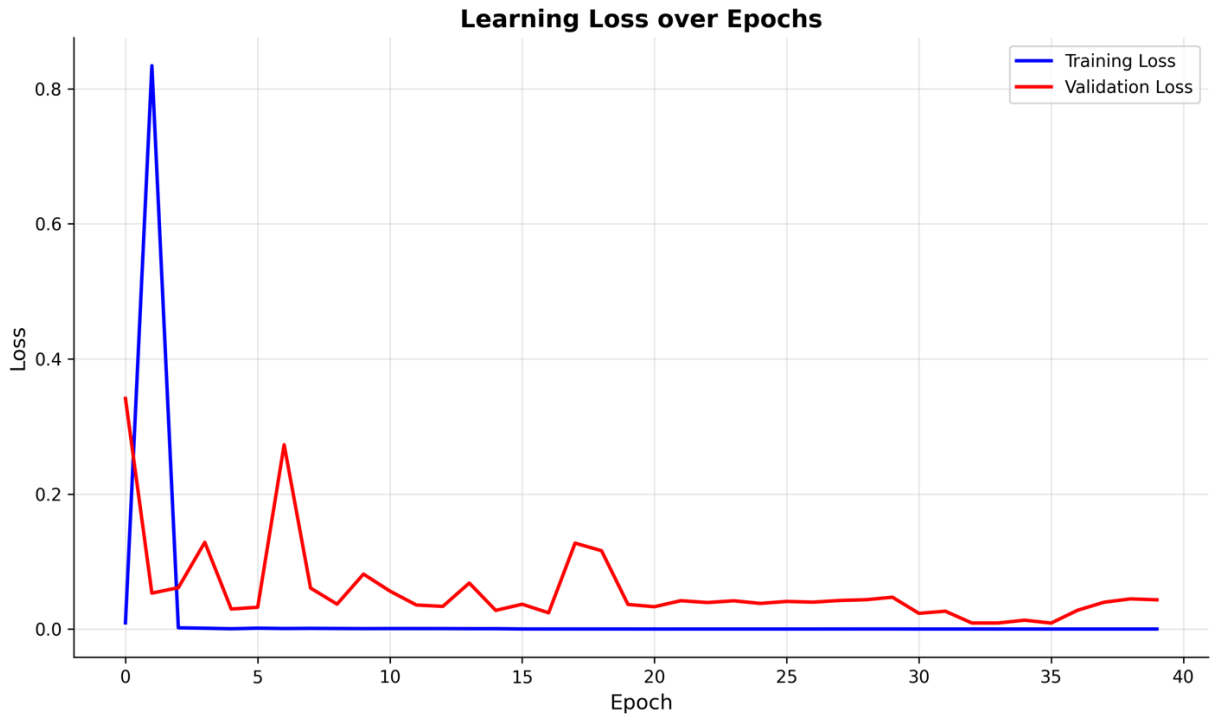
Plot the loss curve while training with different settings.

Without KL annealing training loss 初期 spike 最大到 8.5 然後快速收斂，validation loss 相對平穩；Monotonic 初期 spike 較小約 0.85，validation loss 有一些震盪特別在 epoch 5 左右；Cyclical 初期 spike 最小約 0.1 但 validation loss 呈現週期性震盪模式，在 epoch 5, 10, 25 等地方有規律峰值。從結果看，Without KL annealing 收斂最快但可能有 posterior collapse 問題，Monotonic 提供較穩定訓練，Cyclical 雖然有週期性震盪但能避免 KL vanishing，讓 model 在重建品質和 latent space regularization 之間找到更好平衡。我實作時發現 Cyclical 雖然訓練過程看起來不穩定，但最終生成品質比較好。

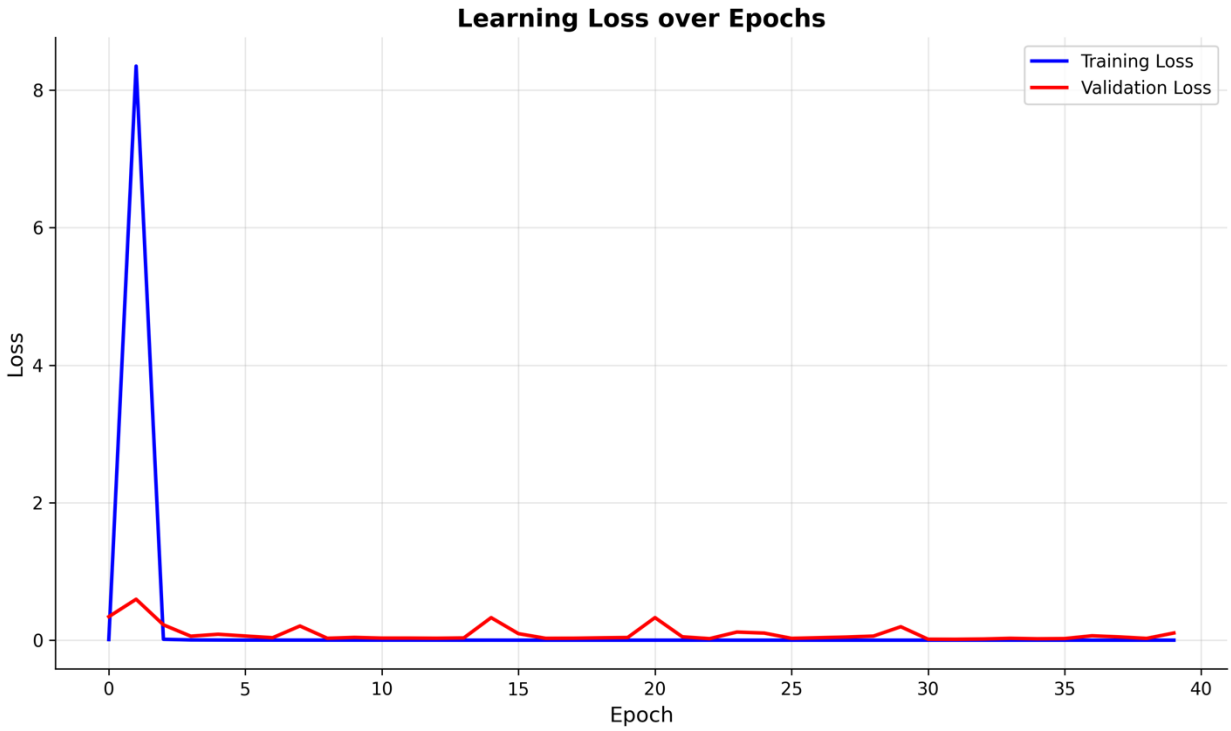
With KL annealing (Monotonic)



With KL annealing (Cyclical)

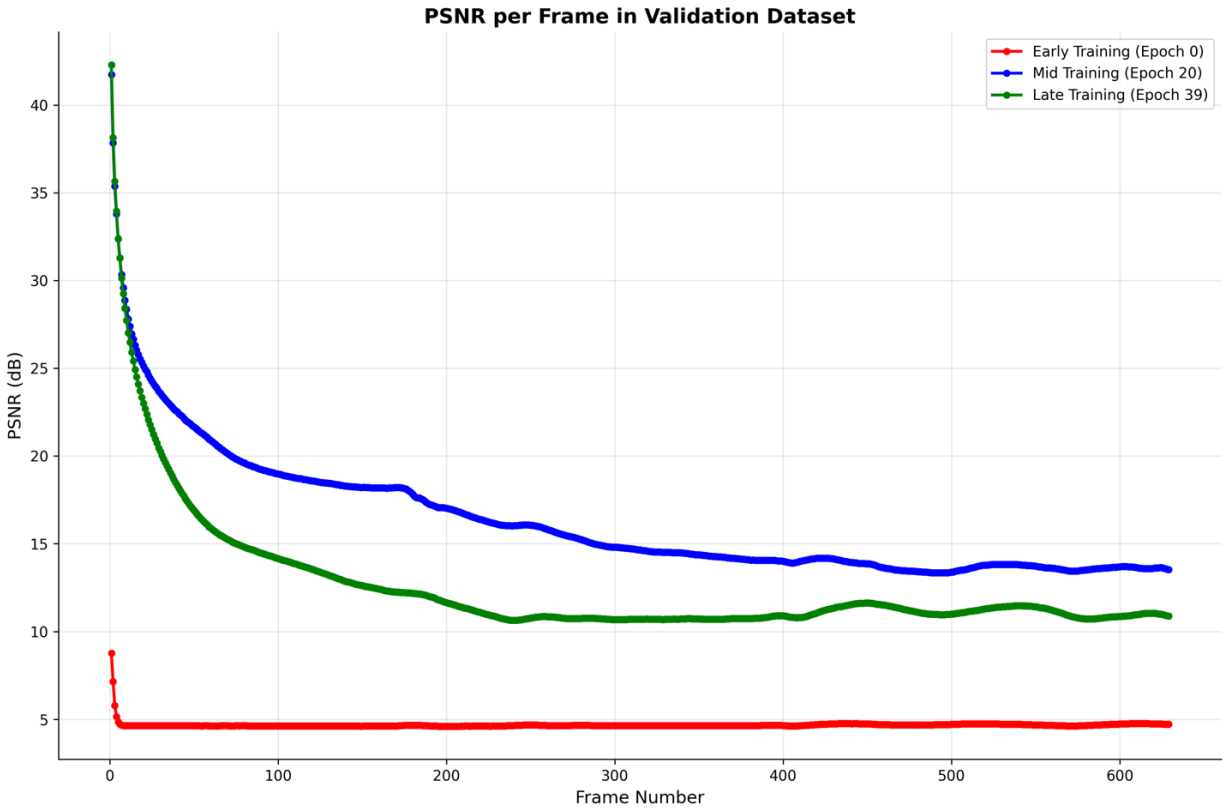


Without KL annealing

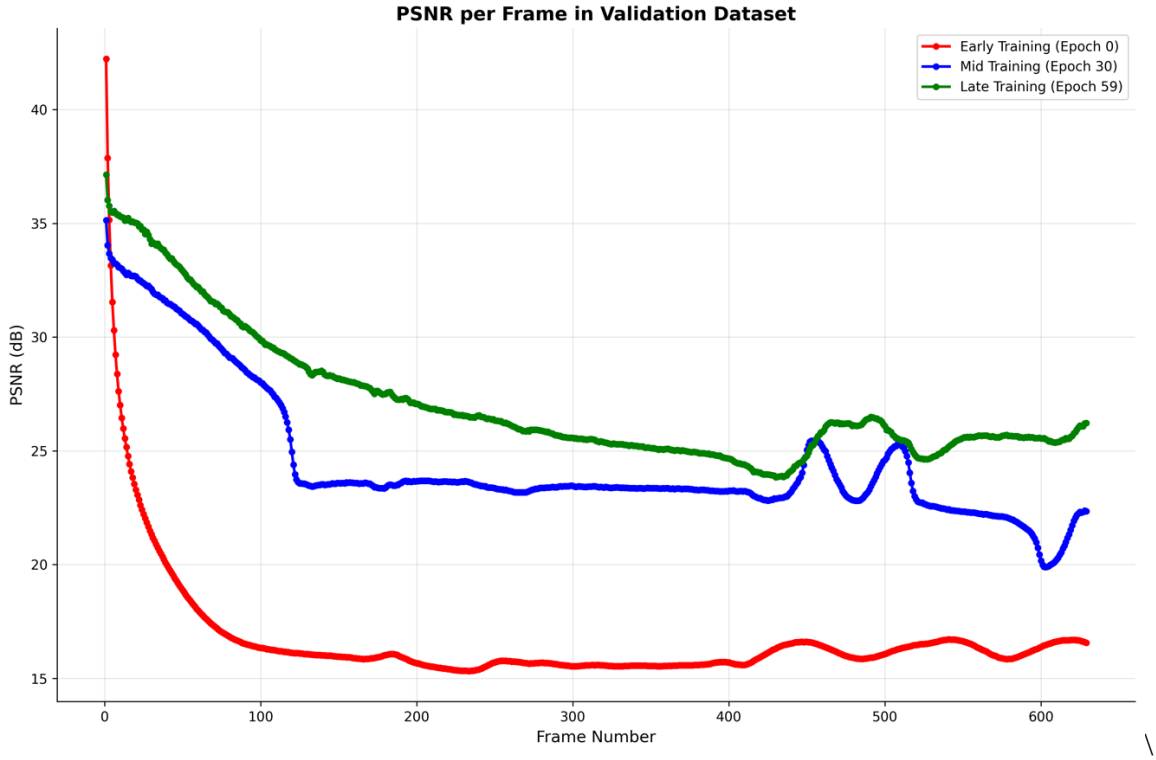


Plot the PSNR-per-frame diagram in the validation dataset and analyze it

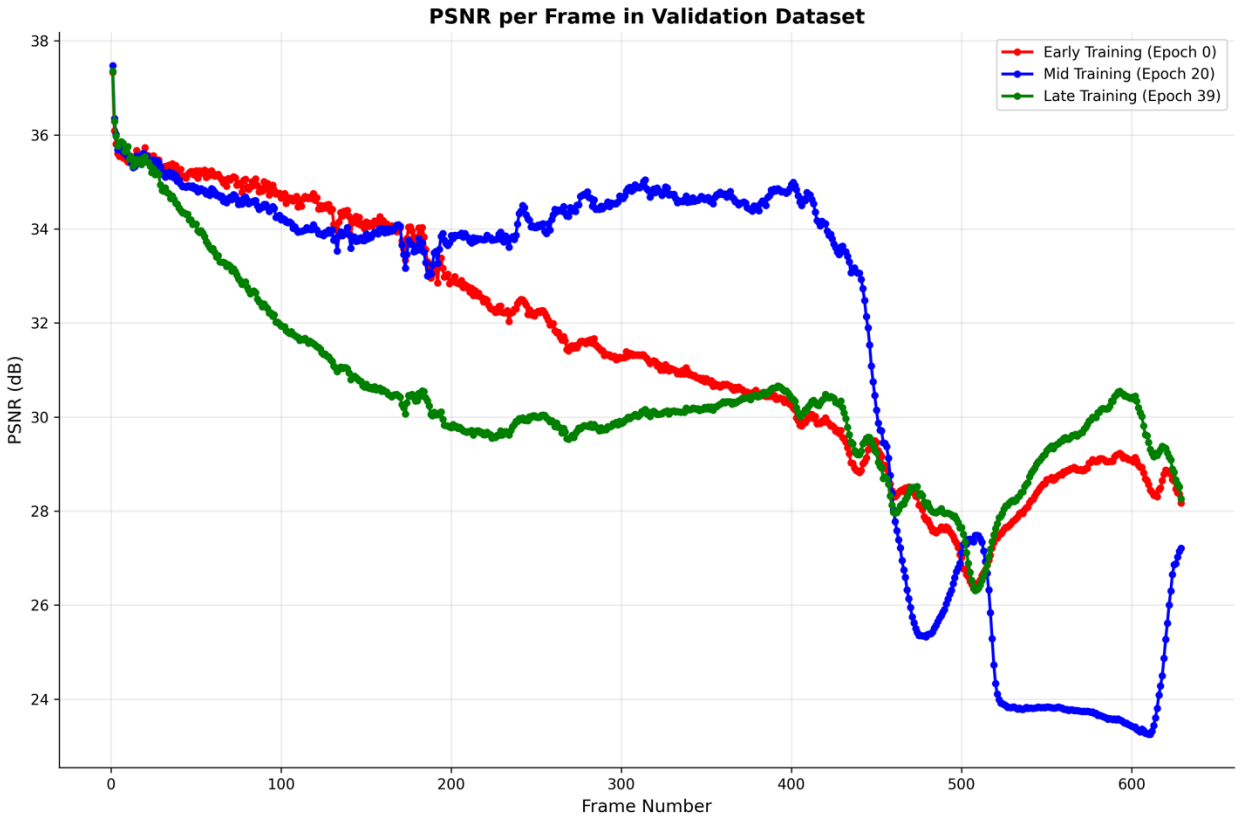
從這三個 stage 的 PSNR per frame 圖可以看出 model 生成品質的演進。Stage 1 顯示明顯學習差異，Early training 只有 5-9 dB，Mid/Late training 都從 40+ dB 開始逐漸下降到 10-15 dB，三條線趨勢相似說明 model 學會基本 frame generation（這時我用的是 KL Linear）。Stage 2 （這裡開始我用 KL Cycling）出現變化，Early training 穩定在 15 dB，但 Mid/Late training 比較不穩定，特別在後半段有波動，這可能是 Teacher Forcing ratio 變化造成 model 適應用自己生成的 frame 當 input。Stage 3 是最理想狀態，都從 37 dB 平滑下降到 22-25 dB 後稍微回升， model 已經穩定學會長序列生成。PSNR 隨 frame number 下降是正常的因為 error 累積，但 Stage 3 的穩定讓 model 達到收斂。
Stage1



Stage2

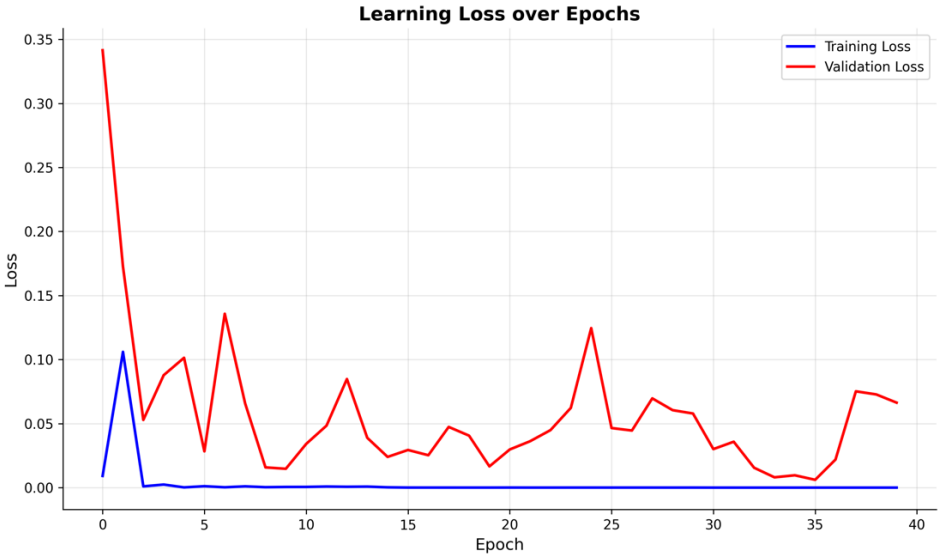


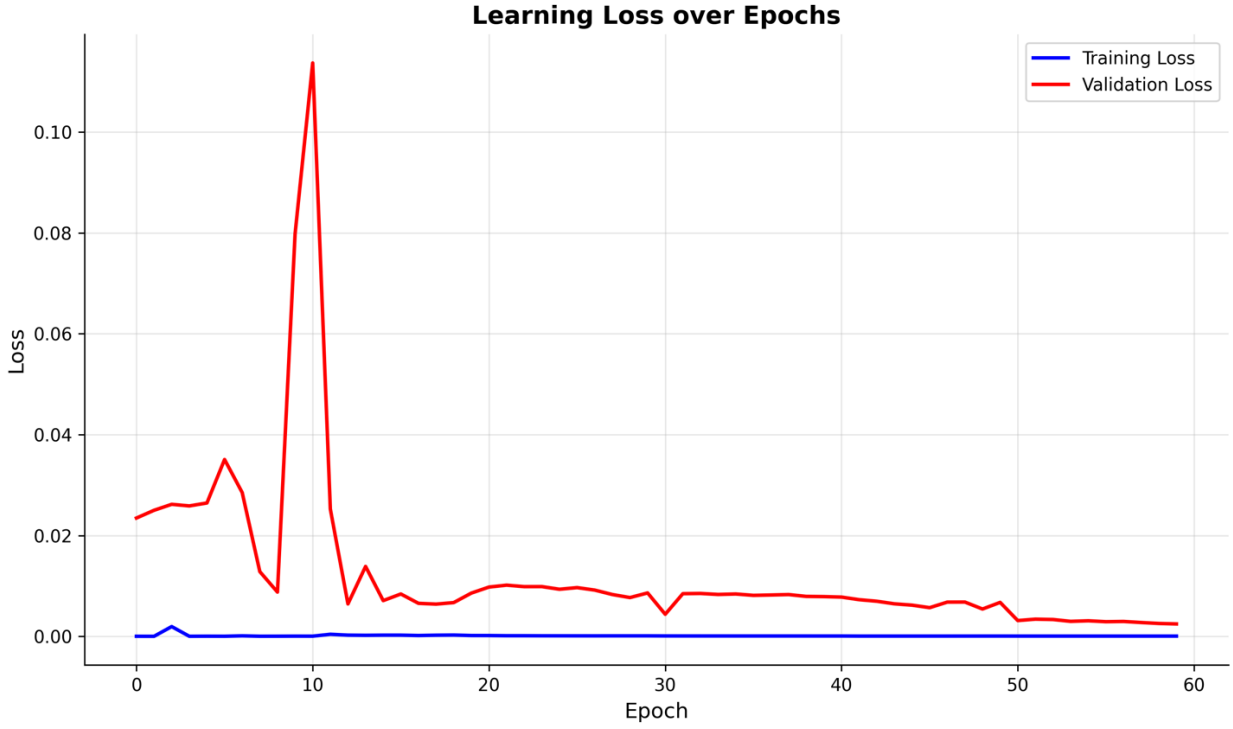
Stage3

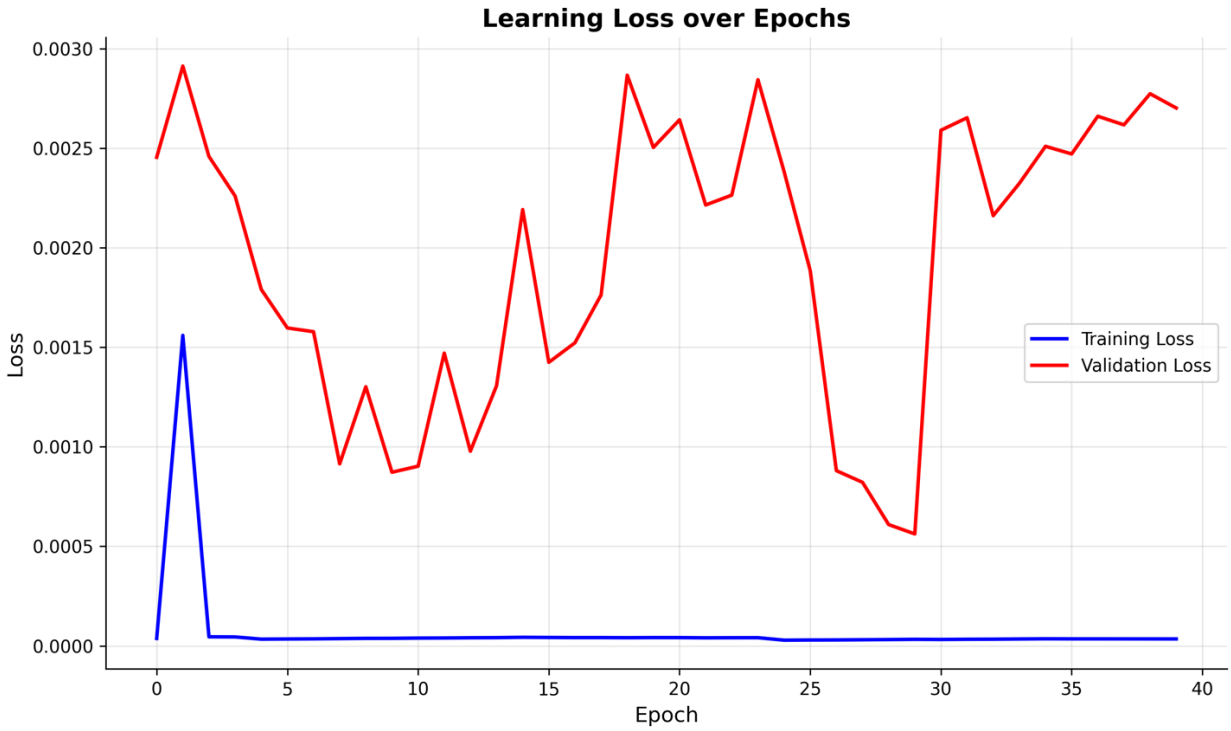


Other training strategy analysis

我一開始都只用跟 KL schedualing train 120 epoch 發現表現不太好，如果我用 cycling 的時候在一開始的時候都會 loss 下降太慢，而 linear 則是到後面的 model 表現都不太好沒辦法準確預測到下個 frame。所以我就改成 hybird，一開始先讓 linear KL 去把 model loss 降下來學到大概的 model feature，後面再用 cycling 兩次去抓到更細緻的 sampling 和防止 overfitting。然後我也調整策略，先用 60 epoch linear 在用兩個 40 epoch cycling 去讓模型 generate frame loss 達到收斂


Learning Loss over Epochs
Training Loss
Validation Loss
Loss
Epoch
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
5
10
15
20
25
30
35
40


Learning Loss over Epochs
Training Loss
Validation Loss
Loss
Epoch
0.10
0.08
0.06
0.04
0.02
0.00
0
10
20
30
40
50
60



Reference

1. https://arxiv.org/pdf/1808.07371
2. https://arxiv.org/pdf/1802.07687
3. https://arxiv.org/pdf/1903.10145
4. https://tomohiroliu22.medium.com/%E6%B7%B1%E5%BA%A6%E5%AD%B8%E7%BF%92paper%E7%B3%BB%E5%88%97-04-variational-autoencoder-vae-a7fbc67f0a2
5. https://blog.csdn.net/m0_58678659/article/details/130311901
6. https://github.com/hank891008/Deep-Learning