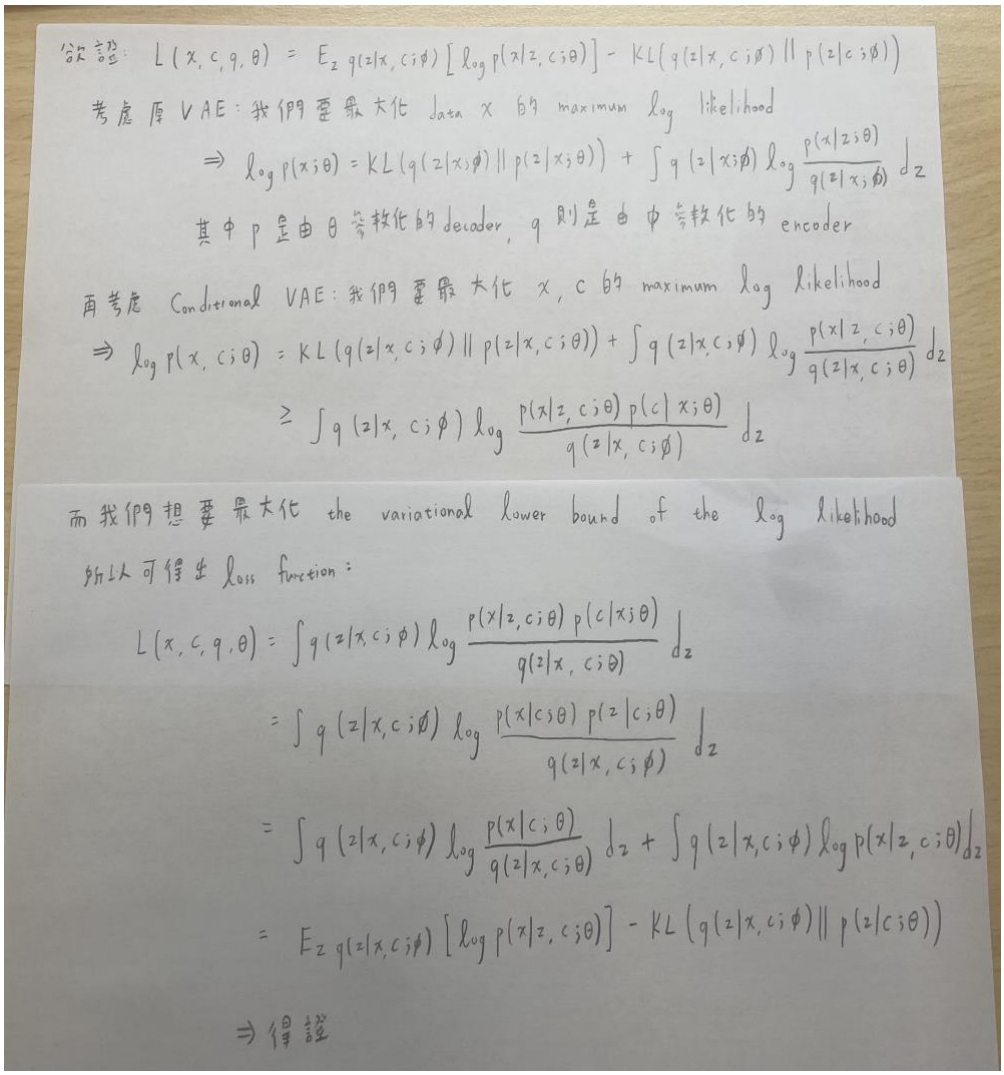
NYCU DL Lab4

Conditional VAE for Video Prediction

313552049 鄭博涵

A. Derivate conditional VAE formula



B. Introduction

In lab4, 我們需要使用一個 VAE-based model 來去實作一個 conditional video prediction. 為了提升模型的性能, 我嘗試了各種 training strategy 和 adjust hyperparameter, 以下部分將詳細介紹實現的具體過程以及對結果的分析

c. Implementation details

1. How do you write your training/testing protocol

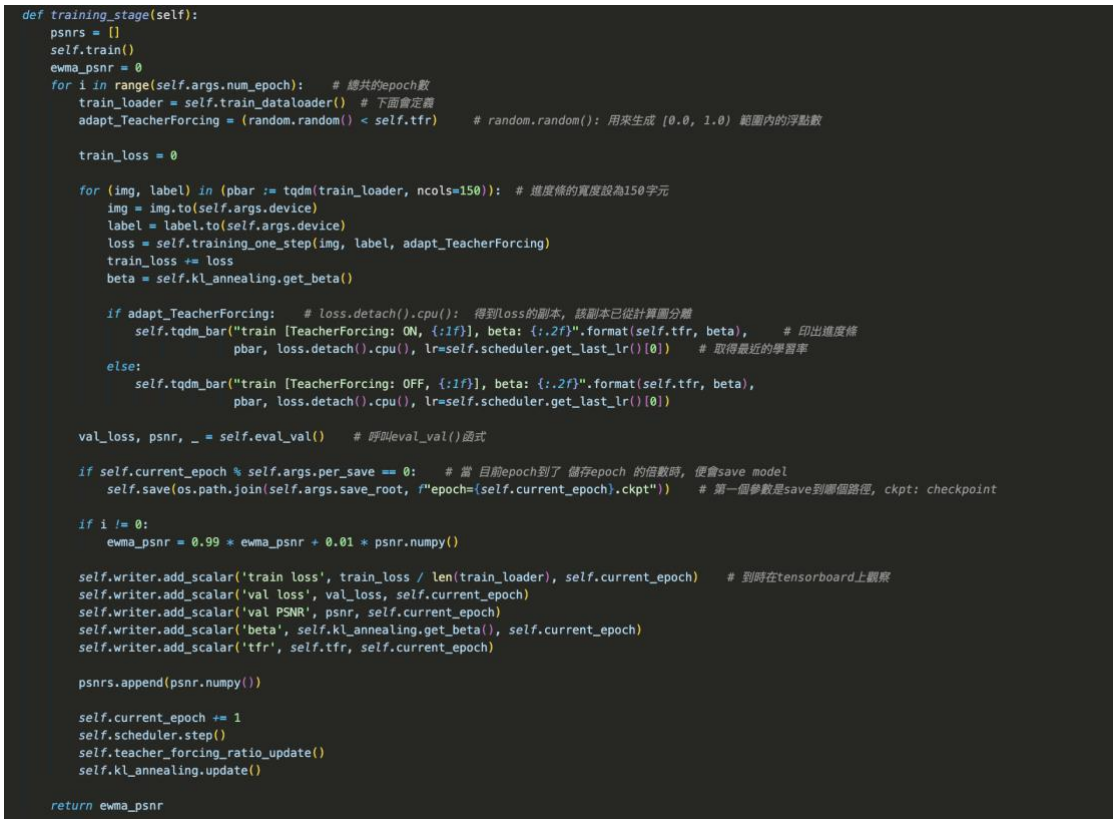
Training protocol:

VAE_model.forward:

```
def forward(self, img, img_last, label, val=False): # val: 在驗證階段時 將z隨機化 -> z允許model從潛在空間中抽取不同的點 -> 增加多樣性
    img_features = self.frame_transformation(img)   # 轉成RGB影像並存入img_features
    img_last = self.frame_transformation(img_last)
    label_features = self.label_transformation(label)

    z, mu ,logvar = self.Gaussian_Predictor(img_features, label_features)

    if val == True:
        z = torch.randn(z.size()).to(self.args.device)
    kl_loss =  kl_criterion(mu, logvar, self.batch_size) # 注意在forward這邊不需要用到 kl_annealing, 是在training_one_step中才會用到

    decoder_output = self.Decoder_Fusion(img_last, label_features, z)   # 將上一幀的特徵與當前幀的特徵合起來解碼 -> improve model's 時序關聯性
    pred = self.Generator(decoder_output)

    return pred, kl_loss
```

在這邊，首先將當前幀，上一幀輸入，標籤輸入 加到幀 Decoder 中，作為高斯預測器的輸入，然後從高斯預測器中取得 潛在變數 z，以及 mu, logvar，再將上一幀，當前標籤和潛在變數輸入至 Decoder_Fusion，最後透過 Generator 產出下一個的預測幀

Training_stage:



```python
def training_stage(self):
    psnrs = []
    self.train()
    ewma_psnr = 0
    for i in range(self.args.num_epoch):    # 總共的epoch數
        train_loader = self.train_dataloader()  # 下面會定義
        adapt_TeacherForcing = (random.random() < self.tfr)     # random.random(): 用來生成 [0.0, 1.0) 範圍內的浮點數

        train_loss = 0

        for (img, label) in (pbar := tqdm(train_loader, ncols=150)):  # 進度條的寬度設為150字元
            img = img.to(self.args.device)
            label = label.to(self.args.device)
            loss = self.training_one_step(img, label, adapt_TeacherForcing)
            train_loss += loss
            beta = self.kl_annealing.get_beta()

            if adapt_TeacherForcing:    # loss.detach().cpu():  得到loss的副本, 該副本已從計算圖分離
                self.tqdm_bar("train [TeacherForcing: ON, {:1f}], beta: {:.2f}".format(self.tfr, beta),     # 印出進度條
                              pbar, loss.detach().cpu(), lr=self.scheduler.get_last_lr()[0])    # 取得最近的學習率
            else:
                self.tqdm_bar("train [TeacherForcing: OFF, {:1f}], beta: {:.2f}".format(self.tfr, beta),
                              pbar, loss.detach().cpu(), lr=self.scheduler.get_last_lr()[0])

        val_loss, psnr, _ = self.eval_val()    # 呼叫eval_val()函式

        if self.current_epoch % self.args.per_save == 0:    # 當 目前epoch到了 儲存epoch 的倍數時, 便會save model
            self.save(os.path.join(self.args.save_root, f"epoch={self.current_epoch}.ckpt"))    # 第一個參數是save到哪個路徑, ckpt: checkpoint

        if i != 0:
            ewma_psnr = 0.99 * ewma_psnr + 0.01 * psnr.numpy()

        self.writer.add_scalar('train loss', train_loss / len(train_loader), self.current_epoch)    # 到時在tensorboard上觀察
        self.writer.add_scalar('val loss', val_loss, self.current_epoch)
        self.writer.add_scalar('val PSNR', psnr, self.current_epoch)
        self.writer.add_scalar('beta', self.kl_annealing.get_beta(), self.current_epoch)
        self.writer.add_scalar('tfr', self.tfr, self.current_epoch)

        psnrs.append(psnr.numpy())

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

    return ewma_psnr
```

在一個 epoch 裡，首先先讀 train_data，並決定是否要採用 TeacherForcing，之後設定進度條並計算 loss，根據是否採用 TeacherForcing，分別在進度條中顯示相關資訊，在每個 epoch 結束之後，計算 val_loss 跟 PSNR，並定期保存 model 的 checkpoint，將各個數據紀錄到 Tensorboard，最後將 scheduler, tfr, kl_annealing 更新

Testing protocol:

```python
""" 跟trainer那邊的forward差不多 """
def forward(self, img, label):
    # TODO
    img_features = self.frame_transformation(img)
    label_features = self.label_transformation(label)
    z, mu, logvar = self.Gaussian_Predictor(img_features, label_features)

    z = torch.randn(z.size()).to(self.args.device)  # 打亂z
    decoder_output = self.Decoder_Fusion(img_features, label_features, z)
    pred = self.Generator(decoder_output)
    return pred
```

```python
def main(args):
    set_seed(0)
    os.makedirs(args.save_root, exist_ok=True)
    model = Test_model(args).to(args.device)
    model.load_checkpoint()
    model.eval_val()
```

```python
""" 使seed固定，以便在demo時跟上傳kaggle的結果一樣  """
def set_seed(seed):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
```

值得一提的是，這邊有設 seed，以便在 demo 時跟上傳 kaggle 的結果是一樣的，

2. How do you implement reparameterization tricks

```python
def reparameterize(self, mu, logvar):   # 重參數化，mu: 均值，logvar: log variance
    std = torch.exp(logvar / 2)  # 開根號
    eplison = torch.rand_like(std)  # 生成與標準差形狀相同的隨機數據
    return mu + eplison * std
```

透過 logvar 開根號來取得標準差 std

然後生成與 std 相同形狀的標準正態分佈隨機噪聲 epsilon

最終返回 mu + epsilon * std, 這是一個以 mu 為均值、std 為標準差的高斯分佈中抽樣得到的潛在變數

通過這種方式生成的隨機變數 z 能夠保持梯度信息，使得我們可以使用標準的反向傳播來優化 VAE

3. How do you set your teacher forcing strategy

```
def teacher_forcing_ratio_update(self):  # 用在training_stage那邊
    # TODO
    if self.current_epoch >= self.tfr_sde:   # tfr_sde: The epoch that teacher forcing ratio start to decay
        self.tfr -= self.tfr_d_step     # tfr_d_step: tfr減少的比例
        self.tfr = max(0.0, self.tfr)    # 不能少於0
```

When 當前 epoch >= tfr_sde 時, 便開始減少 sfr, sfr_d_step 則是一個 epoch 要減少多少的 sfr

4. How do you set your kl annealing ratio

```python
""" beta 會在多個週期內從 start 到 stop 線性增加, 然後在每個週期結束時重置
    這樣的退火策略有助於在模型訓練過程中平衡Reconstruct Loss and KL Divergence  """
class kl_annealing():
    def __init__(self, args, current_epoch = 0):
        # TODO
        self.args = args
        self.kl_anneal_type = args.kl_anneal_type
        self.kl_anneal_cycle = args.kl_anneal_cycle
        self.kl_anneal_ratio = args.kl_anneal_ratio

        self.current_epoch = current_epoch
        n_cycle, ratio = self.kl_anneal_cycle, self.kl_anneal_ratio

        if self.kl_anneal_type == "Cyclical":
            self.beta_list = self.frange_cycle_linear(n_iter=args.num_epoch, n_cycle=n_cycle, ratio=ratio)   # self.beta_list: 用來存放beta列表
            self.beta = torch.tensor(self.beta_list[0]) # 初始化是[0]（第一個元素), 之後update時, beta會慢慢改變

        elif self.kl_anneal_type == "Monotonic":
            self.beta_list = self.frange_cycle_linear(n_iter=args.num_epoch, n_cycle=n_cycle, ratio=ratio)
            self.beta = torch.tensor(self.beta_list[0])

        elif self.kl_anneal_type == "Full_KL":
            n_cycle, ratio = None, None
            self.beta_list = None
            self.beta = torch.tensor(1.0)

        else:   # self.kl_anneal_type == "None":
            n_cycle, ratio = None, None
            self.beta_list = None
            self.beta = torch.tensor(0.0)

    def update(self):
        # TODO
        if self.kl_anneal_type == "None" or self.kl_anneal_type == "Full_KL":
            pass    # do nothing
        else:   # for mode = Cyclical or Monotonic
            self.beta = torch.tensor(self.beta_list[self.current_epoch])
        self.current_epoch += 1

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

```python
""" 參數解釋:
    n_iter: 總epoch數
    start: 每個循環的起始值
    end: 每個循環的結束值
    n_cycle: 循環次數
    ratio: 每個循環中線性增長部分所佔的比例      """
def frange_cycle_linear(self, n_iter, start=0.0, end=1.0, n_cycle=1, ratio=1):
    # TODO
    L = np.ones(n_iter)      # n_iter = num_epoch, 初始化一個長度為n_iter的數組, 值皆為1
    period = n_iter / n_cycle
    step = (end - start) / (period * ratio)

    for c in range(n_cycle):
        v, i = start, 0
        while i <= period * ratio and int(i + c * period) < n_iter:
            L[int(i + c * period)] = v
            v += step
            i += 1
    # print(L)
    return L
```

先判斷現在是屬於哪種 KL_annealing 模式，再設定其對應的 cycle, ratio，並利用取得 beta_list 的第一個元素，來獲取我們當前 epoch 對應的 beta 值

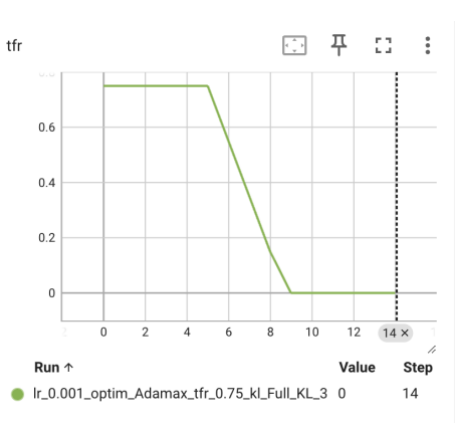
D. Analysis & Discussion

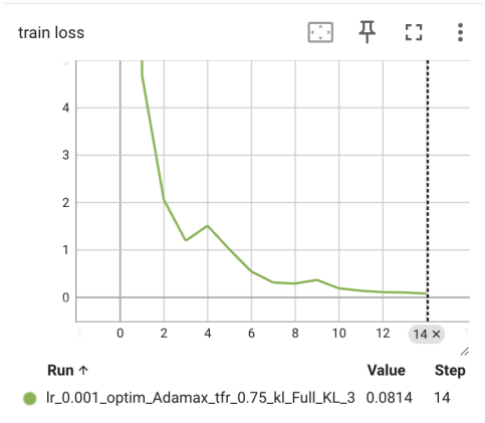
1. Plot Teacher forcing ratio

a. Analysis & compare with the loss curve

其中皆採用 Full_KL 方式(beta 皆為 1).

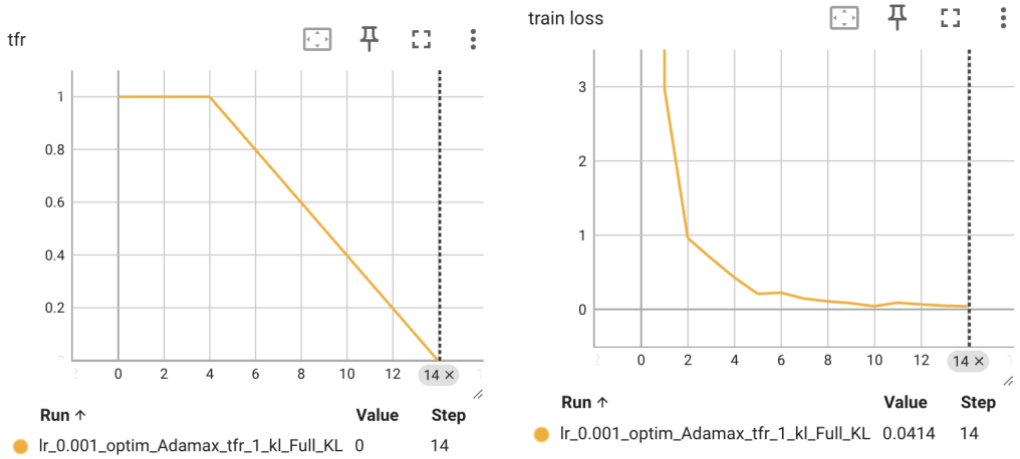
Case 1: tfr = 0.75, tfr_sde = 6, tfr_d_step = 0.2



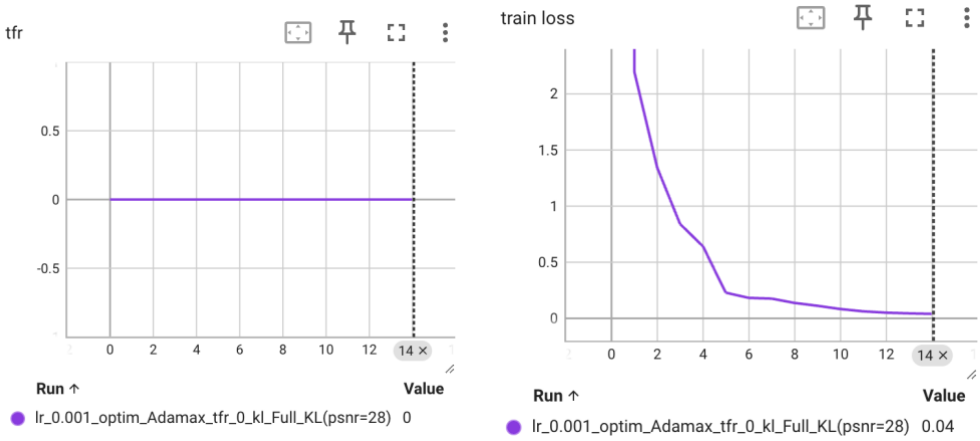


參數說明: tfr 是起始的 TeacherForcing ratio, tfr_sde 表示說第幾個 epoch 開始衰減 tfr, tfr_d_step 則是一個 epoch 裡要衰減多少 tfr

Case 2: tfr = 1, tfr_sde = 5, tfr_d_step = 0.1



Case 3: tfr = 0



Tfr = 0 的狀況下加上 Full_KL 策略是模型表現最好的一個，在 train 時的 PSNR 來到 29.多, val loss 大概在 0.8 左右,我覺得這表示 model 在完全不依賴真實數據作為下一步輸入的情況下仍然能夠很好地生成和預測

2. Plot the loss curve while training with different

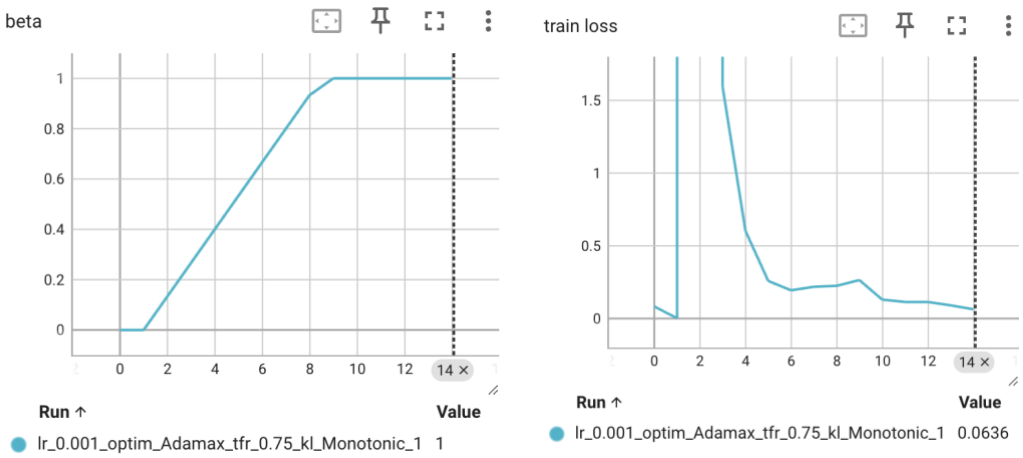
settings. Analyze the difference between them

2.1: With KL annealing (Monotonic)

以下皆使用 lr = 0.001, scheduler = StepLR, optim = Adamax,

tfr = 0.75, tfr_sde = 6, tfr_d_step = 2

左邊為 beta 值，右邊為 loss 圖

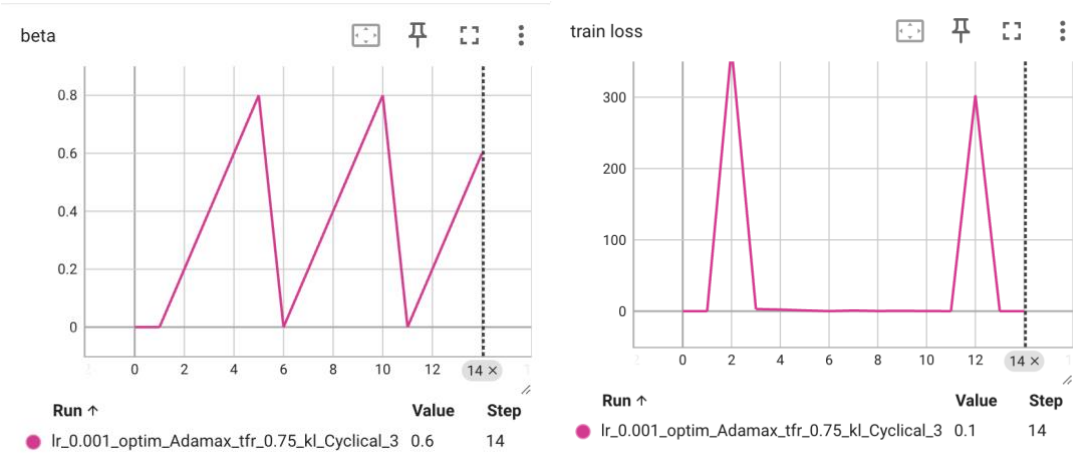


KL_annealing 參數為: Ratio : 0.5, cycle : 1

表示說在整個 training 過程中，一開始 beta 值會從 0 開始，並線性增長，當 train 到一半時 beta 值會到 end (也就是 beta=1)，而後半部的 training 階段, beta 皆保持 1

而從圖表中可看出 beta 值大概在 0.2 時會不穩定，之後 train loss 便緩緩降低

2.2: With KL annealing (Cyclical)



KL_annealing 參數為: Ratio:1, cycle: 3

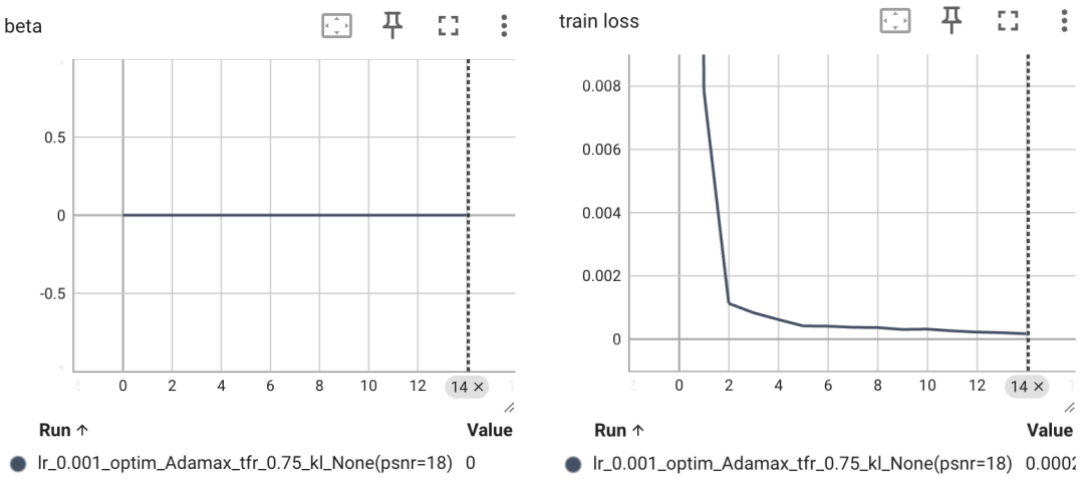
cycle = 3 表示在 training 中 beta 會經歷 3 個週期

而我 train 了 15 個 epoch，所以一個週期就是 5 個 epoch，在週期裡面線性增長，而每當一個週期結束時, beta 值會重置為零，讓模型重新適應 KL 散度的影響，表現跟 Full_KL 差不多，然後圖中可發現 beta 值在 0.2 時, train loss 會突然變很高，之後又下降到約 0.001

比 Monotonic 的 train loss 還低，效果也比 Monotonic 好一點 (PSNR and val loss)

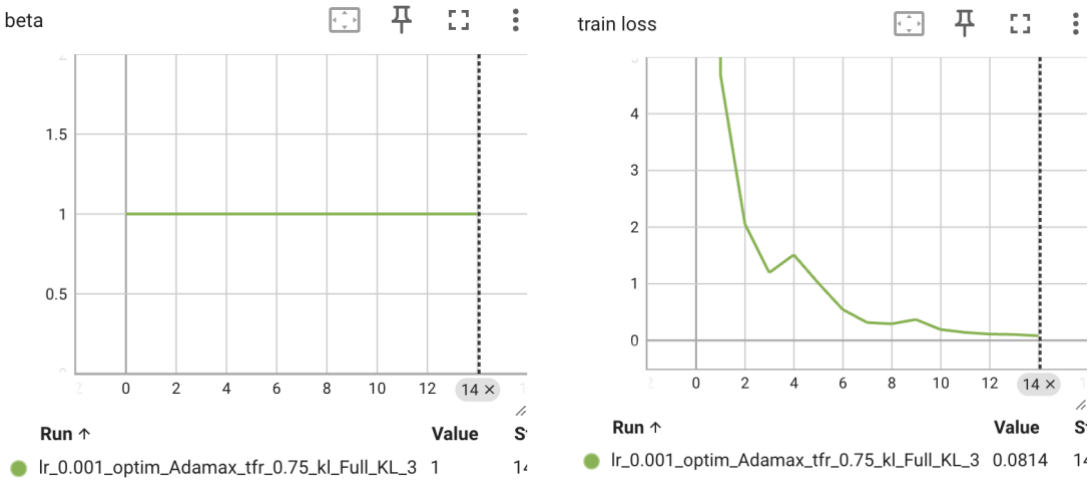
2.3: Without KL annealing

beta = 0 (None) :



在 PSNR 上表現最差，因為其完全忽略了 KL Divergence，但 train loss 較不會有忽然的高點存在的情況，應該是因為 beta 固定的關係，有比較嚴重的 overfitting

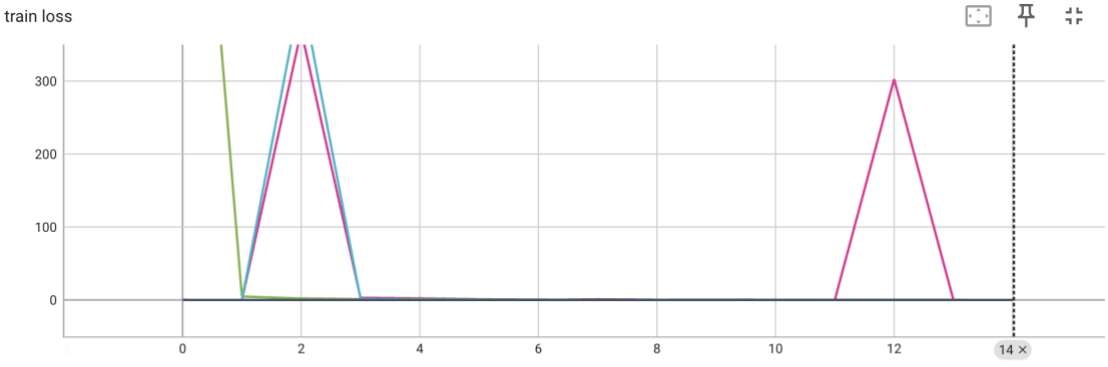
beta = 1 (Full_KL) :



表現比 Cyclical 還好一些，是四種 KL_annealing 策略裡面最高的

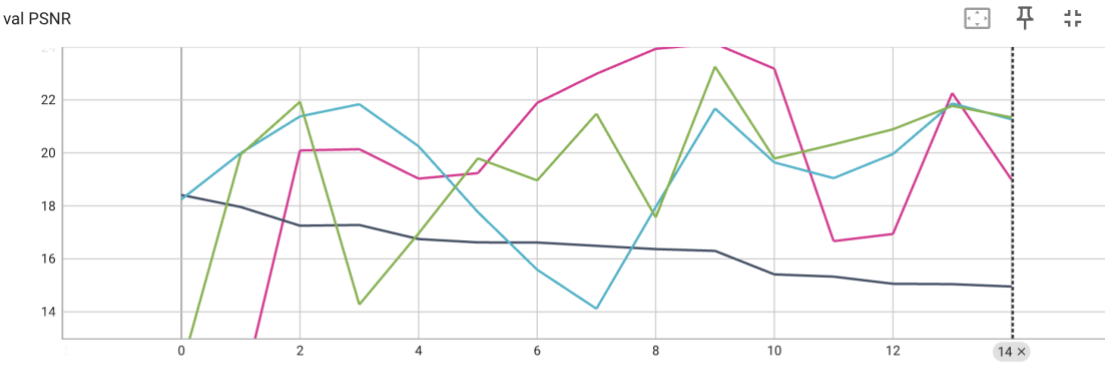
2.4: Compare 四種 KL_annealing

Train loss:



Run	Value	Step	Relative
lr_0.001_optim_Adamax_tfr_0.75_kl_Cyclical_3	0.1	14	33.77 min
lr_0.001_optim_Adamax_tfr_0.75_kl_Full_KL_3	0.0814	14	34.1 min
lr_0.001_optim_Adamax_tfr_0.75_kl_Monotonic_1	0.0636	14	33.67 min
lr_0.001_optim_Adamax_tfr_0.75_kl_None(psnr=18)	0.0002	14	34 min

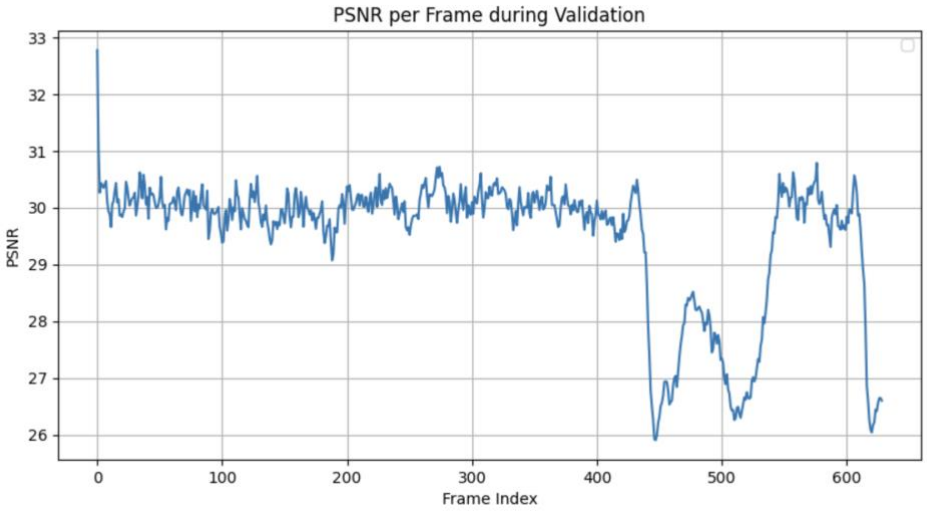
PSNR:



Run	Value	Step	Relative
lr_0.001_optim_Adamax_tfr_0.75_kl_Cyclical_3	18.9919	14	33.77 min
lr_0.001_optim_Adamax_tfr_0.75_kl_Full_KL_3	21.343	14	34.1 min
lr_0.001_optim_Adamax_tfr_0.75_kl_Monotonic_1	21.2697	14	33.67 min
lr_0.001_optim_Adamax_tfr_0.75_kl_None(psnr=18)	14.9552	14	34 min

所以由圖表得知: Full-KL 在我程式碼中 train 起來是效果最好的, Cyclical 跟 Monotonic 居於中間, 而 beta 皆為 0 時是最差的, 由此可知 model 只關注重構，忽略了潛在表示的學習, 通常效果會較差, 而 KL 散度對學習數據的潛在分佈是至關重要的，而單純依賴重構損失不足以讓模型獲得理想的結果

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



使用 Full-KL (beta 皆為 1), TeacherForcingRatio = 0, optimizer = Adamax 的方式 產出最好的 PSNR

4. Other training strategy analysis (Bonus)

由於在這次 training 中 overfitting 相當嚴重,

所以我嘗試了數據增強, dropout, 但不料都沒效果, 而且 loss 還很容易變成 nan, 後來將 optimizer 改成 Adamax 才得到比較好的效果, 之後又調整了 tfr, KL_annealing 等策略, 最後成功將 overffiting 的影響降低