312553040_郭晉維_Lab4_Report

1. Derivate conditional VAE formula

Objective function of conditional VAE :

$L_b(X,c,q,\theta) = E_{z\sim q(Z|X,c;\emptyset)} \log p(\,Z|X,c;\theta) - KL(q(Z|X,c;\emptyset)||p(Z|c))$

$\int q(z|x,c)\log p(\,x|c;\theta)dz$

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

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

$+ \int q(z|x,c)\log q(\,z|c)dz - \int q(z|x,c)\log p(\,z|x,c;\theta)dz$

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

$= L_b(x,c,q,\theta) + KL(q(z|x,c;\theta))$

The KL divergence is non-negative, $KL(q||p) \geq 0$, it follows that $\log p(\,x|c;\theta) \geq L_b(x,c,q,\theta)$, with equality iff $q(z|c) = p(z|x,c;\theta)$.

It means $L_b(x,c,q,\theta)$ is a lower bound on $\log p(\,x|c;\theta)$

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

$= \int q(z|c)\log p(\,x|z,c;\theta)dz + \int q(z|c)\log p(\,z|c)dz - \int q(z|c)\log q(\,z|c)dz$

$= E_{z\sim q(Z|X,c;\emptyset)} \log p(\,Z|X,c;\theta) - KL(q(Z|X,c;\emptyset)||p(Z|c))$

2. Introduction

影片預測(Video Prediction)是電腦視覺領域中一項重要的任務，旨在預測未來影片幀。這項任務通常涉及從現有的影片序列中學習模式和動態，以便生成接下來的影片幀。影片預測可應用於許多領域，如自動駕駛、影片編輯、影片遊戲等。影片預測任務的關鍵挑戰之一是捕捉和理解影片中的時間相關性和動態變化。為了解決這些挑戰，研究人員通常採用各種深度學習模型，如循環神經網絡(RNN)、卷積神經網絡(CNN)、生成對抗網絡(GAN)等。這些模型通過學習影片序列的空間和時間特徵來預測未來的幀。影片預測任務的目標包括生成高質量、逼真的未來幀，同時確保預測的連續性和一致性。這項任務的成功可以極大地改善影片處理和理解的性能，從而在各種應用中實現更好的效果。

Auto-encoder 是由一個 encoder 和一個 decoder 組成，encoder 會將輸入的影像壓縮降維成比較低維度的向量，通常被稱為 code 或 latent vector，能代表輸入影像的精簡化表徵，decoder 則是將 code 再解碼還原為影像，

auto-encoder 適合數據壓縮與還原，不適合生成未見過的數據。Variational auto-encoder(VAE) 是由 Diederik Kingma 和 Max Welling 於 2013 年所提出，VAE 和 auto-encoder 最大的差別是 VAE 的 encoder 不再是直接把影像壓縮成 code，而是產生兩個 vectors，並另外由 normal distribution 產生一個隨機的 vector，來組成最終的 code，這樣的作法一方面能讓 code 符合常態分佈，也將隨機性引入了 auto-encoder，VAE 適合生成未見過的數據，但不能控制生成內容。Conditional VAE 可以在生成數據時通過指定其 label 來生成想生成的數據，Conditional VAE 整體結構和 VAE 差不多，區別是在將 data 輸入 encoder 時把 data 與其 label 合併一起輸入，將編碼 Z 輸入 decoder 時把編碼內容與 label 合併一起輸入，因此在生成數據時，可以先從正態分佈採樣，然後合併想生成的 label，一起送入 decoder，就能生成和 label 類似的 output。

這次的作業讓我更清楚瞭解了深度學習在影片預測上的應用，並用 Conditional VAE 模型，在只知道影片第一幀跟 ground truth 姿態的情況下，完成了整部影片的預測，讓我對於 VAE、Conditional VAE 模型還有其 reparameterization tricks 跟 teacher forcing 還有 KL annealing 這些在深度學習中常用到的方法更加熟悉，也讓我獲益良多。

3. Implementation Details

A. Training protocol

在 training_stage( )函式中，會執行 num_epoch 次的 training iteration，每次 epoch 中，首先會隨機決定此 epoch 要不要使用 teacher forcing 以及 loss function 中 KL-term 的α值，接著透過 training_one_step( )函式計算每個 batch 的 loss 並記錄起來，最後得到此 epoch 的 epoch_loss，再透過 eval( )函式得到 VAE model 在 valid dataset 的 valid_loss 跟 valid_psnr，最後再更新 teacher forcing 的機率，供下一個 epoch 使用。我實作 training_stage( )函式的程式碼如下圖所示。

```
def training_stage(self):
    train_loss = []
    valid_loss = []
    valid_psnr = []
    for i in range(self.args.num_epoch):
        train_loader = self.train_dataloader()
        adapt_TeacherForcing = True if random.random() < self.tfr else False
        epoch_loss = 0
        beta = self.kl_annealing.get_beta(self.current_epoch)
        for (img, label) in (pbar := tqdm(train_loader, ncols=120)):
            img = img.to(self.args.device)
            label = label.to(self.args.device)
            loss = self.training_one_step(img, label, beta, adapt_TeacherForcing)
            epoch_loss += loss
            if adapt_TeacherForcing:
                self.tqdm_bar('train [TeacherForcing: ON, {:.1f}], beta: {:.2f}'.format(self.tfr, beta), pbar, loss.detach().cpu(), lr=self.sch
            else:
                self.tqdm_bar('train [TeacherForcing: OFF, {:.1f}], beta: {:.2f}'.format(self.tfr, beta), pbar, loss.detach().cpu(), lr=self.sc
        epoch_loss /= len(train_loader)
        train_loss.append(epoch_loss.item())
        loss, psnr, frame_psnr, final_img = self.eval()
        valid_loss.append(loss.item())
        valid_psnr.append(psnr)
        print('Epoch : {}, Train_loss : {:.5f}, Valid_loss : {:.5f}, Valid_PSNR : {:.2f}'.format(self.current_epoch,epoch_loss/len(train_loade
        if psnr > 23.0 :
            self.save(os.path.join(self.args.save_root, f"epoch={self.current_epoch}.ckpt"))
        self.current_epoch += 1
        self.scheduler.step()
        self.teacher_forcing_ratio_update()
    self.plot_valid_loss_curve(valid_loss)
    self.plot_psnr_curve(valid_psnr)
```

在 training_one_step ( )函式中，首先會先將 img, label 兩個 input 從(Batch, Sequence, Channel, H, W)轉成(Sequence, Batch, Channel, H, W)，接著分別進行 frame_transformation(img[i])跟 label__transformation(label[i])產生 frame 跟 pose，再透過 Gaussian_Predictor(frame, pose)產生 z, mu, logvar 的常態分佈。在生成圖片的方面，會因為是否採用 teacher forcing 導致不同的情況，在使用 teacher forcing 時，會透過 frame_transformation(img[i-1])產生 pic，即為對前一幀的 ground truth 影像進行 encode，接著再利用 Decoder_Fusion(pic, pose, z)產生 gen_pic，最後再經過 Generator(gen_pic)產生最後的生成圖片 output。而在沒有採用 teacher forcing 時，會透過 frame_transformation(pre_frame)產生 pic，即為對前一幀的 output 影像進行 encode，後續步驟則和上述採用 teacher forcing 時一樣，產生最終的 gen_pic 後再透過 mse_criterion(gen_pic, img[i])計算 MSE-term loss 以及 kl_criterion(mu, logvar, img.shape[1])計算此 batch 的 KL-term loss，在用 loss 進行 loss.backward( )跟 optimizer_step( )，最後回傳 loss。我實作 training_one_step ( )函式的程式碼如下圖所示。

```python
def training_one_step(self, img, label, beta, adapt_TeacherForcing):
    # TODO
    img = img.permute(1, 0, 2, 3, 4)
    label = label.permute(1, 0, 2, 3, 4)
    loss = 0
    if adapt_TeacherForcing :
        for i in range(1, img.shape[0]) :
            frame = self.frame_transformation(img[i])
            pose = self.label_transformation(label[i])
            z, mu, logvar = self.Gaussian_Predictor(frame, pose)
            pic = self.frame_transformation(img[i-1])
            gen_pic = self.Decoder_Fusion(pic, pose, z)
            gen_pic = self.Generator(gen_pic)
            loss += beta * kl_criterion(mu, logvar, img.shape[1])
            loss += self.mse_criterion(gen_pic, img[i])
        loss.backward()
        self.optimizer_step()
        loss /= (img.shape[0] - 1)
    else :
        pre_frame = img[0]
        for i in range(1, img.shape[0]) :
            frame = self.frame_transformation(img[i])
            pose = self.label_transformation(label[i])
            z, mu, logvar = self.Gaussian_Predictor(frame, pose)
            pic = self.frame_transformation(pre_frame)
            gen_pic = self.Decoder_Fusion(pic, pose, z)
            gen_pic = self.Generator(gen_pic)
            pre_frame = gen_pic
            loss += beta * kl_criterion(mu, logvar, img.shape[1])
            loss += self.mse_criterion(gen_pic, img[i])
        loss.backward()
        self.optimizer_step()
        loss /= (img.shape[0] - 1)
    return loss
```

在 eval( )函式中，主要透過 valid__one_step ( )函式得到 VAE model 在 valid dataset 的 valid_loss 跟 valid_psnr 以及預測影片中的每一幀(final_img)及其 psnr(frame_psnr)。在 eval_one_step( )函式中，在 valid_one_step ( )函式中，首先會先將 img, label 兩個 input 從(Batch, Sequence, Channel, H, W)轉成(Sequence, Batch, Channel, H, W)，接著分別進行 frame_transformation(img[i])跟 label__transformation(label[i])產生 frame 跟 pose，再透過 Gaussian_Predictor(frame, pose)產生 z_temp, mu, logvar 的常態分佈，但之後並

不會直接使用 z_temp，而是會另外產生一個 size 跟 z_temp 一樣的$N(0, I)$常態分佈 z。在生成影片幀時，會透過 frame_transformation(pre_frame)產生 pic，即為對前一幀的 output 影像進行 encode，接著再利用 Decoder_Fusion(pic, pose, z)產生 gen_pic，最後再經過 Generator(gen_pic)產生最後的生成圖片 output。用 mse_criterion(gen_pic, img[i])計算 MSE-term loss 以及 Generate_PSNR(gen_pic, img[i])計算 psnr，最後回傳 loss, psnr, frame_psnr, final_img。我實作 eval( )函式跟 training_one_step ( )函式的程式碼如下兩張圖所示。

```python
@torch.no_grad()
def eval(self):
    val_loader = self.val_dataloader()
    for (img, label) in (pbar := tqdm(val_loader, ncols=120)):
        img = img.to(self.args.device)
        label = label.to(self.args.device)
        loss, psnr, frame_psnr, final_img = self.val_one_step(img, label)
        self.tqdm_bar('val', pbar, loss.detach().cpu(), lr=self.scheduler.get_last_lr()[0])

    return loss, psnr, frame_psnr, final_img
def val_one_step(self, img, label):
    # TODO
    img = img.permute(1, 0, 2, 3, 4)
    label = label.permute(1, 0, 2, 3, 4)
    loss = 0
    psnr = 0
    frame_psnr = []
    final_img = []
    pre_frame = img[0]
    for i in range(1, 630) :
        frame = self.frame_transformation(img[i])
        pose = self.label_transformation(label[i])
        z_temp, mu, logvar = self.Gaussian_Predictor(frame, pose)
        z = torch.randn_like(z_temp)
        feature = self.frame_transformation(pre_frame)
        gen_pic = self.Decoder_Fusion(feature, pose, z)
        gen_pic = self.Generator(gen_pic)
        pre_frame = gen_pic
        loss += self.mse_criterion(gen_pic, img[i])
        x = Generate_PSNR(gen_pic, img[i]).item()
        psnr += x
        frame_psnr.append(x)
        if self.args.test:
            final_img.append(gen_pic[0])
    loss /= (img.shape[0] - 1)
    psnr /= (img.shape[0] - 1)
    return loss, psnr, frame_psnr, final_img
```

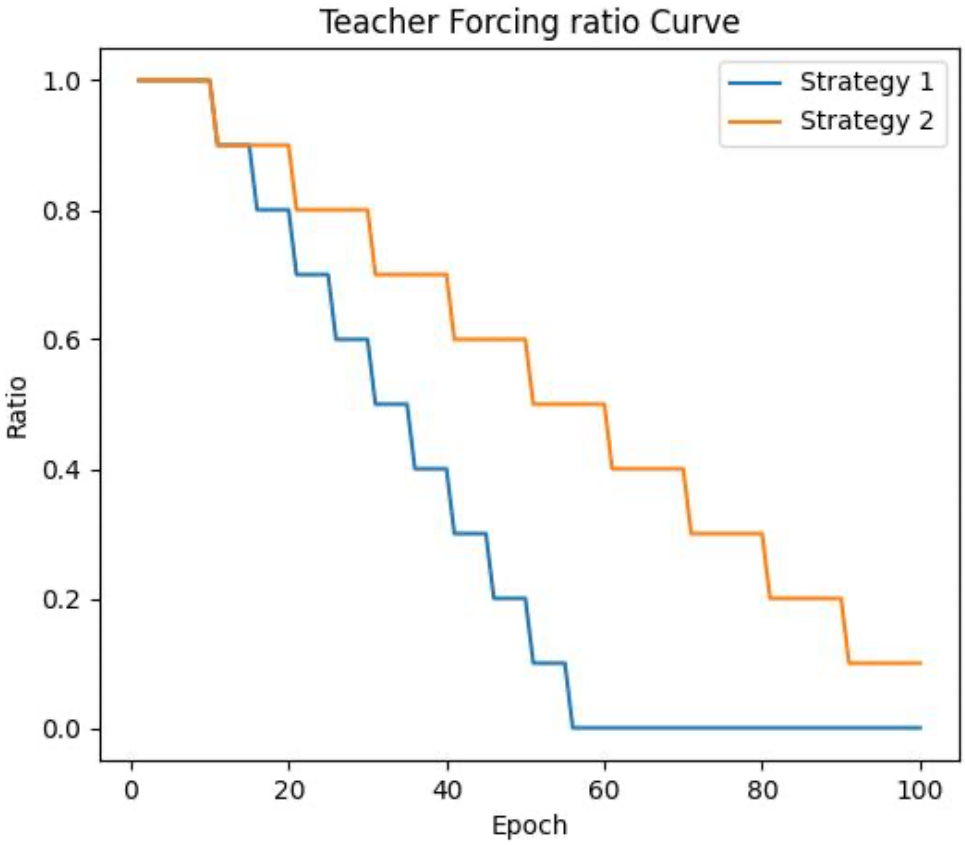
B. Reparameterization tricks

在 reparameterization tricks 部分，因為 logvar = $\log_e \sigma^2$，因此可以推導出 $\sigma = e^{0.5*\text{logvar}}$，接著產生 size 跟 σ 相同的隨機雜訊 ϵ ，最後回傳 $\mu + \epsilon * \sigma$，也就是一個可微的隨機變量 z。我實作 reparameterize ( )函式的程式碼如下圖所示。

```python
def reparameterize(self, mu, logvar):
    # TODO
    # https://blog.csdn.net/yangweipeng708/article
    std = torch.exp(0.5 * logvar)
    epison = torch.randn_like(std)
    return mu + epison * std
```

C. Teacher forcing strategy

在 teacher forcing strategy 的部分，我嘗試了兩種不同的策略，如下圖所示。第一種策略是從第 11 個 epoch 開始，每經過 5 個 epoch 就把 teacher forcing ratio 調降 0.1，直到 teacher forcing ratio 為 0。第二種策略則是從第 11 個 epoch 開始，每經過 10 個 epoch 才將 teacher forcing ratio 調降 0.1，因此直到最後 teacher forcing ratio 還是會有 0.1。這兩種策略的差別在於 strategy 1 下降速度較快，因此 training 的後半部分都是沒有採用 teacher forcing 的，而 strategy 2 下降速度較慢，training 的後半部分還是會有機率採取 teacher forcing。而從實驗結果看來，strategy 1 會較 strategy 2 擁有更好的 training 效果，在接下來的 **4.Analysis & Discussion** 部分會再仔細的分析。我實作 teacher_forcing_update ( )函式的程式碼如下圖所示。

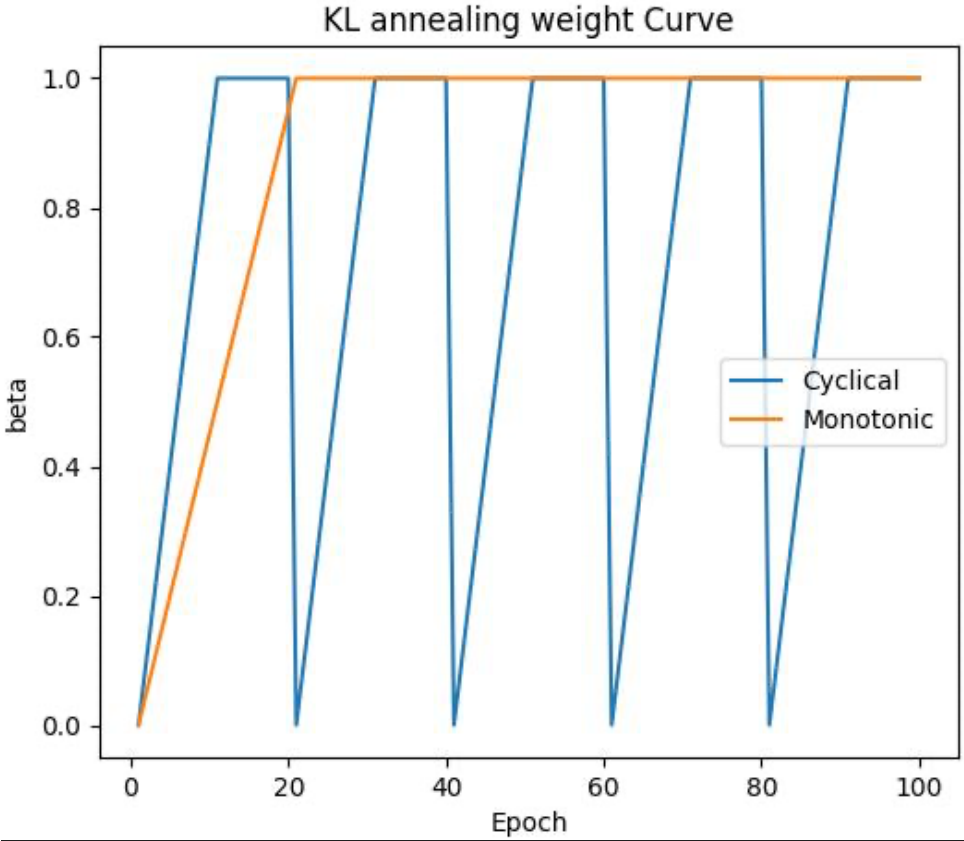



```python
def teacher_forcing_ratio_update(self):
    # TODO
    min = 0.0
    n = 5
    r = 10 % n
    if self.current_epoch >= self.tfr_sde and self.tfr > min and self.current_epoch % n == r:
        self.tfr -= self.tfr_d_step
    self.tfr = round(self.tfr, 2)
```

D. KL annealing ratio

在 KL annealing 的部分，分成了 Cyclical 跟 Monotonic 兩種方法。Cyclical 部分我設定 20 epoch 為一個 cycle，每個 cycle 中前半部分會讓 beta 線性的從 0 上升到 1，後半部分則是讓 beta 維持在 1。Monotonic 部分我則是使用前 20 個

epoch 讓 beta 線性的從 0 上升到 1，之後就讓 beta 一直維持在 1。而從實驗結果看來，Cyclical 會較 Monotonic 擁有更好的 training 效果，在接下來的 **4.Analysis & Discussion** 部分會再仔細的分析。我實作 class kl_annealing ( )的程式碼如下圖所示。

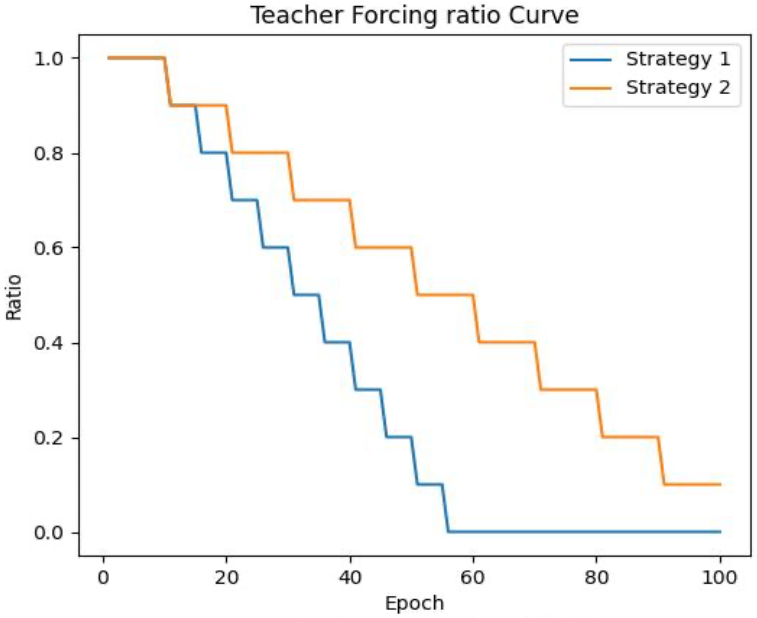


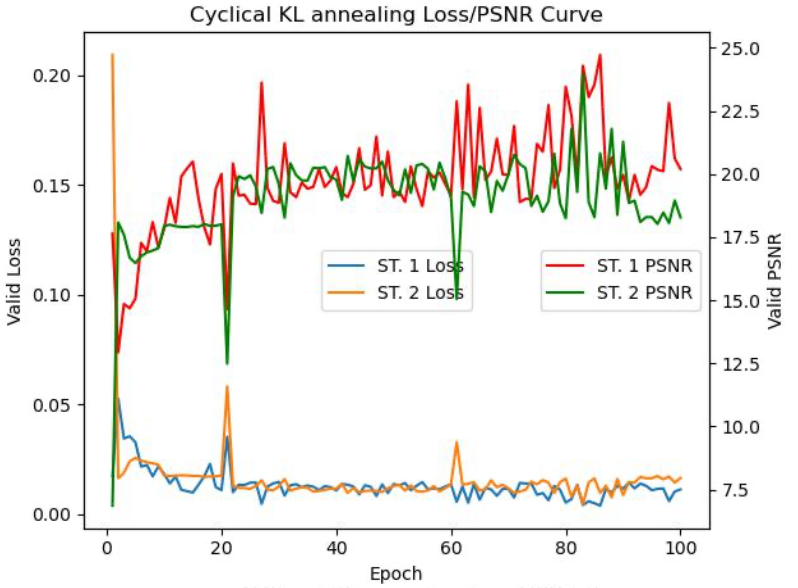

```python
class kl_annealing():
    def __init__(self, args, current_epoch=0):
        # TODO
        super(kl_annealing, self).__init__()
        self.args = args
        self.cur_epoch = current_epoch
    def get_beta(self, epoch):
        # TODO
        if self.args.kl_anneal_type == 'No':
            return 1
        else:
            beta = self.frange_cycle_linear(self.args.num_epoch, 0.0, 1.0,  self.args.kl_anneal_cycle, self.args.kl_anneal_ratio)
            return beta[epoch]
    def frange_cycle_linear(self, n_iter, start=0.0, stop=1.0,  n_cycle=1, ratio=0.5):
        # TODO
        # https://github.com/haofuml/cyclical_annealing/blob/master/plot/plot_schedules.ipynb
        if self.args.kl_anneal_type == 'Cyclical':
            pass
        elif self.args.kl_anneal_type == 'Monotonic':
            ratio = 0.2
            n_cycle = 1
        L = np.ones(n_iter) * stop
        period = n_iter/n_cycle
        if ratio == 1 :
            ratio = (period - 1)/float(period)
        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
```

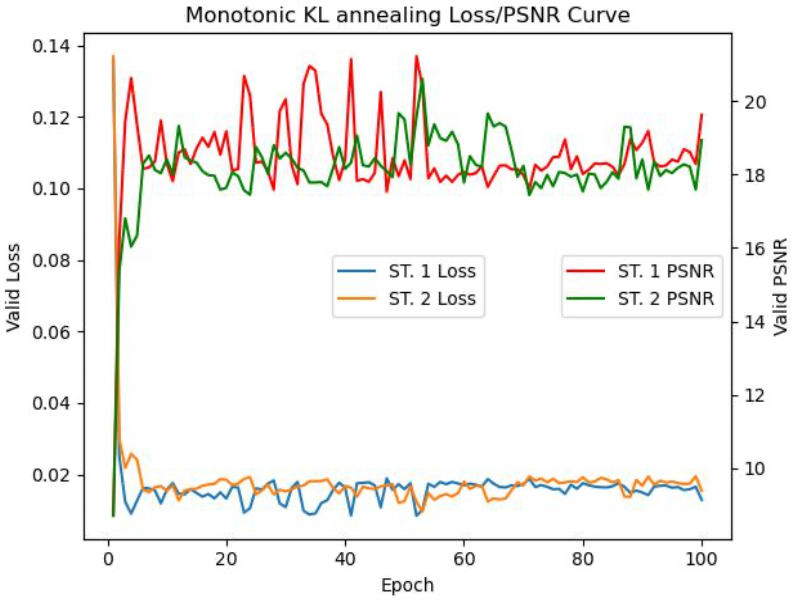
4. Analysis & Discussion

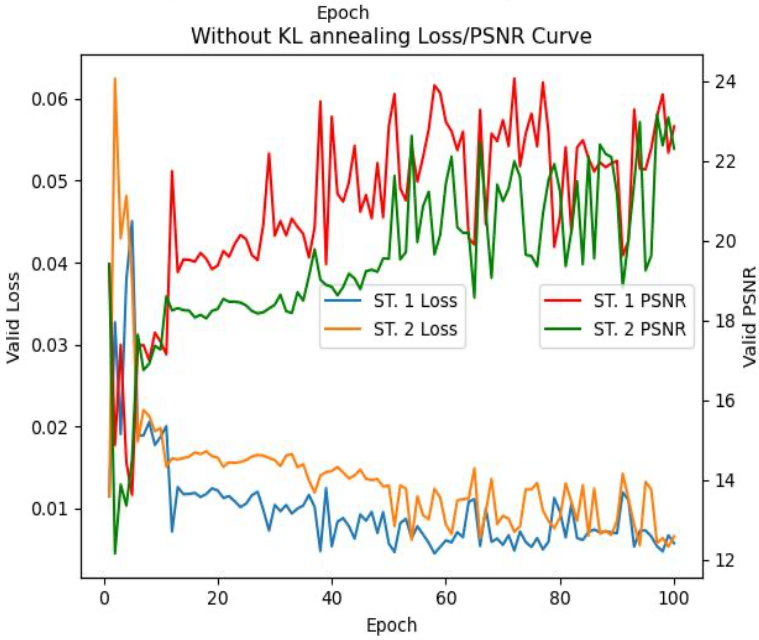
A. Different Teacher forcing ratio

在不同 teacher forcing strategy 的部分，我比較了 strategy 1 跟 strategy 2 在 Cyclical、Monotonic、Without 三種 KL annealing 方法所造成的影響，可以看到不管使用哪種 KL annealing 方法，strategy 1 不但最後的 valid_loss 跟 valid_PSNR 都會比 strategy 2 來得好，而且若是觀察 training 過程中得到的 valid_PSNR 最佳結果，同樣是採用 strategy 1 時比較高。我認為會出現這樣的結果是因為 strategy 1 在 training 後半部已將 teacher forcing ratio 降為 0，因此在面對不能使用 ground truth 的 valid datasets 時能有較好的適應性，反觀 strategy 2 可能會在 training 時太過依賴 teacher forcing 所帶來的效果，導致實際在 valid datasets 的表現上不如預期，因此我認為在 training 時有足夠多的 epoch 讓 VAE model 學習完全不採用 teacher forcing 的方法，對最後的結果是有幫助的。Different teacher forcing ratio 的實驗結果如下面四張圖所示。



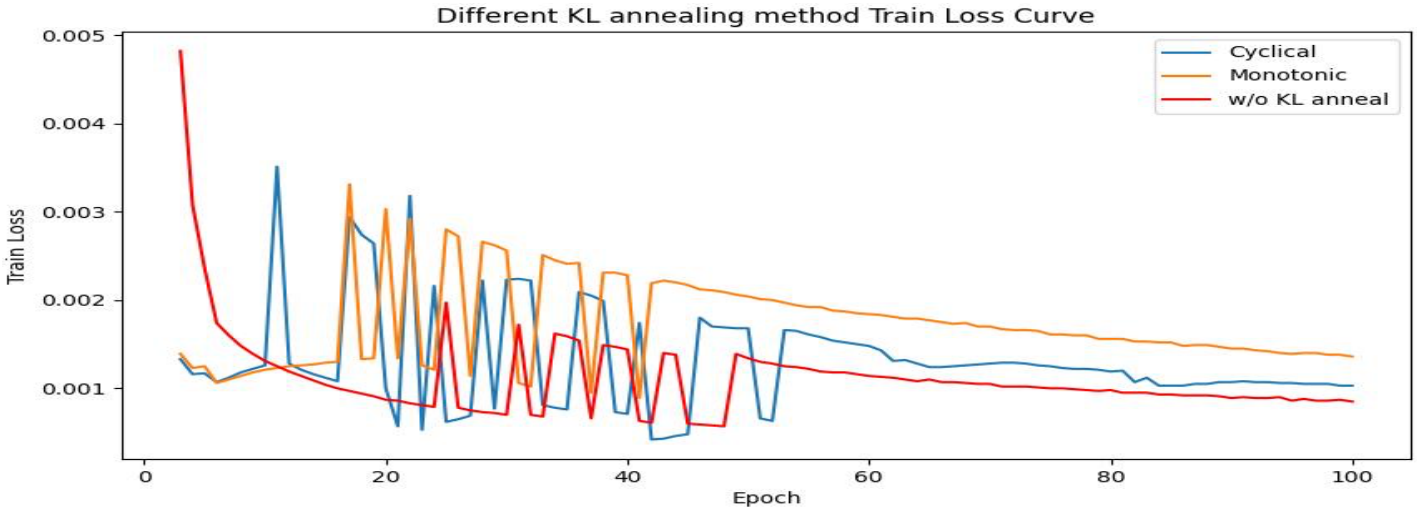


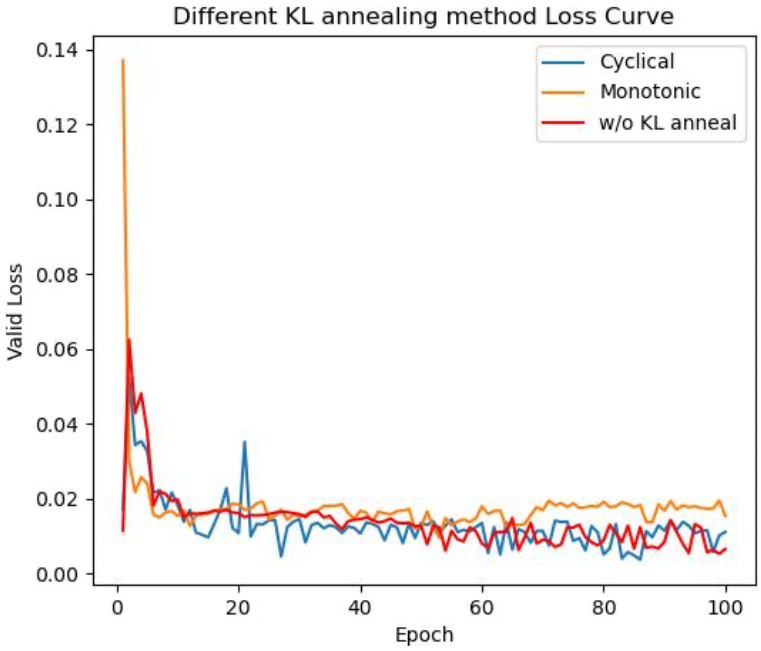


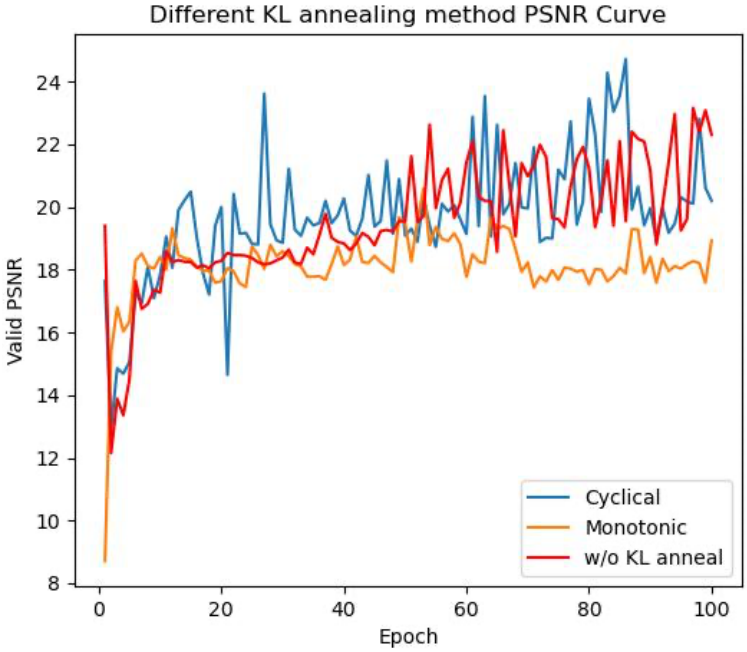


B. Different KL annealing method

在不同 KL annealing 的部分，我在三種方法都同樣使用了上面提到的 teacher forcing strategy 來進行實驗，從下面左邊的 Loss Curve 圖可以看出來 Cyclical 跟 Without KL annealing 兩種方法都比 Monotonic 好上不少，下面右邊的 PSNR Curve 圖也可以看出 Cyclical 跟 Without KL annealing 的 PSNR 結果比起 Monotonic 有明顯的提升。我認為是因為 Cyclical 相較於 Monotonic 有週期性的讓 beta 發生變化，進而有效解決 KL 消失趨近於 0 的問題，使得 model 學習到的特徵可以有效的被反映在數據上。另外雖然 Cyclical 在 valid loss 跟 valid PSNR 整體的表現趨勢並沒有明顯優於 Without KL anneal，甚至在 train loss 部分表現肉眼可見的比較差，但若是單純比較 training 過程中的最佳結果，會發現 Cyclical 會比 Without KL anneal 來得出色。我認為是因為週期變化的 beta 讓 model 在進行反向傳播時增加了一些隨機性，使得 model 有時候可以探索到更好的結果，也能因此得到表現較佳的 model。Different KL annealing method 的實驗結果如下面三張圖所示。







綜合上述兩個部分提到的 Different teacher forcing ratio 以及 Different KL annealing method，我將每種設定中的最佳結果作成了下列兩個表格，可以看到各種設定在 training 過程中的最佳 model 不論是在 valid datasets 還是 test dataset 上的表現都是 **Teacher Forcing Strategy 1 & Cyclical KL annealing** 這個組合有最好的結果，若單看 teacher forcing ratio 或是 KL annealing method 的話也可以看出跟上述兩個部分的結論，也就是 **Teacher Forcing Strategy 1 > Teacher Forcing Strategy 2** 還有 **Cyclical KL annealing > w/o KL annealing > Monotonic KL annealing**。

KL annealing / Teacher Forcing		Cyclical	Monotonic	w/o KL anneal
Strategy 1	Valid Loss	**0.00379**	0.00845	0.00452
	Valid PSNR	**24.716**	21.216	24.060
Strategy 2	Valid Loss	0.00447	0.00967	0.00534
	Valid PSNR	23.972	20.600	23.159

Best Valid Loss / PSNR in different setting

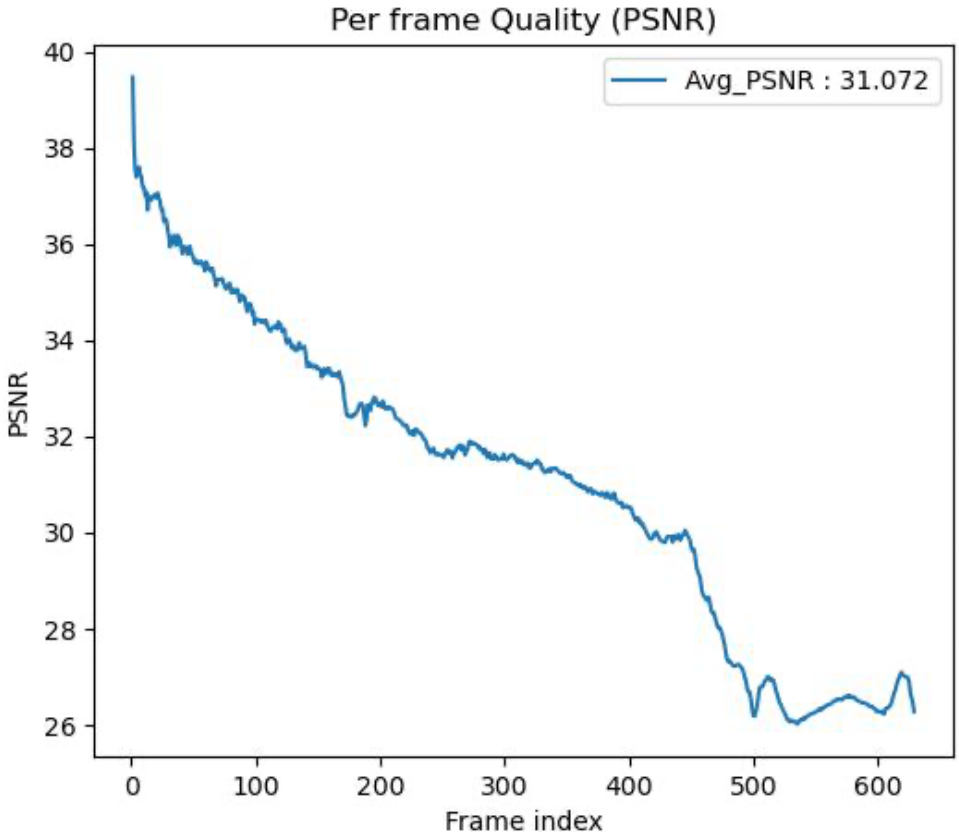
KL annealing / Teacher Forcing		Cyclical	Monotonic	w/o KL anneal
Strategy 1	Public Score	**22.979**	19.775	21.894
	Private Score	**24.805**	21.133	22.870
Strategy 2	Public Score	21.771	18.773	21.418
	Private Score	24.460	20.135	22.787

Best Test Public / Private score in different setting

C. PSNR-per frame diagram in validation dataset

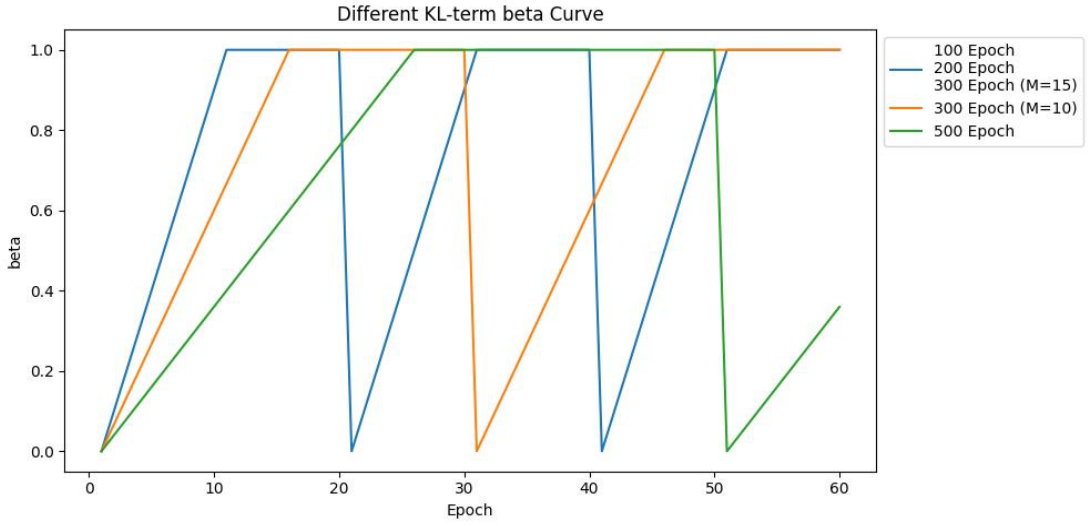
我最後得到表現最好的 model 是使用 teacher forcing strategy 1、Monotonic KL annealing、500 training epoch(會在 **D. Other training strategy analysis** 部分仔細的分析)，這個 model 在 kaggle 上的 public score 跟 private score 如下圖一所示，而這個 model 在 valid dataset 中每個 frame 的 PSNR 變化值如下頁圖二所示。

```
(yutt) ibm@ibm:/media/ibm/F/mark/DLP/lab4/Lab4_template$ python demo.py
Public score:  26.530037102531846
Private score:  30.71713202297305
```

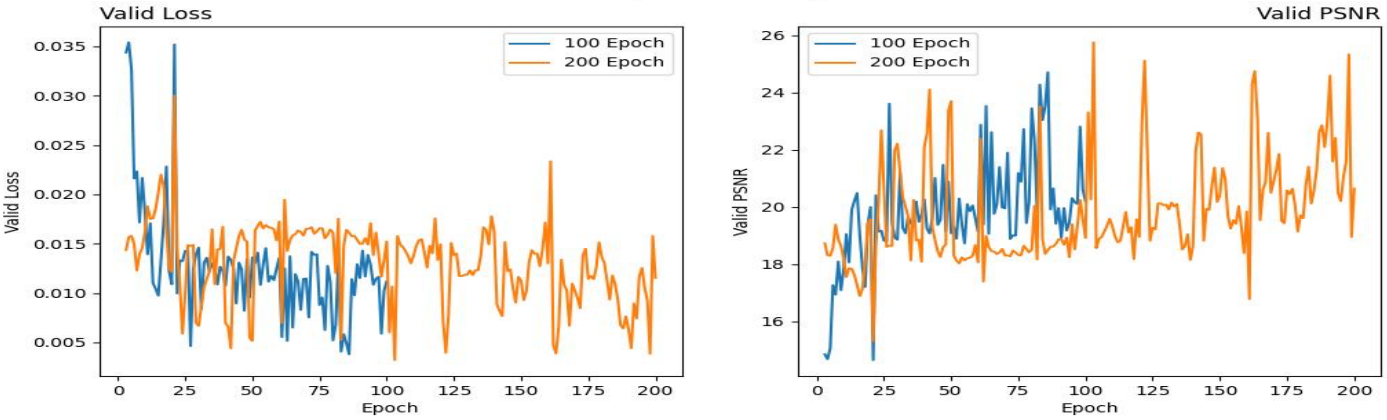


D. Other training strategy analysis

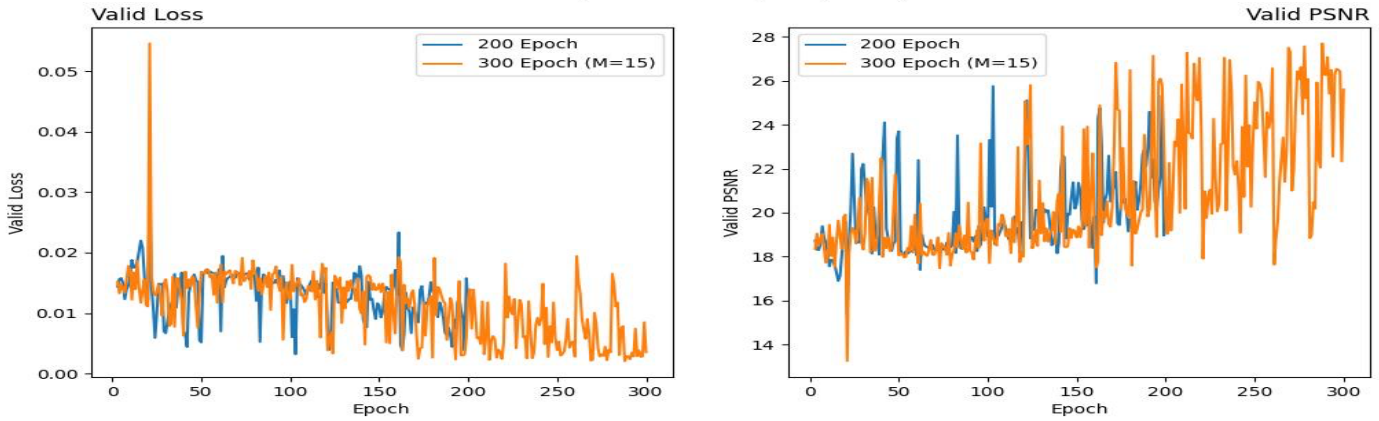
為了得到更好的結果，我一開始試圖將 training epoch 從 100 調整到 200，結果發現 training 過程會得到表現更好的 model，於是我再度嘗試將 training epoch 上升到 300，一樣能得到更好的 model。因為想避免只是一昧的增加 training epoch，我把 300 Epoch 的 M 由 15 調降至 10，意即原本使 beta 歸零的 epoch 數量從 20 上升至 30，而結果顯示較長的 beta 週期可以讓 model 的結果表現提升不少，最後我將 training epoch 上升到 500，並把 M 維持在 10(使 beta 歸零的 epoch 為 50)，也得到了更理想的結果。下圖是各種實驗中的 beta 趨勢圖，下頁的四張圖則是將不同實驗依序兩兩比較的圖，可以看到 training 過程中最佳的 Valid Loss 跟 Valid PSNR 都有明顯的進步。



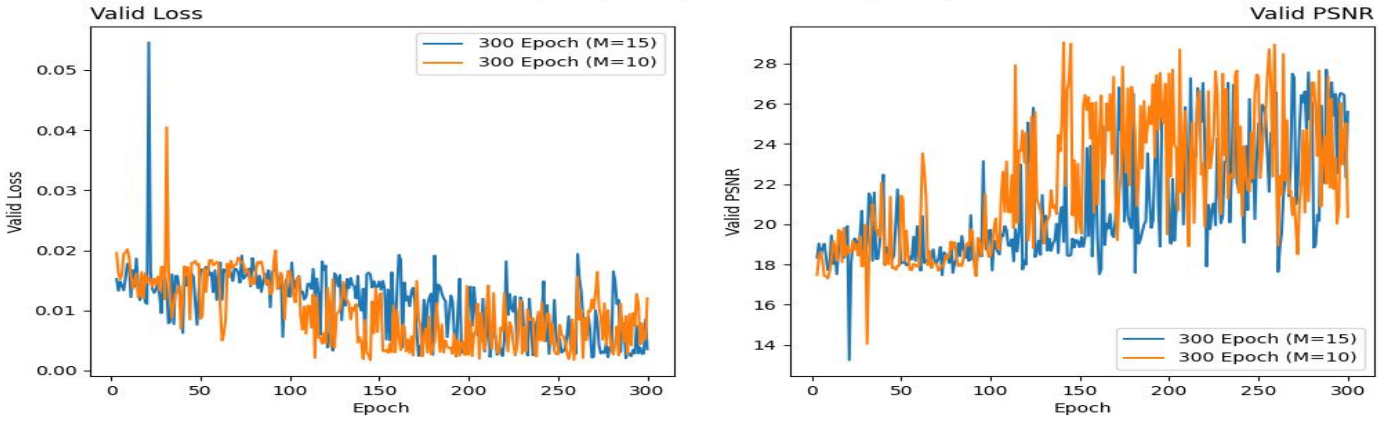
100 Epoch v.s. 200 Epoch



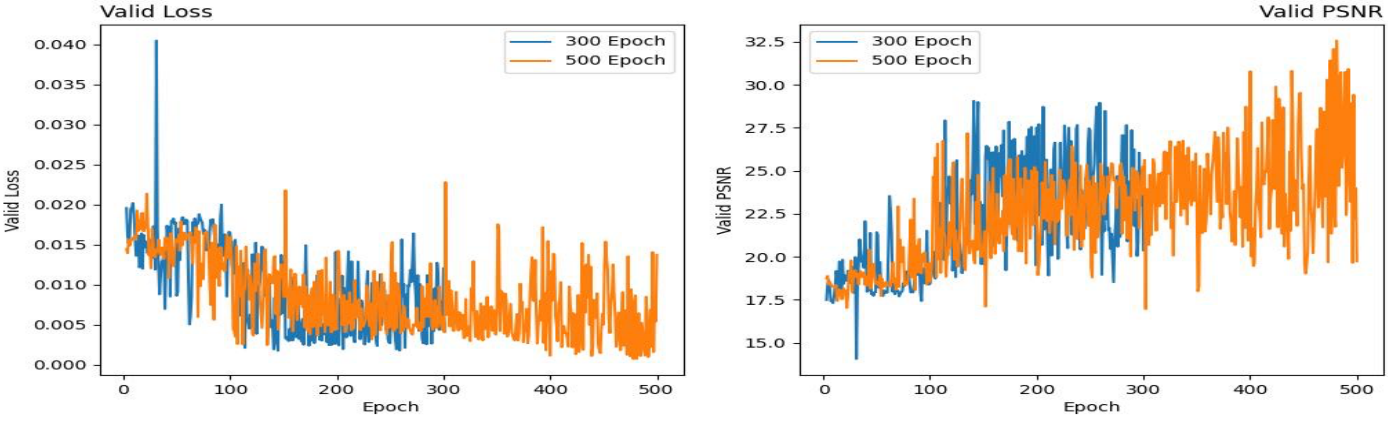
200 Epoch v.s. 300 Epoch (M=15)



300 Epoch (M=15) v.s. 300 Epoch (M=10)



300 Epoch v.s. 500 Epoch



我將上述五種實驗的最佳結果作成了下列表格，可以看到隨著 training epoch 數量的增加，表格中不論是 Valid Loss, Valid PSNR, Test Public Score, Test Private Score 幾乎都呈現出了由左至右越來越好的結果，足以推斷出當 training epoch 增加，model 都能在 Cyclical KL annealing 的隨機性下找到更好的結果，就像 300 Epoch 時的最佳結果並不是出現在 1~200 Epoch 間，500 Epoch 時的最佳結果也不是出現在 1~300 Epoch 間，因此我認為增加 training epoch 是一個有用的 training strategy，並不只是運氣好而已，我認為若是將 training epoch 從 500 繼續往上增加的話，得到擁有更好表現的 model 的機率非常的高。

Training Epoch	100 Epoch (M=5)	200 Epoch (M=10)	300 Epoch (M=15)	300 Epoch (M=10)	500 Epoch (M=10)
Best Valid Loss	0.00379	0.00319	0.00215	0.00173	**0.00070**
Best Valid PSNR	24.716	25.760	27.703	28.9521	**31.416**
Best PSNR Epoch	***85***	***102***	***287***	***258***	***474***
Test Public Score	22.979	24.012	25.135	24.465	**26.530**
Test Private Score	24.805	24.916	26.191	26.522	**30.717**