Deep learning　　Lab4 Report　陳品翰 313552042

A. Introduction (5%)

這次的 Lab 是要實做 Conditional VAE，在訓練的時候會有 image 跟 label，分別是畫面跟動作，在訓練時會把 image 跟 label 先轉換後給 Gaussian_Predictor，得到 z(latent)，這個 z 包含了 label 所沒有的畫面細節(光線、陰影、人物穿著….)，再用這個 z 跟轉換後的 image、label 傳進 Decoder_Fusion，他會融合三者資訊，最後把結果傳給 Generator 生出下一幀的圖片，在訓練時會用 beta 平衡 reconstruction 跟 KL loss 的比例，來達成生成圖片跟把分布趨近於標準常態分布。在 Test 時，則是用標準常態分布來抽取 z，並且依照上一幀的圖片跟目前這幀的 label，來預測下一幀的畫面。

B. Implementation details (30%)

i. How do you write your training/testing protocol (15%)

Trainer:

```
def training_one_step(self, img, label, adapt_TeacherForcing):
    # TODO
    self.optim.zero_grad()
    loss = 0
    x_hat = img[:, 0]
    for t in range(1, label.size(1)):
        pose = label[:, t]
        frame = img[:, t]

        x_feat = self.frame_transformation(x_hat)
        p_feat = self.label_transformation(pose)
        #z, mu, logvar = self.Gaussian_Predictor(frame, pose)
        z, mu, logvar = self.Gaussian_Predictor(x_feat, p_feat)
        z = z.expand(-1, -1, x_feat.shape[2], x_feat.shape[3])

        decoder_feat = self.Decoder_Fusion(x_feat, p_feat, z)
        x_hat = self.Generator(decoder_feat)

        recon_loss = self.mse_criterion(x_hat, frame)
        kl_loss = kl_criterion(mu, logvar, self.batch_size)
        loss += 0.85*(recon_loss) + 0.15*(self.kl_annealing.get_beta() * kl_loss)

        if adapt_TeacherForcing:
            x_hat = frame.detach()

    loss.backward()
    nn.utils.clip_grad_norm_(self.parameters(), 1.)
    self.optim.step()
    loss_value = loss.item()
    ######
    if not torch.isfinite(loss):
        print(f"Loss 非數值（NaN/Inf）出現，跳過此 batch")
        return torch.tensor(0.0, requires_grad=True, device=self.args.device)
    ######
    return loss
```

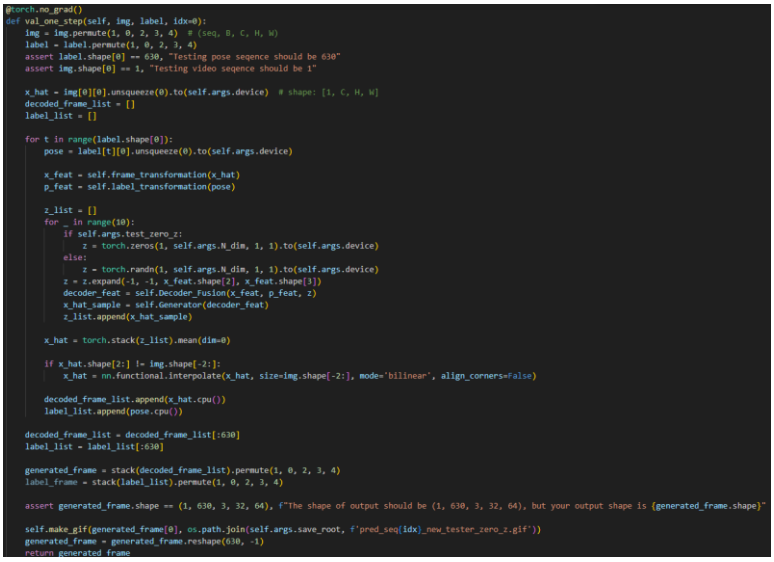
在訓練時，因為第一幀沒有預測的圖片，所以直接用原始圖片，之後進到 for loop，會先把上一幀的圖片跟目前的 label 做轉換，並且丟進 Gaussian_Predictor 來得到 Posterior 分布，並且用 reparameterization 得到 z(latent)，之後把轉換後的圖片跟 label、z 放進 Decoder_Fusion，以融合這些資訊，最後給 Generator 生成圖片。

Loss 的計算:用 MSE 去計算目前生成的圖片跟 ground truth 的差異，用 KL loss 去計算 z 的分布跟標準常態分布的差別，最後用 beta 來調

整兩者之間的比例，來達到最好的結果。

Teacher forcing 用來避免錯誤的累積，加速收斂。

Tester:



```python
@torch.no_grad()
def val_one_step(self, img, label, idx=0):
    img = img.permute(1, 0, 2, 3, 4)  # (seq, B, C, H, W)
    label = label.permute(1, 0, 2, 3, 4)
    assert label.shape[0] == 630, "Testing pose seqence should be 630"
    assert img.shape[0] == 1, "Testing video seqence should be 1"

    x_hat = img[0][0].unsqueeze(0).to(self.args.device)  # shape: [1, C, H, W]
    decoded_frame_list = []
    label_list = []

    for t in range(label.shape[0]):
        pose = label[t][0].unsqueeze(0).to(self.args.device)

        x_feat = self.frame_transformation(x_hat)
        p_feat = self.label_transformation(pose)

        z_list = []
        for _ in range(10):
            if self.args.test_zero_z:
                z = torch.zeros(1, self.args.N_dim, 1, 1).to(self.args.device)
            else:
                z = torch.randn(1, self.args.N_dim, 1, 1).to(self.args.device)
            z = z.expand(-1, -1, x_feat.shape[2], x_feat.shape[3])
            decoder_feat = self.Decoder_Fusion(x_feat, p_feat, z)
            x_hat_sample = self.Generator(decoder_feat)
            z_list.append(x_hat_sample)

        x_hat = torch.stack(z_list).mean(dim=0)

        if x_hat.shape[2:] != img.shape[-2:]:
            x_hat = nn.functional.interpolate(x_hat, size=img.shape[-2:], mode='bilinear', align_corners=False)

        decoded_frame_list.append(x_hat.cpu())
        label_list.append(pose.cpu())

    decoded_frame_list = decoded_frame_list[:630]
    label_list = label_list[:630]

    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}_new_tester_zero_z.gif'))
    generated_frame = generated_frame.reshape(630, -1)
    return generated_frame
```

先把第一幀傳進來，接著做預測，跟 train 一樣都會先對 image 跟 label 做轉換，之後我會對 z 抽樣 10 次，生成 10 張不同的圖片後，做平均，得到目前這幀的圖片，之後再用它來生成下一幀。

ii. How do you implement reparameterization tricks (5%)

```python
logvar = torch.clamp(logvar, min=-10, max=10)
std = torch.exp(0.5 * logvar)
eps = torch.randn_like(std)
return mu + eps * std
```

我是使用標準作法，但加入了 logvar 的數值裁減，避免 logvar 過大或過小導致不穩定。藉由先預測出 mean 跟 log-variance，再以可微分方式產生 latent vector z，模型可以在訓練階段對 posterior 分布進行有效學習。

iii. How do you set your teacher forcing strategy (5%)

```python
def teacher_forcing_ratio_update(self):
    # TODO
    if self.current_epoch >= self.tfr_sde:
        self.tfr = max(0.0, self.tfr - self.tfr_d_step)

    self.tfr = max(0.0, self.tfr - 0.005)  # 每一個 epoch 減少少量
```

每個 epoch 都會減 0.005，當超過設定的 self.tfr_sde 後，每次會減少 self.tfr_d_step 跟 0.005，其中的變數都是超參數，可以自己設定。

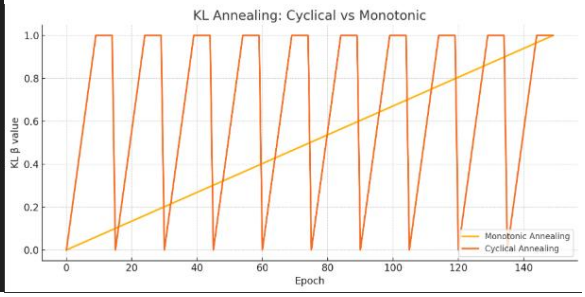
iv. How do you set your kl annealing ratio (5%)

```python
def __init__(self, args, current_epoch=0):
    # TODO
    self.args = args
    self.current_epoch = current_epoch
    self.beta = 0.0
    self.beta_list = []
    if args.kl_anneal_type == 'Cyclical':
        self.schedule = self.frange_cycle_linear(
            n_iter=args.num_epoch,
            start=0.0, stop=1.0,
            n_cycle=args.kl_anneal_cycle,
            ratio=args.kl_anneal_ratio)
    elif args.kl_anneal_type == 'Monotonic':
        self.schedule = np.linspace(0.0, 1.0, args.num_epoch)
    else:
        self.schedule = [1.0] * args.num_epoch

def update(self):
    # TODO
    if self.current_epoch < len(self.schedule):
        self.beta = self.schedule[self.current_epoch]
    else:
        self.beta = 1.0
    self.current_epoch += 1
    self.beta_list.append(self.beta)
def get_beta(self):
    # TODO

    return self.beta

def frange_cycle_linear(self, n_iter, start=0.0, stop=1.0,  n_cycle=1, ratio=1):
    # TODO
    L = np.ones(n_iter) * stop
    period = n_iter / n_cycle
    step = (stop - start) / (period * ratio)
    for c in range(n_cycle):
        v, i = start, 0
        while v <= stop and (int(i + c * period) < n_iter):
            L[int(i + c * period)] = v
            v += step
            i += 1
    return L
```



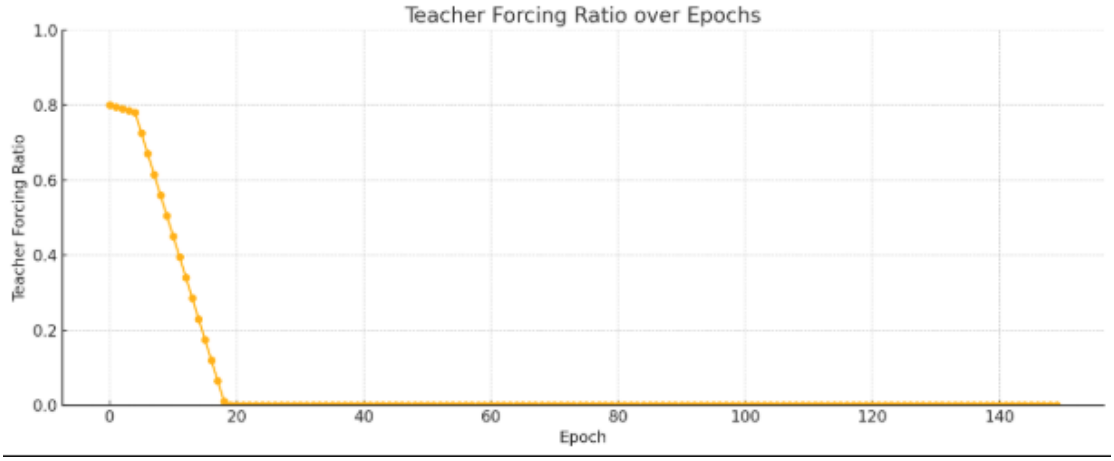
Cyclical 設定為:anneal_cycle:10 anneal_ratio:0.6

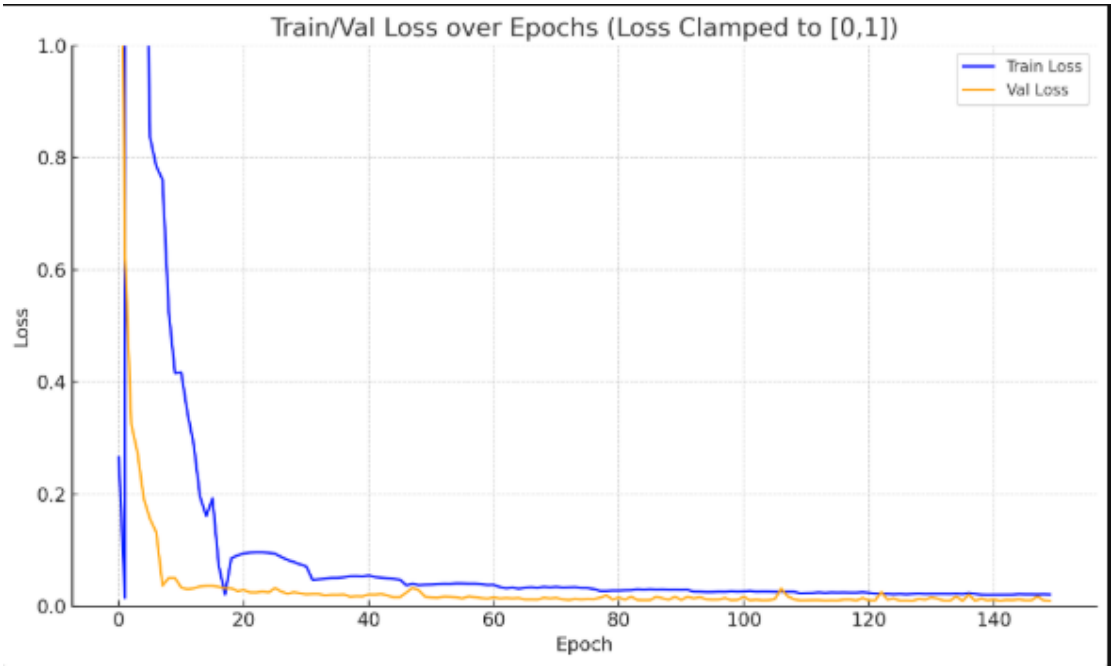
這代表整體訓練 150 epoch 中，會分成 10 個週期，每個週期前 60%（即 9 epoch）讓 β 緩慢線性上升，後 40%（6 epoch）則固定為 1。這樣可以有效交替聚焦在重建與 KL 正則化，並促進模型學習穩定的 latent space。

Monotonic:就是線性增加。

C. Analysis & Discussion (25%)

i. Plot Teacher forcing ratio (5%)





與 Loss 曲線的比較與分析 本次實驗採用了 monotonic decay 的 teacher forcing 策略。在前 19 個 epoch 中，teacher forcing ratio 從 0.8 緩慢下降至 0；第 19 epoch 之後完全關閉，模型改為完全依賴自身預測進行生成（fully autoregressive）。 我們將這樣的策略與 Loss 曲線進行比對分析：

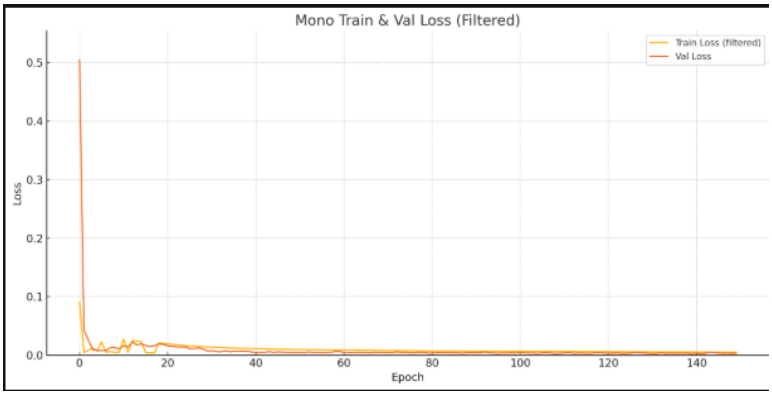
第 0～5 個 epoch：雖然此時 teacher forcing ratio 高於 0.73，但 loss 表現卻非常不穩定，甚至一度上升到超過 1.0，表示模型在初期仍處於收斂階段，即使有 ground truth 引導也無法有效學習。

第 6～18 個 epoch：teacher forcing 比例快速下降（從 0.67 下降到 0.01），同時 loss 曲線也呈現明顯下降趨勢，代表模型已逐步適應自身輸出的輸入條件，並在減少依賴 ground truth 的同時仍能穩定訓練。

第 19 個 epoch 之後：teacher forcing ratio 為 0，但 loss 持續維持在低且穩定的狀態，顯示模型已順利過渡到 autoregressive 模式，並能自行產生合理的預測結果。 本次分析說明：單調遞減的 teacher forcing 設計可有效讓模型從「依賴答案」過渡到「自主預測」，且不會導致效能下降。

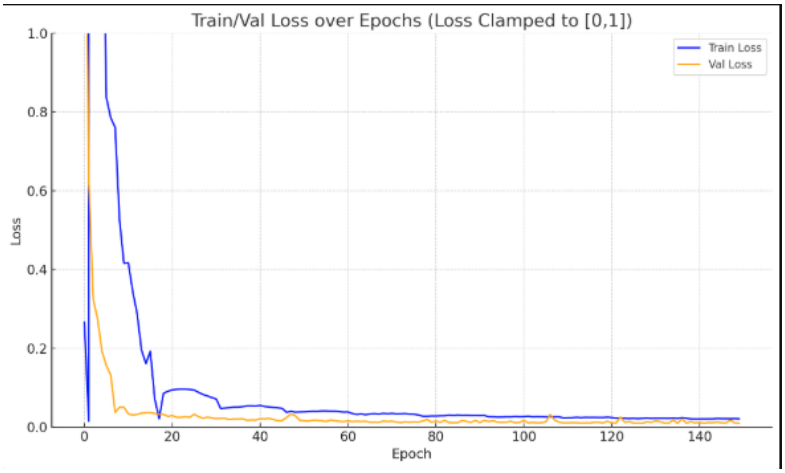
ii. Plot the loss curve while training with different settings. Analyze the difference between them (15%)

Monotonic:



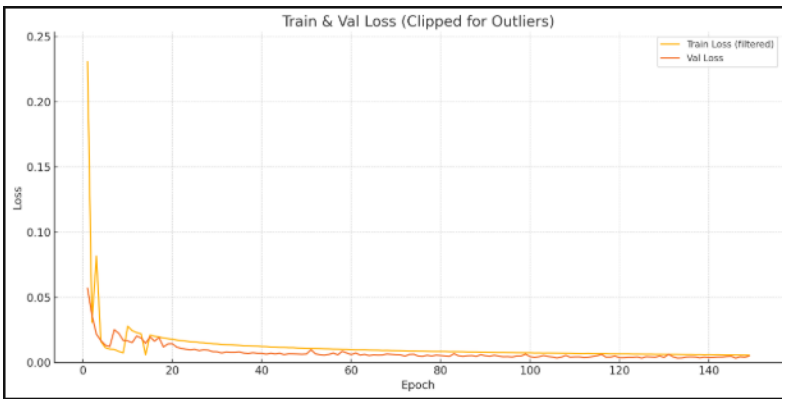
從圖中可觀察到，模型在訓練初期 loss 急速下降，於約第 10 個 epoch 後快速收斂，且 Train/Val Loss 曲線趨近一致，代表模型訓練穩定且無過擬合現象。由於 monotonic 策略會逐步將 KL 權重從 0 緩慢增大到 1，模型初期專注於重建損失，有助於學習高品質的視覺輸出，隨著 KL 權重提升，再逐步引導潛在變數對齊 prior 分布。整體而言是最穩定的訓練方式。

Cyclical:



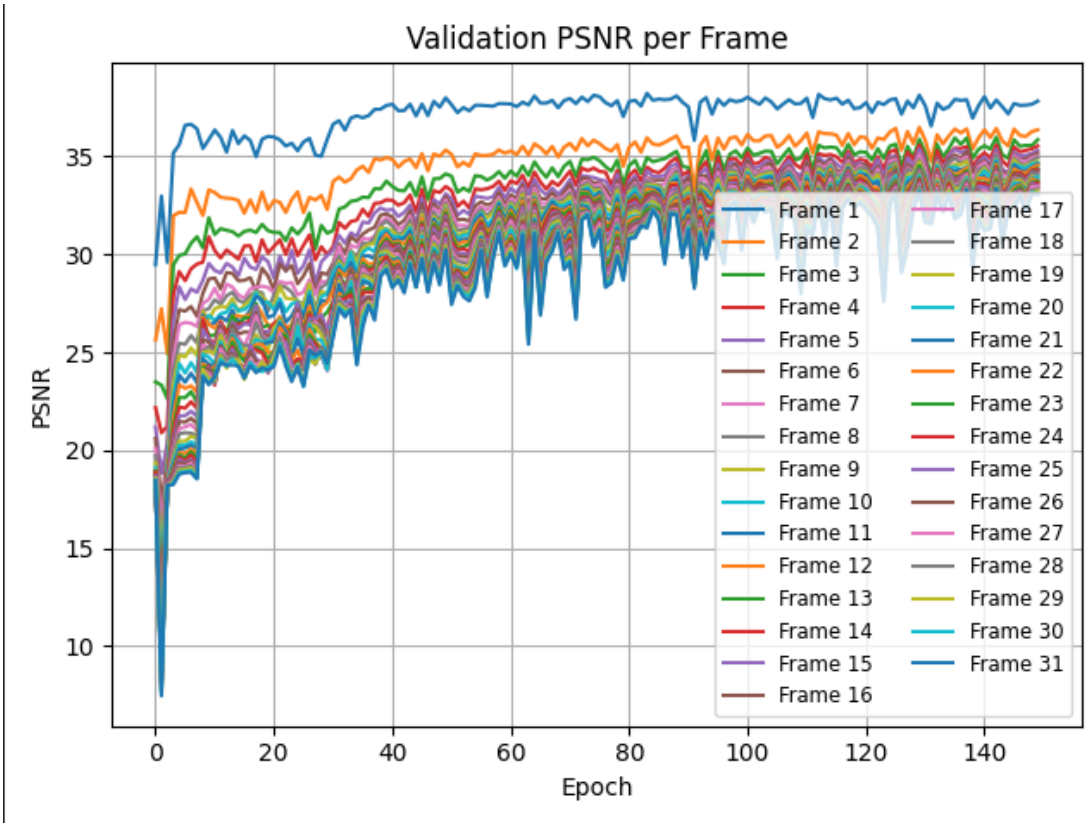
此策略的 loss 曲線在前 20 個 epoch 有較明顯的震盪，特別是訓練 loss 波動較大。這是因為 cyclical 策略會定期將 KL 權重重置為 0，再逐步增加到 1，如此循環進行。雖然初期不穩定，但這樣的設計能有效減緩 KL vanishing 問題，使模型探索更多潛在空間的可能性。從結果可見，最終 loss 與 monotonic 接近，代表長期學習效果良好，只是在穩定性與訓練效率上稍差。

Without:



無使用 KL annealing 時，模型直接套用 full KL 權重，導致初期 KL loss 壓過 reconstruction loss，造成潛在空間無法有效學習，容易產生 KL collapse 現象。雖然 loss 下降仍快速，但最終收斂效果明顯較差，且訓練與驗證 loss 出現 gap，表示模型可能過擬合或學習效果受限，實際生成品質也較不理想。

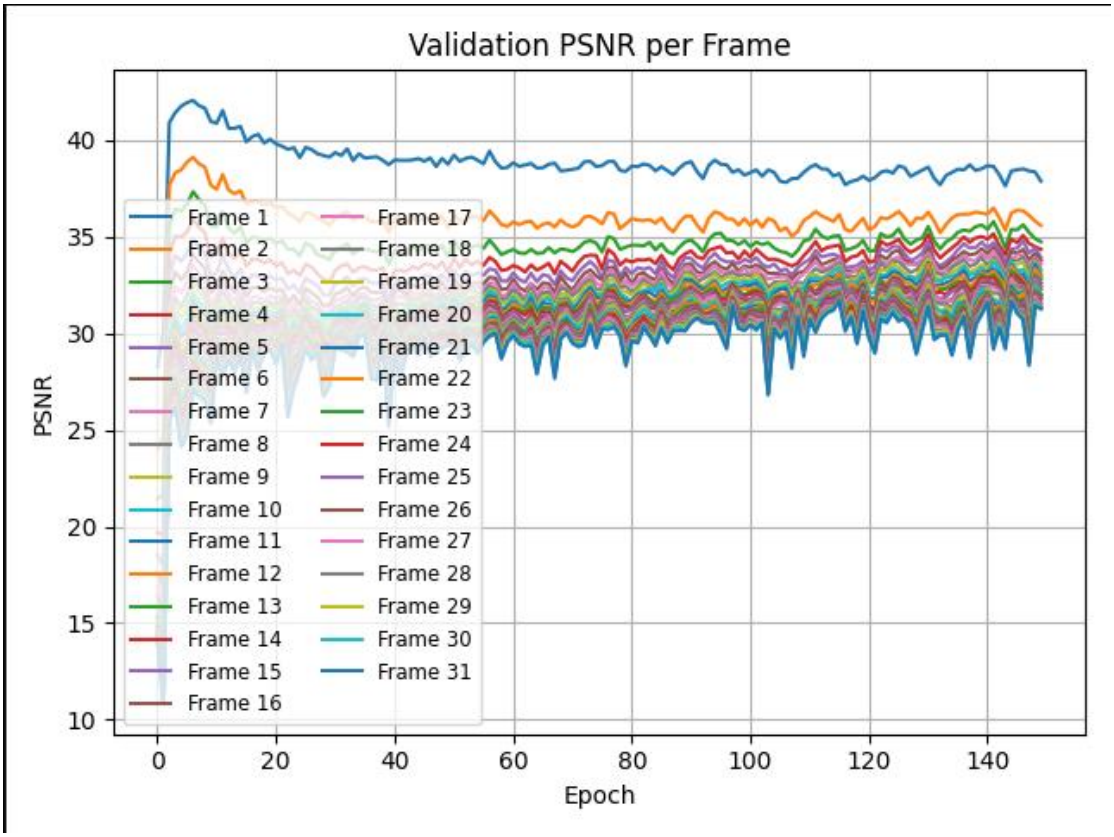
iii. Plot the PSNR-per-frame diagram in the validation dataset



上圖是在 Cyclical 的 policy 下執行的結果

我在 validation 時是測試 32 個 frame，圖片的意思是每個 epoch 每個 frame 的 PSNR

iv. Other training strategy analysis



這個是我讓 beta 用 sigmoid 方式變換的結果，可以看到結果比 cyclical 還要差，可能的原因是因為太早讓 KL 發生作用，使的 decoder 很難利用 z 裡的資訊來生成有細節的畫面。