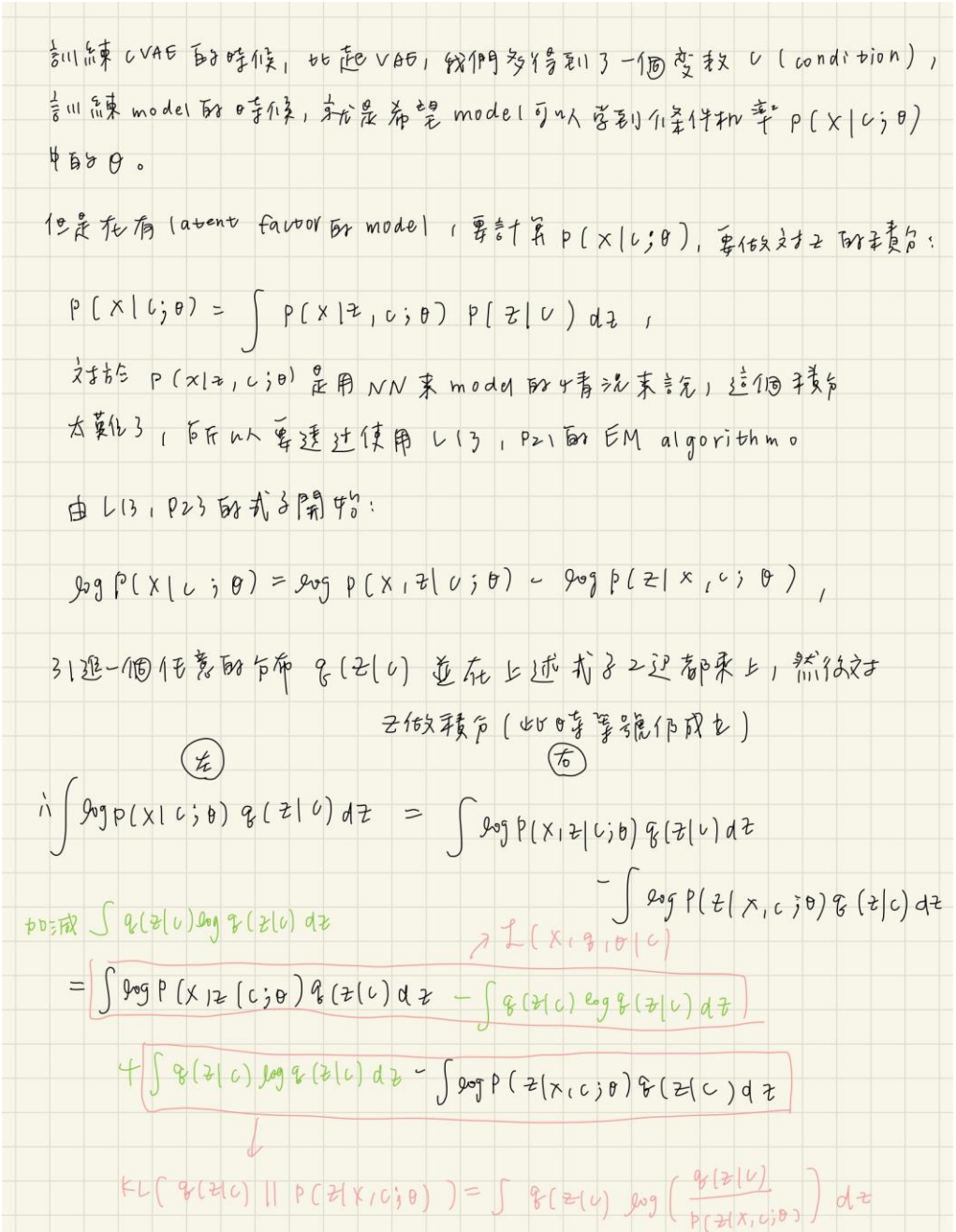
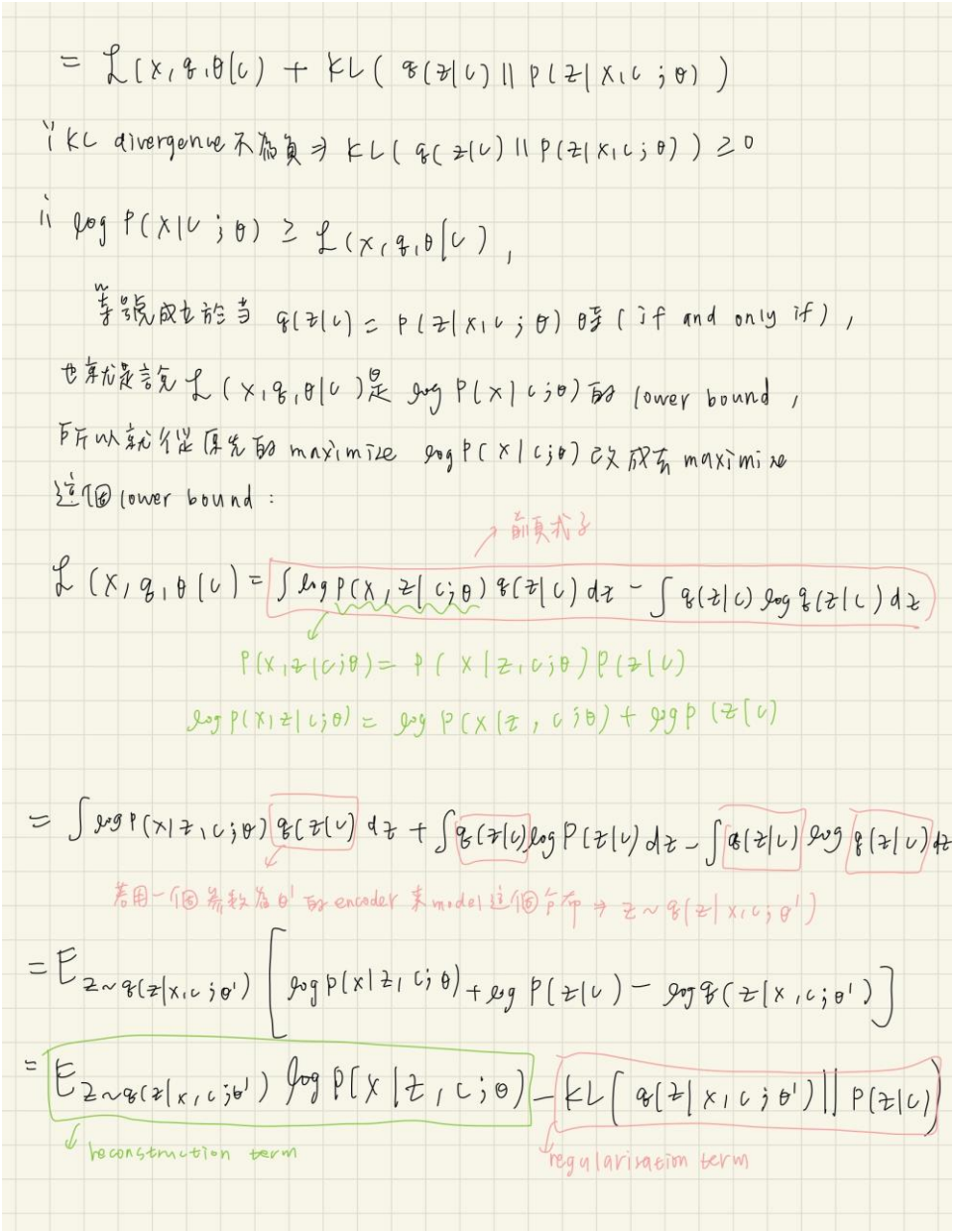
• Introduction

VAE(Variational AutoEncode)是一種常用的 generative model。在這次的 lab 中，我們實作了一種 VAE 的變形—CVAE(conditional VAE)，來預測影片後來的樣子。在這次的訓練中，我們使用了 BAIR Robot Pushing dataset(有很多 frame 大小是 64*64 的機械手臂在桌子上推東西的時間序列影像，裡面也提供了機械手臂在每個 frame 採取的 action 還有機械手臂的 end effector 位置的資訊)。model 的訓練目標就是希望可以透過得到機械手臂推東西過程的前兩個 frame，來預測出它未來 10 個 frame 的樣子。而用來評估預測結果好壞的標準就是使用 PSNR，是一個用來計算影像失真的量化標準。

• Derivation of CVAE (Please use the same notation in Fig.1a)





- Implementation details
  - Describe how you implement your model (encoder, decoder, reparameterization trick, dataloader, etc.)

    下圖是 implement encoder 的方式。Encoder 本身是由多層的 convolution 組成的 vgg layer 所構成的。所以在影像送進 encoder 後，encoder 就會把影像 encode 成 latent code，同時在 encode 的過程中，也會 output 出 4 種不同大小的 feature map。

```python
class vgg_layer(nn.Module):
    def __init__(self, nin, nout):
        super(vgg_layer, self).__init__()

        self.main = nn.Sequential(
                nn.Conv2d(nin, nout, 3, 1, 1),
                nn.BatchNorm2d(nout),
                nn.LeakyReLU(0.2, inplace=True)
                )

    def forward(self, input):
        return self.main(input)
```

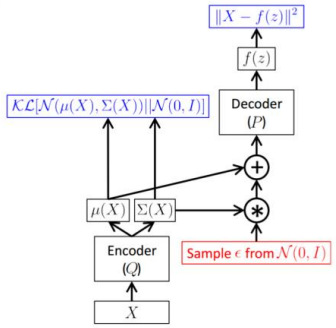
```python
class vgg_encoder(nn.Module):
    def __init__(self, dim):
        super(vgg_encoder, self).__init__()
        self.dim = dim
        # 64 x 64
        self.c1 = nn.Sequential(
                vgg_layer(3, 64),
                vgg_layer(64, 64),
                )
        # 32 x 32
        self.c2 = nn.Sequential(
                vgg_layer(64, 128),
                vgg_layer(128, 128),
                )
        # 16 x 16
        self.c3 = nn.Sequential(
                vgg_layer(128, 256),
                vgg_layer(256, 256),
                vgg_layer(256, 256),
                )
        # 8 x 8
        self.c4 = nn.Sequential(
                vgg_layer(256, 512),
                vgg_layer(512, 512),
                vgg_layer(512, 512),
                )
        # 4 x 4

        # torch.nn.Tanh
        #       Applies the Hyperbolic Tangent (Tanh) function element-wise.
        self.c5 = nn.Sequential(
                nn.Conv2d(512, dim, 4, 1, 0),
                nn.BatchNorm2d(dim),
                nn.Tanh()
                )

        # torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False)
        #       Applies a 2D max pooling over an input signal composed of several input planes.
        self.mp = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

    def forward(self, input):
        h1 = self.c1(input) # 64 -> 32
        h2 = self.c2(self.mp(h1)) # 32 -> 16
        h3 = self.c3(self.mp(h2)) # 16 -> 8
        h4 = self.c4(self.mp(h3)) # 8 -> 4
        h5 = self.c5(self.mp(h4)) # 4 -> 1
        return h5.view(-1, self.dim), [h1, h2, h3, h4]
```

Encoder output 出的這個 latent code(純粹是 encode 目前的 frame)，會再被送去一個普通的 LSTM 裡面，LSTM 就可以去學習時間上的關係，最後再 output 出一個除了目前這個 frame 的訊息之外，也帶有過去時間點的 frame 的資訊的 latent code。這個 latent code 再被送到 decoder 去做解碼，輸出預測的下一個 frame 的樣子。

Decoder 的 implement 方式如下圖所示。在架構中，除了和 encoder 一樣用到有 convolution 的 vgg layer 之外，因為 decoder 最重要的是要做 decode，所以在架構上還使用到了 deconvolution (transposed convolution)。

```python
class vgg_decoder(nn.Module):
    def __init__(self, dim):
        super(vgg_decoder, self).__init__()
        self.dim = dim
        # 1 x 1 -> 4 x 4
        self.upc1 = nn.Sequential(
                nn.ConvTranspose2d(dim, 512, 4, 1, 0),
                nn.BatchNorm2d(512),
                nn.LeakyReLU(0.2, inplace=True)
                )
        # 8 x 8
        self.upc2 = nn.Sequential(
                vgg_layer(512*2, 512),
                vgg_layer(512, 512),
                vgg_layer(512, 256)
                )
        # 16 x 16
        self.upc3 = nn.Sequential(
                vgg_layer(256*2, 256),
                vgg_layer(256, 256),
                vgg_layer(256, 128)
                )
        # 32 x 32
        self.upc4 = nn.Sequential(
                vgg_layer(128*2, 128),
                vgg_layer(128, 64)
                )
        # 64 x 64
        self.upc5 = nn.Sequential(
                vgg_layer(64*2, 64),
                nn.ConvTranspose2d(64, 3, 3, 1, 1),
                nn.Sigmoid()
                )
        self.up = nn.UpsamplingNearest2d(scale_factor=2)

    def forward(self, input):
        vec, skip = input
        d1 = self.upc1(vec.view(-1, self.dim, 1, 1)) # 1 -> 4
        up1 = self.up(d1) # 4 -> 8
        d2 = self.upc2(torch.cat([up1, skip[3]], 1)) # 8 x 8
        up2 = self.up(d2) # 8 -> 16
        d3 = self.upc3(torch.cat([up2, skip[2]], 1)) # 16 x 16
        up3 = self.up(d3) # 8 -> 32
        d4 = self.upc4(torch.cat([up3, skip[1]], 1)) # 32 x 32
        up4 = self.up(d4) # 32 -> 64
        output = self.upc5(torch.cat([up4, skip[0]], 1)) # 64 x 64
        return output

```

接著說明是如何實作 reparameterization trick 的部分。在 frame 通過 encoder，且再通過一個 gaussian LSTM 之後，會輸出兩個值，分別代表了這個過程中所學到的 gaussian distribution 的 mean 和 log variance。但是直接從這個分布 sample 出 z 值的話，會讓整個訓練過程無法 end-to-end 訓練(因為 sample 不能微分)，所以需要採用 reparameterization trick 的方式，如下圖所示。



也就是透過從 normal distribution 中抽出一個值，然後把它呈上目標的 gaussian 的 std，再加上目標 gaussian 的 mean，來作為從目標 gaussian 中抽出的值。

```
76      def reparameterize(self, mu, logvar):
77
78          std = torch.exp(0.5 * logvar)   # exp(logvar) -> var, exp(0.5 * logvar) = var^(1/2) = std
79
80          # Returns a tensor with the same size as input that is filled with random numbers from a normal distribution with mean 0 an
81          # torch.randn_like(input) is equivalent to torch.randn(input.size(), dtype=input.dtype, layout=input.layout, device=input.d
82          eps = torch.randn_like(std)     # sample epson from normal distribution
83
84          return mu + std * eps
```

Dataloader 的話，首先在建立 dataset 的時候，他就會根據 data 的 mode(train/validate/mode)，先去把那些對應 mode 的 data 的資料夾名字先記錄下來。

```
13  class bair_robot_pushing_dataset(Dataset):
14      def __init__(self, args, mode='train', transform=default_transform):
15          assert mode == 'train' or mode == 'test' or mode == 'validate'
16          self.root = '{}/{}'.format(args.data_root, mode)
17          self.seq_len = args.n_past + args.n_future
18          # self.seq_len = max(args.n_past + args.n_future, args.n_eval)
19          self.mode = mode
20          if mode == 'train':
21              self.ordered = False
22          else:
23              self.ordered = True
24
25          self.transform = transform
26          self.dirs = []
27          for dir1 in os.listdir(self.root):
28              for dir2 in os.listdir(os.path.join(self.root, dir1)):
29                  self.dirs.append(os.path.join(self.root, dir1, dir2))
30
31          self.seed_is_set = False
32          self.idx = 0
33          self.cur_dir = self.dirs[0]
```

然後再拿資料的時候(會呼叫__getitem__)，他就會去拿出對應 index 的資料夾的前 12 張照片(past + future = 12)用來作為各個時刻的 input image，以及 action.csv 和 endeffector_positions.csv 所記錄的機械手臂的 action 和位置用來當 input 所需的 condition。

```python
def get_seq(self):
    if self.ordered:
        self.cur_dir = self.dirs[self.idx]
        if self.idx == len(self.dirs) - 1:
            self.idx = 0
        else:
            self.idx += 1
    else:
        self.cur_dir = self.dirs[np.random.randint(len(self.dirs))]

    image_seq = []
    for i in range(self.seq_len):
        fname = '{}/{}.png'.format(self.cur_dir, i)
        img = Image.open(fname)
        image_seq.append(self.transform(img))   # image shape = (3, 64, 64)
    image_seq = torch.stack(image_seq)

    return image_seq

def get_csv(self):
    with open('{}/actions.csv'.format(self.cur_dir), newline='') as csvfile:
        rows = csv.reader(csvfile)
        actions = []
        for i, row in enumerate(rows):
            if i == self.seq_len:
                break
            action = [float(value) for value in row]
            actions.append(torch.tensor(action))

        actions = torch.stack(actions)
```

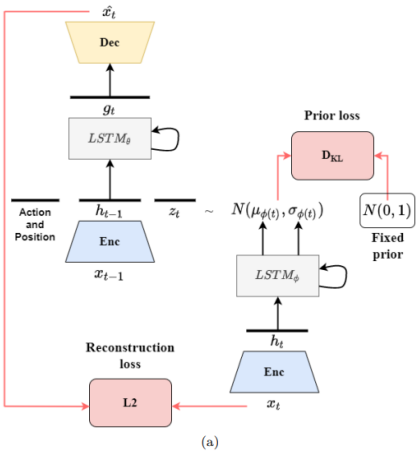
```python
def __getitem__(self, index):
    self.set_seed(index)
    seq = self.get_seq()
    cond =  self.get_csv()
    return seq, cond
```

- Describe the teacher forcing (including main idea, benefits and drawbacks.)

Teacher forcing 是一個常被用來訓練 RNN 的策略。一般在訓練這種有序列關係的資料的時候(像是這次的資料有時間先後的關係，或是句子翻譯的時候句子裡的詞也有先後順序的關係)，一般的訓練方式應該是拿 t-1 時刻的 output 作為 t 時刻的 input，但是這樣可能導致在一開始訓練的時候，因為 model 還沒有辦法預測出不錯的東西，把這個拿來當 input 的話就又更影響下一個時刻的訓練，導致訓練較不易。所以 teacher forcing 做的事，是透過在”train”的時候將 t-1 時刻的 ground truth 拿來做為 t 時刻的 input(“test”的時候維持一樣是用 t-1 output 做 t input，因為 test 的時候不應該有 ground truth 參與)，可以避免前面提到的拿很糟的預測當 input 去影響後面的訓練的問題，可以讓訓練過程收斂得更快一點。但是缺點就是，如果整個訓練過程，都是用這樣的方法來訓練的話，可能會導致 test 的時候，因為突然拿了預測的結果當 input 了，導致表現得沒有訓練時好；而且因為這樣在訓練的時候太過度依賴 ground truth 了，可能也會導致模型的 generalizability 沒有那麼好。

```python
use_teacher_forcing = True if random.random() < args.tfr else False

hidden = [modules['encoder'](x[i]) for i in range(args.n_past+args.n_future)]
for i in range(1, args.n_past + args.n_future):
    # print(i)
    h_t = hidden[i][0]    # each elememt of hidden includes (hidden code, [4 feature maps])

    if args.last_frame_skip or i < args.n_past:
        h_t_minus_1, skip = hidden[i-1]
    else:
        h_t_minus_1 = hidden[i-1][0]

    z_t, mu, logvar = modules['posterior'](h_t)

    # print(f'cond shape: {cond[i-1].shape}')
    # print(f'h_t_1 shape: {h_t_minus_1.shape}')
    # print(f'z_t shape: {z_t.shape}')
    # print(f'cat shape: {torch.cat([cond[i-1], h_t_minus_1, z_t], 1).shape}')

    g_t = modules['frame_predictor'](torch.cat([cond[i-1], h_t_minus_1, z_t], 1))
    x_pred = modules['decoder']([g_t, skip])
    mse += nn.MSELoss()(x_pred, x[i])
    kld += kl_criterion(mu, logvar, args)
    if not use_teacher_forcing :
        hidden[i] = modules['encoder'](x_pred)
    # raise NotImplementedError
beta = kl_anneal.get_beta()
loss = mse + kld * beta
loss.backward()
```

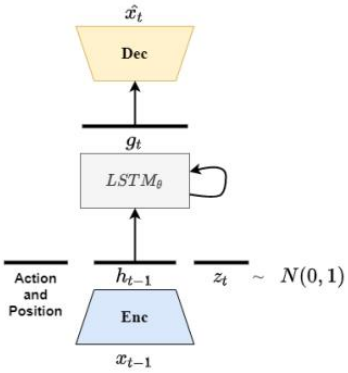
訓練時的 model 連結部分如上圖所示，是參考了 spec 中所提供的架構圖去做連結的。第 68 行和第 92 行就是當不使用 teacher forcing 的時候，會把 xt-1 替換成模型預測的結果。



(a)

```
317  def pred(x, cond, modules, epoch, args, device):
318      # initialize the hidden state.
319      modules['frame_predictor'].hidden = modules['frame_predictor'].init_hidden()
320      modules['posterior'].hidden = modules['posterior'].init_hidden()
321      x_0 = x[0]
322      pred_sequence = []
323      pred_sequence.append(x_0)
324      hidden = [modules['encoder'](x[i]) for i in range(args.n_past+args.n_future)]
325      for i in range(1, args.n_past + args.n_future):
326
327          # Tensor.detach()
328          # Returns a new Tensor, detached from the current graph.
329          # The result will never require gradient.
330          h_t = hidden[i][0].detach()    # each elememt of hidden includes (hidden code, [4 feature maps])
331
332          if args.last_frame_skip or i < args.n_past:
333              h_t_minus_1, skip = hidden[i-1]
334          else:
335              h_t_minus_1 = hidden[i-1][0]
336          h_t_minus_1 = h_t_minus_1.detach()
337
338          if i < args.n_past:
339              z_t, _, _ = modules['posterior'](hidden[i][0])      # hidden[i][0] = hidden code
340              modules['frame_predictor'](torch.cat([cond[i-1], h_t_minus_1, z_t], 1))
341              pred_sequence.append(x[i])
342          else:
343              z_t = torch.randn(args.batch_size, args.z_dim).to(device)
344              g_t = modules['frame_predictor'](torch.cat([cond[i-1], h_t_minus_1, z_t], 1)).detach()
345              x_pred = modules['decoder']([g_t, skip]).detach()
346              hidden[i] = modules['encoder'](x_pred)
347              pred_sequence.append(x_pred)
348              # print(f'pre_de')
349      return pred_sequence
```

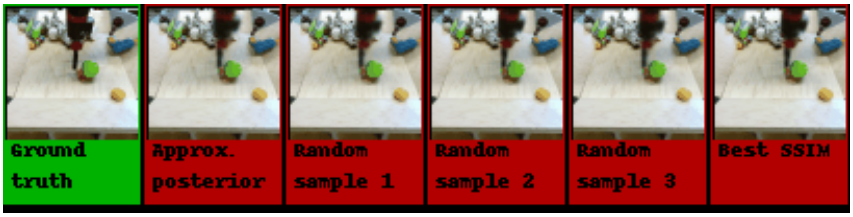
Test 和 validate 時的 model 連結部分如上圖所示，也是參考了 spec 中所提供的架構圖去做連結的。此時不論 tfr 為何，因為是 test/validate，xt-1 都必須使用模型預測出的結果(在開始要預測未來 frame 的時候。最前兩個就不用，仍是用給的 frame)。



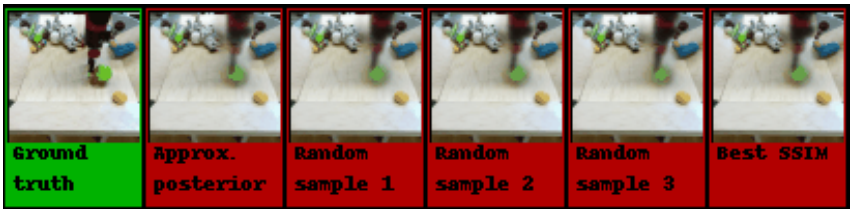
- Results and discussion
  - Show your results of video prediction

  (a) Make videos or gif images for test result (select one sequence)

  Cyclical:



  Monotonic:



  (b) Output the prediction at each time step (select one sequence)

  (c) Cyclical:

  Monotonic:

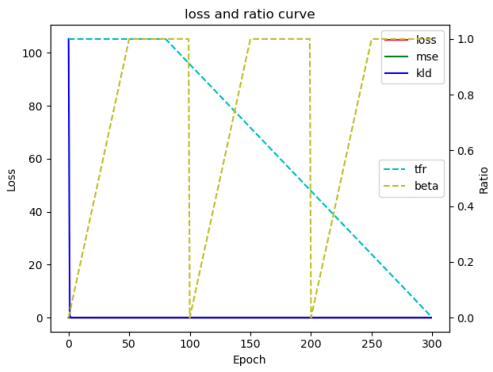
  - Plot the KL loss and PSNR curves during training

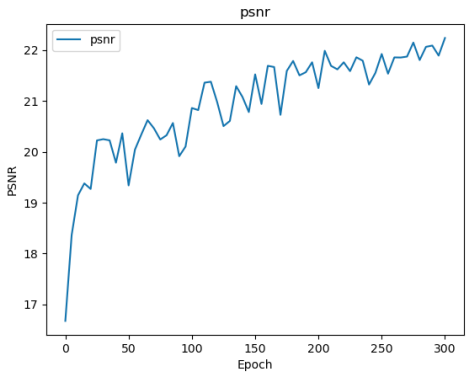
  以下的兩個結果是用同樣的 hyperparameter，但一個是 monotonic 的 KL，一個是 cyclical 的 KL train 到 300 個 epoch 的結果。

  除了 KL 之外的 hyperparameter 設定如下：

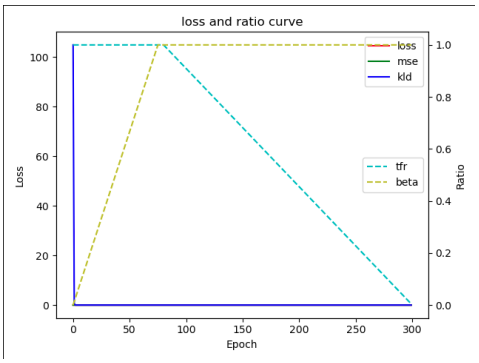
  Rnn_size = 256, z_dim = 128, g_dim = 64, tfr_start_decay_epoch = 80 (從第 80 個 epoch 之後，teach forcing ratio 會從 1 開始線性降低，直到第 300 個

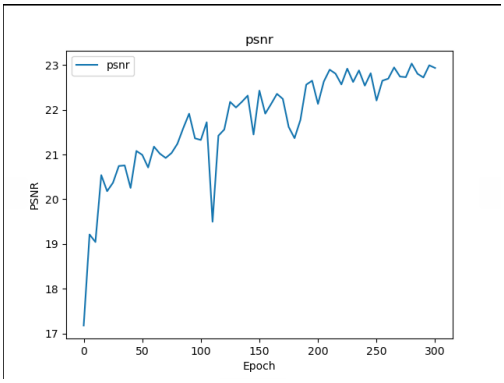
epoch 變成 0), batch_size = 16, niter = 300, last_frame_skip = true, lr = 0.002
Cyclical: train PSNR = 22.2, test PSNR = 21.8





Monotonic: train PSNR = 22.9, test PSNR = 22.7





• Discuss the results according to your setting of teacher forcing ratio, KL weight, and learning rate.

在訓練 VAE 的過程中，有可能會發生 KL vanish 的問題。就是說因為訓練 VAE 的目標是要 minimize reconstruction loss 和 kl loss，但是當 kl loss 真的幾乎等於 0 的時候，posterior q(z|x)和 prior q(z)就變得幾乎相同，就相當於 encoder 在 encode input x 變成 latent z 的時候，input x 和 latent z 幾乎就變得獨立了，因此也導致 decoder 最後在預測出 x'的時候，並沒有依靠 latent z 來預測，而是光靠很強的 decoder 自己就能預測。但這樣的結果並不是我們想要的，因此引進了 KL annealing 的作法。KL annealing 的做法，在計算 loss 的時候，kl loss 項會乘上一個變數(也就是這裡的 beta)，透過這個變數來控制 model 學習時關注的地方。在一開始的時候，beta 會被設為 0.001(趨近於 0)，然後隨著 training epoch 數變多 beta 會慢慢增加為一。這樣的做法的好處是，因為最一開始的時候，KL 的 loss 非常大(也就是 posterior q(z|x)和 prior q(z)的 KL divergence 差的很多)，beta 設成 0 可以避免 loss 全部被 kl loss dominate。而且這樣也可以幫助在一開始的時候，loss 主要來自 reconstruction term，可以先讓 model 學習如何預測出比較好的照片，這麼做的同時也對於減少 KL loss 有幫助(雖然讓 reconstruction term 和 KL term 最小的方向不完全一樣，但他們也不是反方向，因此也能幫助降低

KL term)。

雖然從上圖的 loss and ratio curve 中無法直接觀察出來(因為 KL loss 在 epoch 0 的時候很大，所以把 loss 那邊的 scale 拉得很高，所以即使之後的 KL loss 有稍微改變，在圖上看起來都還是趨近 0 的樣子)，但可以從 training 的紀錄中直接觀察：在一開始的時候 kl loss 還很大，因為 posterior q(z|x)和 prior q(z)的 KL divergence 還差的很多，幾個 epoch 之後，他就幾乎迅速的降到 0。

```
2 [epoch: 00] loss: 0.00459 | mse loss: 0.00459 | kld loss: 105.19369
3 ====================== validate psnr = 16.67621 ========================
4 [epoch: 01] loss: 0.01991 | mse loss: 0.00506 | kld loss: 0.74220
5 [epoch: 02] loss: 0.00399 | mse loss: 0.00393 | kld loss: 0.00162
6 [epoch: 03] loss: 0.00328 | mse loss: 0.00324 | kld loss: 0.00072
7 [epoch: 04] loss: 0.00303 | mse loss: 0.00296 | kld loss: 0.00083
8 [epoch: 05] loss: 0.00280 | mse loss: 0.00273 | kld loss: 0.00066
9 ====================== validate psnr = 18.36506 ========================
```

而 monotonic KL 和 cyclical KL 是差在，monotonic KL 的 beta 就是從 0 慢慢變 1，之後就一直維持 1；cyclical KL 的 beta 變化則是有周期的，beta 從 0 變成 1 之後，在週期到了以後，會再變回 0 然後再慢慢變成 1。從 training 的紀錄中可以發現，在 kl annealing 完成一個 cycle，進入下一個 cycle 的時候(也就是 epoch 100 和 epoch 200 的時候)，此時的 kl loss 都會突然的變高(因為又突然都變成幾乎用 reconstruction term 來更新了)。

```
117 ====================== validate psnr = 20.10453 ========================
118 [epoch: 96] loss: 0.00187 | mse loss: 0.00177 | kld loss: 0.00010
119 [epoch: 97] loss: 0.00176 | mse loss: 0.00176 | kld loss: 0.00000
120 [epoch: 98] loss: 0.00158 | mse loss: 0.00158 | kld loss: 0.00000
121 [epoch: 99] loss: 0.00190 | mse loss: 0.00189 | kld loss: 0.00000
122 [epoch: 100] loss: 0.00165 | mse loss: 0.00165 | kld loss: 0.00200
123 ====================== validate psnr = 20.86201 ========================
124 [epoch: 101] loss: 0.00175 | mse loss: 0.00175 | kld loss: 0.00008
125 [epoch: 102] loss: 0.00180 | mse loss: 0.00179 | kld loss: 0.00003
126 [epoch: 103] loss: 0.00180 | mse loss: 0.00180 | kld loss: 0.00003
127 [epoch: 104] loss: 0.00181 | mse loss: 0.00181 | kld loss: 0.00002
128 [epoch: 105] loss: 0.00164 | mse loss: 0.00164 | kld loss: 0.00001
129 ====================== validate psnr = 20.82156 ========================
```

```
237 ====================== validate psnr = 21.76129 ========================
238 [epoch: 196] loss: 0.00238 | mse loss: 0.00237 | kld loss: 0.00000
239 [epoch: 197] loss: 0.00223 | mse loss: 0.00223 | kld loss: 0.00000
240 [epoch: 198] loss: 0.00229 | mse loss: 0.00229 | kld loss: 0.00000
241 [epoch: 199] loss: 0.00238 | mse loss: 0.00238 | kld loss: 0.00000
242 [epoch: 200] loss: 0.00228 | mse loss: 0.00228 | kld loss: 0.00933
243 ====================== validate psnr = 21.25363 ========================
244 [epoch: 201] loss: 0.00236 | mse loss: 0.00236 | kld loss: 0.00026
245 [epoch: 202] loss: 0.00266 | mse loss: 0.00262 | kld loss: 0.00096
246 [epoch: 203] loss: 0.00235 | mse loss: 0.00235 | kld loss: 0.00002
247 [epoch: 204] loss: 0.00230 | mse loss: 0.00230 | kld loss: 0.00002
248 [epoch: 205] loss: 0.00222 | mse loss: 0.00221 | kld loss: 0.00001
```

在我的訓練結果中，monotonic KL 的訓練結果在最後其實稍稍為比 cyclical KL 的訓練結果好了一點點(但一般來說應該要是 cyclical)，我覺得這個原因可能是因為在最易開始的 epoch 的時候，monotonic 的初始 PSNR 有 17.1 而 cyclical 的初始 PSNR 只有 16.6，雖然 cyclical 在訓練過程中應該較好，但這個影響還是比不上初始造成的影響

而在對 teacher forcing ratio 作調整的時候，可以發現當 teacher forcing ratio 開始下降的時候(以上面的例子，是第 80 個 epoch 開始的時候)，此時 loss 也會稍微變高一點。因為拿來當 input 的資料，開始不再完全只有 ground truth 了，開始有了預測出的有 bias 的 data，所以使得預測出的 loss 稍微變大了。但是這樣做了以後，可以幫助模型慢慢去學習如何在有 bias 的情況下預測出好的結果，因此在這之後，預測出的 image 的平均 psnr 仍有慢慢上升。

```
92 [epoch: 75] loss: 0.00132 | mse loss: 0.00115 | kld loss: 0.00017
93 ====================== validate psnr = 20.24319 ========================
94 [epoch: 76] loss: 0.00131 | mse loss: 0.00114 | kld loss: 0.00017
95 [epoch: 77] loss: 0.00132 | mse loss: 0.00115 | kld loss: 0.00017
96 [epoch: 78] loss: 0.00130 | mse loss: 0.00113 | kld loss: 0.00017
97 [epoch: 79] loss: 0.00131 | mse loss: 0.00114 | kld loss: 0.00017
98 [epoch: 80] loss: 0.00130 | mse loss: 0.00114 | kld loss: 0.00016
99 ====================== validate psnr = 20.32701 ========================
00 [epoch: 81] loss: 0.00167 | mse loss: 0.00151 | kld loss: 0.00016
01 [epoch: 82] loss: 0.00163 | mse loss: 0.00147 | kld loss: 0.00017
02 [epoch: 83] loss: 0.00166 | mse loss: 0.00150 | kld loss: 0.00016
03 [epoch: 84] loss: 0.00151 | mse loss: 0.00134 | kld loss: 0.00016
04 [epoch: 85] loss: 0.00163 | mse loss: 0.00152 | kld loss: 0.00011
05 ====================== validate psnr = 20.56716 ========================
06 [epoch: 86] loss: 0.00193 | mse loss: 0.00184 | kld loss: 0.00010
07 [epoch: 87] loss: 0.00161 | mse loss: 0.00160 | kld loss: 0.00001
08 [epoch: 88] loss: 0.00162 | mse loss: 0.00160 | kld loss: 0.00002
```

我自己訓練的時候，不知道為什麼感覺訓練的 PSNR 一直沒辦法像朋友的一樣高，就算我們使用了相同的 hyperparameter。因為在我的結果中，感覺訓練的 PSNR 趨勢是對的，不過感覺整體 PSNR 增加的趨勢沒有那麼快。原本有想過是不是 lr 太小的關係，所以有試過把 lr 從原本的 0.002 調到 0.004 和 0.003 過，但是出來的結果在前半部都比原來的還要差很多，因此就沒有繼續把它 train 完。
也試過把 batch size 在調大一點到 22 或是 20，不過兩個都因為 cuda memory 不足而無法 train，因此最後還是維持 16。

我後來終於發現原因了！助教的 Lab5 spec 上寫的預設參數 z_dim = 128, g_dim = 64，我一開始是照著這個設定的，但實際上應該要是 z_dim = 64, g_dim = 128，所以前面的結果才會那麼低。後來我將 z_dim 和 g_dim 改成正確的值之後重新 train，就能達到 PSNR 25 以上了。

- Reference

https://zhuanlan.zhihu.com/p/340568861