Introduction (5%)

在 Lab5 中，我們要利用 CVAE 架構配合 LSTM 模型來做影片預測模型，用影片中的前兩張幀來預測後面 10 幀。

CVAE 全名為 Conditional Variational Autoencoder，是一個較為基本的 Autoencoder(AE)，AE 其實是由 encoder 和 decoder 組成，encoder 原來是把資料編碼成一堆數字方便儲存，然後再通過 decoder 來解碼得出原來的資料。AE 則是把高維度的影像壓縮成低維度的 latent vector，再藉由 decoder 來把 latent vector 解碼成為原來的輸入。

VAE 的模型把 encoder 的輸入生成標準分佈，然後再結合其中的 sample 來作為解碼器的輸入，最後生成原來的輸出。本次實驗中的 CVAE 在結構上利用$\mathrm{LSTM}_{\emptyset}$的模型把 encoder 的輸入生成標準分佈，然後再結合 sample 的 z、前一個輸入(X_{t-1})的 latent vector 和 condition 來作為解碼器的輸入，最後依賴LSTM_{θ}的特性來預測並生成下一個輸出(X_t)。

加入 condition 主要原因是想藉由 condition 的輸入增加可調控的因素，把抽樣出來的 Z_t 結合前一個輸入和可控的 condition 就能生成下一個輸入，直觀來說已經算是一個 generator。

Derivation of CVAE (Please use the same notation in Fig.1a)(10%)

VAE equation: $L(X,q,\theta) = E_{z\sim q(Z|X;\emptyset)} logp(X|Z;\theta) - KL(q(Z|X;\emptyset)||p(Z))$

CVAE equation: $L(X,q,\theta|C) = E_{z\sim q(Z|X;\emptyset)} logp(X|Z,C;\theta) - KL(q(Z|X,C;\emptyset)||p(Z|C))$ where

$$L(X,q,\theta|C) = \int q(Z|C)\, log\, p(X|Z,C;\theta)\, dZ - \int q(Z|C)\, log\, q(Z|C)\, dZ$$

$$KL(q(Z|X,C;\emptyset)||p(Z|C)) = \int q(Z|X,C;\emptyset)\, log \frac{q(Z|X,C;\emptyset)}{p(Z|C)} dZ$$
$$= \int q(Z|X,C;\emptyset)\, log(q(Z|X,C;\emptyset) - p(Z|C))\, dZ$$

Derivation:

$$p(X|Z,C;\theta) = \frac{p(X,Z|C;\theta)}{p(Z|X,C;\theta)}$$

$$
\begin{aligned}
logp(X|Z,C;\theta) &= logp(X|Z,C;\theta) - logp(Z|X,C;\theta) \\
&= \int q(Z|C)\log p(X|Z,C;\theta)\,dZ - \int q(Z|C)\log q(Z|C)\,dZ \\
&+ \int q(Z|C)\log q(Z|C)\,dZ - \int q(Z|C)\log p(Z|X,C;\theta)\,dZ \\
&= L(X,q,\theta|C) + KL(q(Z|C)||p(Z|X,C;\theta))
\end{aligned}
$$

$$
\begin{aligned}
L(X,q,\theta|C) &= logp(X|Z,C;\theta) - KL(q(Z|C)||p(Z|C)) \\
&= E_{Z\sim q(Z|X,C;\theta)}\log p(X|Z,C;\theta) + E_{Z\sim q(Z|C;\theta)} logp(Z|C) \; - E_{Z\sim q(Z|X,C;\theta)} \\
&= E_{Z\sim q(Z|X,C;\theta)}\log p(X|Z,C;\theta) + \frac{1}{N}\sum_{i=1}^{N} q(Z|C;\theta)\, logp(Z|C) \; - logq(Z|X,C;\theta) \\
&= E_{Z\sim q(Z|X,C;\theta)}\log p(X|Z,C;\theta) + KL(q(Z|C)||p(Z|X,C;\theta))
\end{aligned}
$$

$$
\begin{aligned}
\therefore \quad E_{Z\sim q(Z|X;\emptyset)} logp(X|Z,C;\theta) \; &= \; L(X,q,\theta|C) \; + KL(q(Z|X,C;\emptyset)||p(Z|C)) \\
&\geq L(X,q,\theta|C) \\
&= \; E_{Z\sim q(Z|X,C;\emptyset)}\log p(X|Z,C;\emptyset) + KL(q(Z|C)||p(Z|X,C;\emptyset))
\end{aligned}
$$

Implementation details (15%)

- Describe how you implement your model. (e.g. dataloader, encoder, decoder, etc) (10%)

Dataset loader:

```
class bair_robot_pushing_dataset(Dataset):
    def __init__(self, args, mode='trian', transform=default_transform, frame_num=12, 
frame_in=2):
        assert mode == 'train' or mode == 'test' or mode == 'validate' or mode == 'trial'
        self.mode = '/' + mode + '/'
        self.args = args
        self.transform = transform
        # self.seed = args.seed
        self.seed_is_set = False
        self.frame_num = frame_num
        self.frame_in_num = 2
        self.batches_len = 0
        self.dirpath = self.args.data_root + self.mode
        self.seq = self.get_seq()
        self.csv = self.get_csv()

    def set_seed(self, seed):
        # print(self.seed_is_set)
        if not self.seed_is_set:
            self.seed_is_set = True
            np.random.seed(seed)

    def __len__(self):
        return self.batches_len
```

```
    def get_seq(self):
        frames = torch.tensor([])  # .to(gpu, dtype=torch.float)
        for (dirpath, dirnames, filenames) in os.walk(self.dirpath):
            for x in filenames:
                if x.endswith(".png") and filenames.index(x) < self.frame_num:
                    img = Image.open(dirpath + '/' + x)
                    # convert_tensor = transforms.ToTensor()
                    img_tensor = self.transform(img)
                    img_tensor = img_tensor  # .to(gpu, dtype=torch.float)
                    frames = torch.cat((frames, img_tensor))
                else:
                    continue
        frame_len = int(frames.size()[0] / (self.frame_num * 3))
        frames = torch.reshape(frames, (frame_len, self.frame_num, 3, 64, 64))
        return frames

    def get_csv(self):
        excel_action = torch.tensor([])  # .to(gpu, dtype=torch.float)
        excel_end = torch.tensor([])  # .to(gpu, dtype=torch.float)
        excel_action_total = torch.tensor([])  # .to(gpu, dtype=torch.float)
        excel_end_total = torch.tensor([])  # .to(gpu, dtype=torch.float)
        for (dirpath, dirnames, filenames) in os.walk(self.dirpath):
            for x in filenames:
                if x.startswith("action"):
                    excel_action = get_text(dirpath, x, self.frame_num)
                    excel_action_total = torch.cat((excel_action_total, excel_action))
                elif x.startswith("endeffector"):
                    # print('end: ', x)
                    excel_end = get_text(dirpath, x, self.frame_num)
                    excel_end_total = torch.cat((excel_end_total, excel_end))
                else:
                    continue

        excel_total = torch.cat((excel_action_total, excel_end_total), 1)
        # print(excel_total.size())
        action_len = int(excel_action_total.size()[0] / self.frame_num)
        self.batches_len = action_len
        excel_total = torch.reshape(excel_total, (action_len, self.frame_num, 7))
        return excel_total

    def __getitem__(self, index):
        self.set_seed(index)
        seq = self.seq[index]
        cond = self.csv[index]
        return seq, cond


def get_text(dirpath, filename, frame_num):
    excel = torch.tensor([])  # .to(gpu, dtype=torch.float)
    row_num = 0
    with open(dirpath + '/' + filename, newline='') as csvfile:
        spamreader = csv.reader(csvfile, delimiter=' ', quotechar='|')
        for row in spamreader:
            if row_num < frame_num:
                text = row[0].split(",")  # [', '.join(row)]
```

```python
                text_arr = np.array([text], dtype=float)
                text_tns = torch.Tensor(text_arr)
                text_tns = text_tns  # .to(gpu, dtype=torch.float32)
                excel = torch.cat((excel, text_tns))
                row_num += 1
            else:
                break
    return excel
```

首先我們有兩種資料需要讀取，分別是 image 和 condition，先以 os.walk 走訪 dataset 裡面的所有資料，並區分.png 和.csv，根據輸入 frame_len 來決定每個資料夾讀取資料的數量，預設為讀取 12 張圖像後開始讀取下個資料夾，最後將資料 reshape 成我們需要的形狀，然後 return 給 get item。

最後因為這個方法要先完整讀取資料太慢了，所以在 coding 上有改過，改為儲存讀取資料的路徑，這樣在讀取 data 時就不用一整筆資料讀取，可以一筆筆輸入到 model 裡訓練。

Train process:

```python
def train(x, cond, modules, optimizer, kl_anneal, args, device):
    criterion = nn.CrossEntropyLoss()
    modules['frame_predictor'].zero_grad()
    modules['posterior'].zero_grad()
    modules['encoder'].zero_grad()
    modules['decoder'].zero_grad()

    # initialize the hidden state.
    modules['frame_predictor'].hidden = modules['frame_predictor'].init_hidden()
    modules['posterior'].hidden = modules['posterior'].init_hidden()
    mse = 0
    kld = 0
    use_teacher_forcing = True if random.random() < args.tfr else False
    decoder_output_total = torch.tensor(x[0].unsqueeze(0)).to(device)
    encoder_output_total = [modules['encoder'](x[0])]  # h5.view(-1, self.dim), [h1~4]
    for i in range(1, args.n_past + args.n_future):  # frames_size
        if use_teacher_forcing or i == 1:
            encoder_output_total.append(modules['encoder'](x[i]))
            if args.last_frame_skip or i < args.n_past:
                encoder_output_past, remain = encoder_output_total[i - 1]  # h5, [h1~4]
            else:
                encoder_output_past = encoder_output_total[i - 1][0]
            z, mu, logvar = modules['posterior'](encoder_output_total[i][0])
        else:
            encoder_output_past = modules['encoder'](decoder_output_total[i - 1])[0]
            mu, logvar = args.mu, args.var
            z = reparameter(mu, logvar, (args.batch_size, args.z_dim), device)
        lstm_input = torch.cat((encoder_output_past, cond[i - 1], z), 1)
        lstm_output = modules['frame_predictor'](lstm_input)
        decoder_output = modules['decoder']((lstm_output, remain))
        decoder_output_total = torch.cat((decoder_output_total, decoder_output.unsqueeze(0)))
```

```
        mse += criterion(decoder_output, x[i])
        kld += kl_criterion(mu, logvar, args)

    loss = mse + kld * args.kl_beta
    loss.backward()
    optimizer.step()

    return loss.detach().cpu().numpy() / (args.n_past + args.n_future), mse.detach().cpu().numpy()
(args.n_past + args.n_future), kld.detach().cpu().numpy() / (args.n_future + args.n_past)
```

首先我們根據已知的 2 幀作為 encoder 輸入，再透過$LSTM_{\emptyset}$的模型來生成標準分佈，然後作為前 1 幀的 encoder 輸入來產 latent vector，之後再輸入到的$LSTM_{\theta}$模型來預測下 1 幀的輸出，再透過 decoder 解碼，最後再根據對比輸出的 X_{t_pred} 和輸入的 X_t 的 loss 用 BP 來更新所有模型的參數。而 teacher forcing 會以提供輸入來強迫模型學習，當 teacher focing 為 True，模型會把 groud truth 作為輸入，當 teacher focing 為 False，模型會以自己的 decoder output 作為輸入。

KL Annealing:

```
class kl_annealing():
    def __init__(self, args):
        super().__init__()
        self.epoch = 0
        self.period = args.niter / args.kl_anneal_cycle   # 300/3 = 100
        self.ratio = self.period / args.kl_anneal_ratio   # 100/2 = 50
        self.step = 1 / self.ratio  # 1/50 = 0.02

    def update(self, cyclical):
        if cyclical:
            self.epoch %= self.period
            beta = self.epoch * self.step
        else:
            beta = self.epoch * (self.step / 2)

        return 1 if beta > 1 else beta

    def get_beta(self, epoch, cyclical):
        self.epoch = epoch
        return self.update(cyclical)
```

KL annealing 會根據輸入的 epoch 來調整 KL Divergence loss，在開始訓練時會先以 0 開始，然後慢慢根據 KL beta 來調整，而此次實驗中我們用 Monotonic 和 Cyclical 兩種不同 beta 變化來實作並比較差別。

Teacher-forcing:

```
if epoch >= args.tfr_start_decay_epoch and args.tfr >= args.tfr_lower_bound:
    args.tfr = args.tfr - args.tfr_decay_step
```

根據 epoch 來慢慢降低 teacher-forcing 的 ground truth 輸入，令 encoder-decoder 先開始學習。

Reparameterize:

```
def reparameterize(self, mu, logvar):
    std = torch.exp(0.5*logvar)
    eps = torch.randn_like(std)
    return mu + eps*std
```

根據$LSTM_{\phi}$輸出的 μ 和 logσ 來生成 z 以用作$LSTM_{\theta}$的預測輸入。

Valid/Test code:

```
def pred_norm(validate_seq, validate_cond, modules, args, device):
        decoder_output_total = torch.tensor(validate_seq[0].unsqueeze(0)).to(device)
    encoder_output_total = [modules['encoder'](validate_seq[0
    for i in range(1, args.n_past + args.n_future):  # frames_size
        if i == 1:
            encoder_output_total.append(modules['encoder'](validate_seq[i]))
            z, mu, logvar = modules['posterior'](encoder_output_total[i][0])
            if args.last_frame_skip or i < args.n_past:
                encoder_output_past, remain = encoder_output_total[i - 1]  # h5
            else:
                encoder_output_past = encoder_output_total[i - 1][0]
        else:
            encoder_output_past = modules['encoder'](decoder_output_total[i - 1])[0]
            mu, logvar = args.mu, args.var
            z = reparameter(mu, logvar, (args.batch_size, args.z_dim), device)
        lstm_input = torch.cat((encoder_output_past, validate_cond[i-1], z), 1)
        lstm_output = modules['frame_predictor'](lstm_input)
        decoder_output = modules['decoder']((lstm_output, remain))
        decoder_output_total = torch.cat((decoder_output_total, decoder_output.unsqueeze(0)))
    return decoder_output_total


def pred_rec(validate_seq, validate_cond, modules, args, device):
    decoder_output_total = torch.tensor(validate_seq[0].unsqueeze(0)).to(device)
    encoder_output_total = [modules['encoder'](validate_seq[0])]  # h5
    for i in range(1, args.n_past + args.n_future):  # frames_size
        encoder_output_total.append(modules['encoder'](validate_seq[i]))
        z, mu, logvar = modules['posterior'](encoder_output_total[i][0])
        if i == 1:
            if args.last_frame_skip or i < args.n_past:
                encoder_output_past, remain = encoder_output_total[i - 1]  # h5
            else:
```

```
                encoder_output_past = encoder_output_total[i - 1][0]
        else:
            encoder_output_past = modules['encoder'](decoder_output_total[i - 1])[0]
        lstm_input = torch.cat((encoder_output_past, validate_cond[i-1], z), 1)
        lstm_output = modules['frame_predictor'](lstm_input)
        decoder_output = modules['decoder']((lstm_output, remain))
        decoder_output_total = torch.cat((decoder_output_total, decoder_output.unsqueeze(0)))
    return decoder_output_total
```

基本上和 training procedure 大致相同，只是移除了 teacher forcing 部份，只用前 2 幀作為輸入，之後都是以 decoder 輸出作為 encoder 輸入，而 pred_norm 部份則以 N(0,1)標準分佈來取代原有的$LSTM_{\phi}$生成的標準分佈。

Save image and plot it:

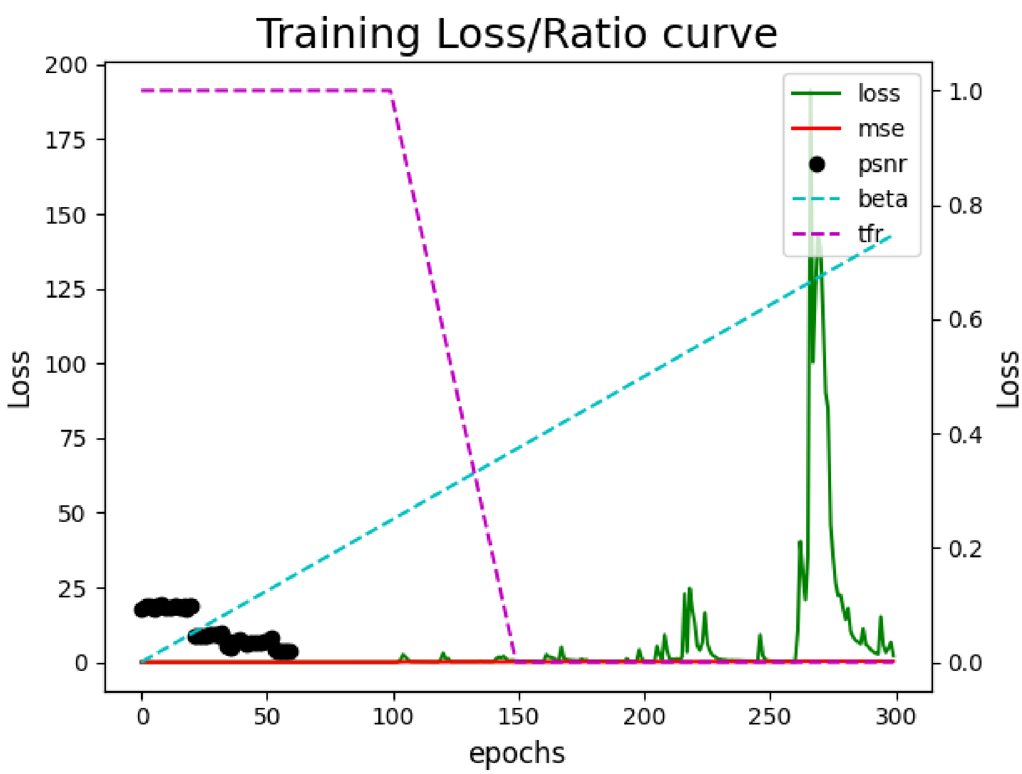
```
def show_from_tensor(tensor, path, title=None):
    img = tensor.clone()
    img = tensor_to_np(img)
    plt.figure()
    plt.axis("off")
    plt.imshow(img)
    if title is not None:
        plt.title(title)
    plt.savefig(path + title)
    plt.close()

def tensor_to_np(tensor):
    img = tensor.mul(255).byte()
    img = img.cpu().numpy().transpose((1, 2, 0))
    return img

def reparameter(mu, logvar, size, device):
    var = torch.ones(size).to(device) * logvar
    mu = torch.zeros(size).to(device) * mu
    std = torch.exp(0.5 * var)
    eps = torch.randn_like(std)
    z = mu + eps * std
    return z

def plot_output(ref_seq, pred_seq, rec_seq, args, spt=1):
    path = './gif/'
    frame_num = args.n_past + args.n_future

    for batch in range(args.batch_size):
        filenames1, filenames2, filenames3 = [], [], []
        for frame in range(frame_num):
            filename1 = f'{str(1)}_{batch}_{frame}.png'
            show_from_tensor(ref_seq[frame, batch], path, filename1)  # save fig
            filenames1.append(path + filename1)

            filename2 = f'{str(2)}_{batch}_{frame}.png'
            show_from_tensor(pred_seq[frame, batch], path, filename2)  # save fig
```

```
            filenames2.append(path + filename2)

            filename3 = f'{str(3)}_{batch}_{frame}.png'
            show_from_tensor(rec_seq[frame, batch], path, filename3)  # save fig
            filenames3.append(path + filename3)
        # build gif
        with imageio.get_writer(path + 'ref_seq_' + str(batch) + '.gif', mode='I') as writer:
            for filename in filenames1:
                image1 = imageio.imread(filename)
                writer.append_data(image1)

        with imageio.get_writer(path + 'pred_seq_' + str(batch) + '.gif', mode='I') as
writer:
            for filename in filenames2:
                image2 = imageio.imread(filename)
                writer.append_data(image2)

        with imageio.get_writer(path + 'rec_seq' + str(batch) + '.gif', mode='I') as writer:
            for filename in filenames3:
                image3 = imageio.imread(filename)
                writer.append_data(image3)

        fig = plt.figure()
        plt.subplot(1, 3, spt)
        plt.title(str(spt))
        plt.axis("off")
        ims = []
        for frame in range(frame_num):
            img = Img.imread(path + f'{str(spt)}_{batch}_{frame}.png')
            # print('img: ', img.shape)
            im = plt.imshow(img, animated=True)
            ims.append([im])

        spt = 2
        plt.subplot(1, 3, spt)
        plt.title(str(spt))
        plt.axis("off")
        ims2 = []
        for frame in range(frame_num):
            img2 = Img.imread(path + f'{str(spt)}_{batch}_{frame}.png')
            im2 = plt.imshow(img2, animated=True)
            ims2.append([im2])

        spt = 3
        plt.subplot(1, 3, spt)
        plt.title(str(spt))
        plt.axis("off")
        ims3 = []
        for frame in range(frame_num):
            img3 = Img.imread(path + f'{str(spt)}_{batch}_{frame}.png')
            im3 = plt.imshow(img3, animated=True)
            ims3.append([im3])

        ani = animation.ArtistAnimation(fig, ims, interval=50, blit=True, repeat_delay=1000)
        ani2 = animation.ArtistAnimation(fig, ims2, interval=50, blit=True,
```

```
repeat_delay=1000)
        ani3 = animation.ArtistAnimation(fig, ims3, interval=50, blit=True, 
repeat_delay=1000)
        # plt.show()
        # plt.close(fig)
```

將生成的圖像生成並儲存為.png 檔然後轉成.gif 檔並顯示出來(如有需要)。

- Describe the teacher forcing (including main idea, benefits and drawbacks) (5%)
  如上一部份的說明，teacher forcing 是以 groud truth 來迫使模型學習，但如果一直使用 teacher forcing，模型可以學習得很快，但很容易 overfitting。
  我試著用比較大的 tfr 然後慢慢提升 beta，結果不太理想，loss 還突然變大很多，看起來像是 overfitting 遇到不太一樣的情境。



Notice: You must prove that you use previous predicted frame to predict next frame, i.e. teacher forcing ratio = 0 when testing (paste/screenshot your code

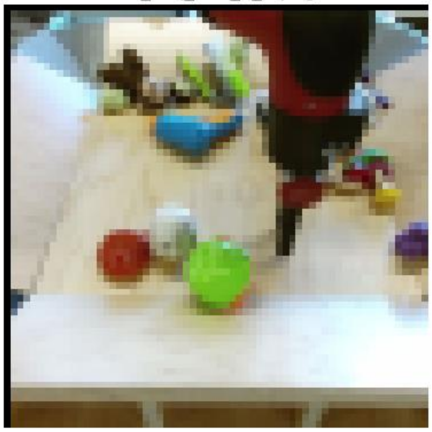
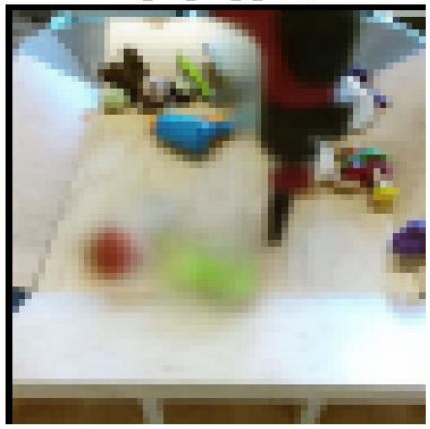
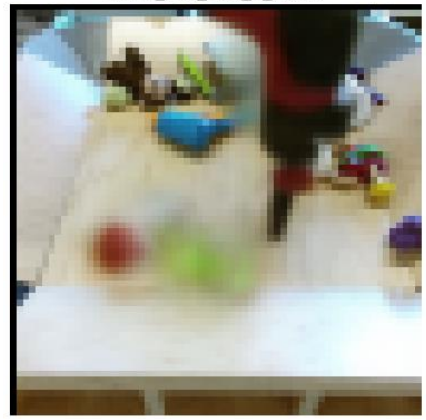
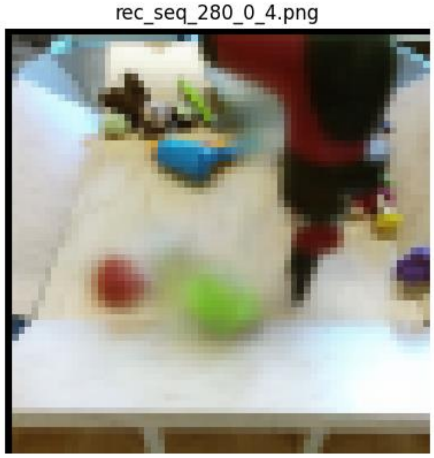
Results and discussion (30%)

- Show your results of video prediction (10%)
  - Make videos or gif images for test result (5%)
    用了短網址上傳了可用 30 天的 gif，沒有密碼，直接按確認：
    目標 gif: https://imgus.cc/dP8v2
    CVAE 預測 gif: https://imgus.cc/7oxA5
    靜態分佈抽樣預測 gif: https://imgus.cc/GGWPL
    或查看附上的 gif，對應以上的網址:
    norm_seq_280_0, rec_seq_280_0, ref_seq_280_0
    PSNR: 24.5 左右
    可以看到，其實 gif 的動作和原本的 gif 其實是非常相似的，但有一些細節的地方不夠清楚。

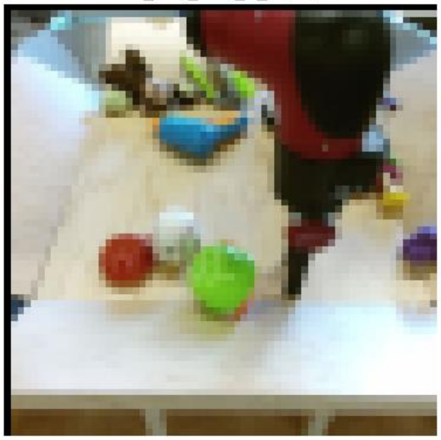
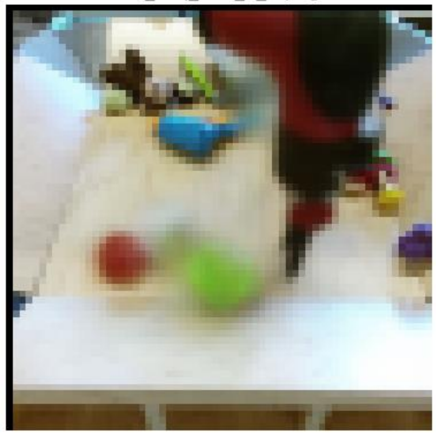
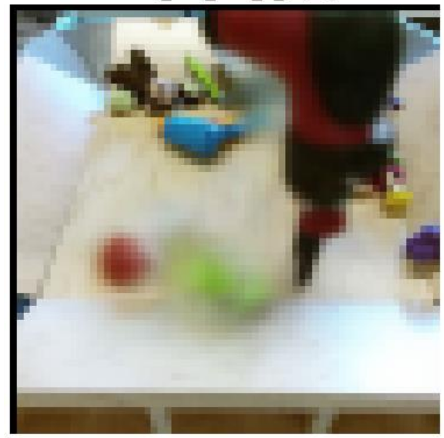
  - Output the prediction at each time step (5%)

目標圖片	CVAE 模型預測	CVAE 模型預測(靜態抽樣)
ref_seq_280_0_0.png 	rec_seq_280_0_0.png 	norm_seq_280_0_0.png 
ref_seq_280_0_1.png 	rec_seq_280_0_1.png 	norm_seq_280_0_1.png 

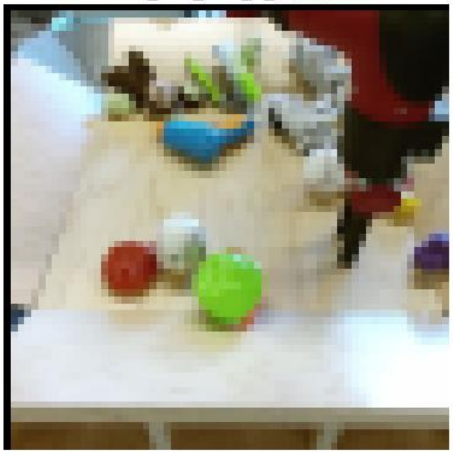
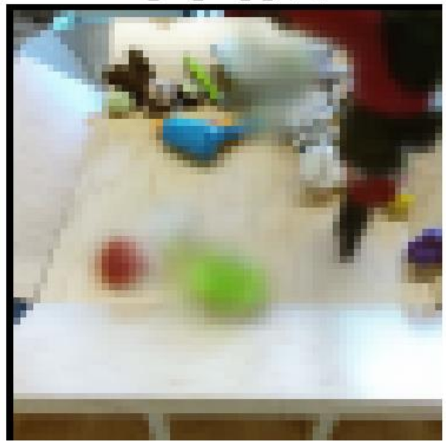
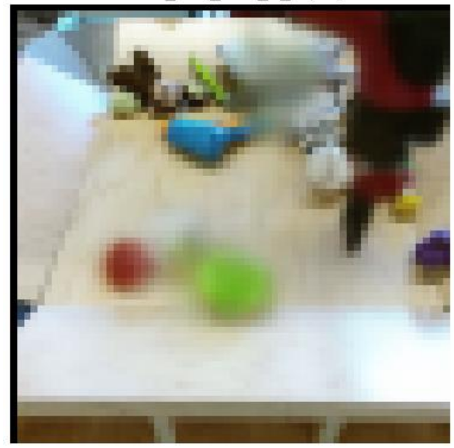
DLP_LAB5_310551003_陳嘉慶

ref_seq_280_0_2.png 	rec_seq_280_0_2.png 	norm_seq_280_0_2.png 
ref_seq_280_0_3.png 	rec_seq_280_0_3.png 	norm_seq_280_0_3.png 
ref_seq_280_0_4.png 	rec_seq_280_0_4.png 	norm_seq_280_0_4.png 

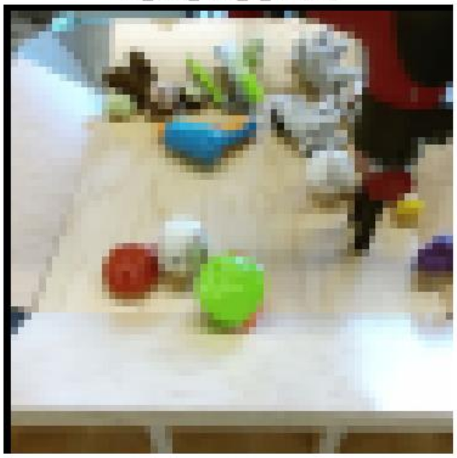
DLP_LAB5_310551003_陳嘉慶

ref_seq_280_0_5.png 	rec_seq_280_0_5.png 	norm_seq_280_0_5.png 
ref_seq_280_0_6.png 	rec_seq_280_0_6.png 	norm_seq_280_0_6.png 
ref_seq_280_0_7.png 	rec_seq_280_0_7.png 	norm_seq_280_0_7.png 

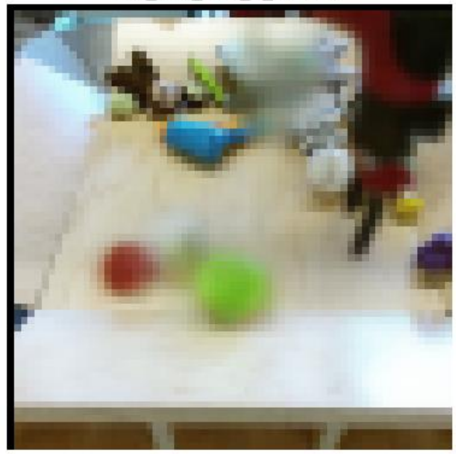
DLP_LAB5_310551003_陳嘉慶

ref_seq_280_0_8.png 	rec_seq_280_0_8.png 	norm_seq_280_0_8.png 
ref_seq_280_0_9.png 	rec_seq_280_0_9.png 	norm_seq_280_0_9.png 
ref_seq_280_0_10.png 	rec_seq_280_0_10.png 	norm_seq_280_0_10.png 

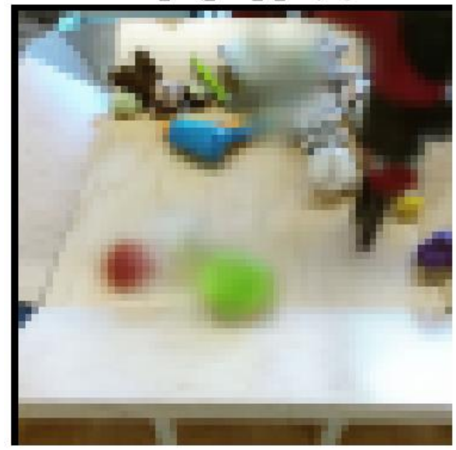
ref_seq_280_0_11.png



rec_seq_280_0_11.png

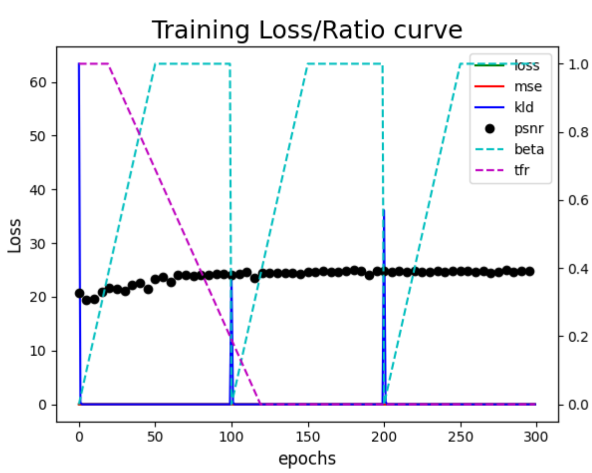


norm_seq_280_0_11.png

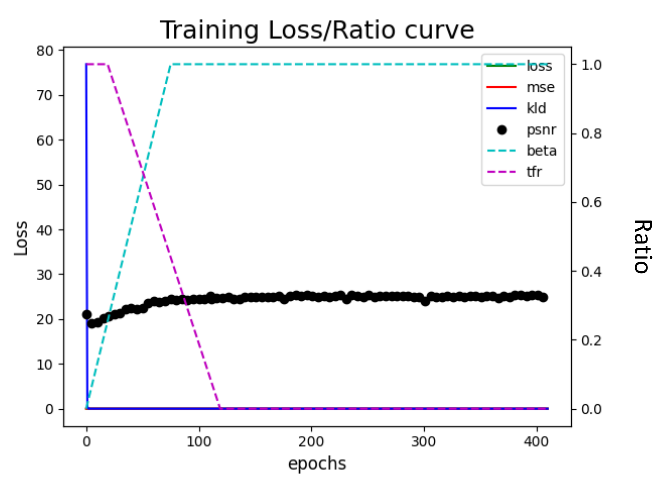


可以看得到 CVAE 可以預測到機械手臂的移動，但是還無法完整的還原背景，這應該是導致 PSNR 上不去的原因。

- Plot the losses, average PSNR and ratios. (5%)
  Parameter setting:
  batch_size=12, beta=0.0001, beta1=0.9, cond_size=7, cuda=True, data_root='./data/processed_data', epoch_size=600, g_dim=128, kl_anneal_cycle=4, kl_anneal_cyclical=True, kl_anneal_ratio=2, kl_beta=0, last_frame_skip=False, load_model_name='model.pth', log_dir='./logs/fp/, lr=0.002, mean=1.0, model_dir='', n_eval=10, n_future=10, n_past=2, niter=300, num_workers=10, optimizer='adam', posterior_rnn_layers=1, predictor_rnn_layers=2, rnn_size=256, save_model_name='model.pth', seed=1, tfr=1.0, tfr_active=True, tfr_decay_step=0.01, tfr_lower_bound=0, tfr_start_decay_epoch=20, var=0.0, z_dim=64



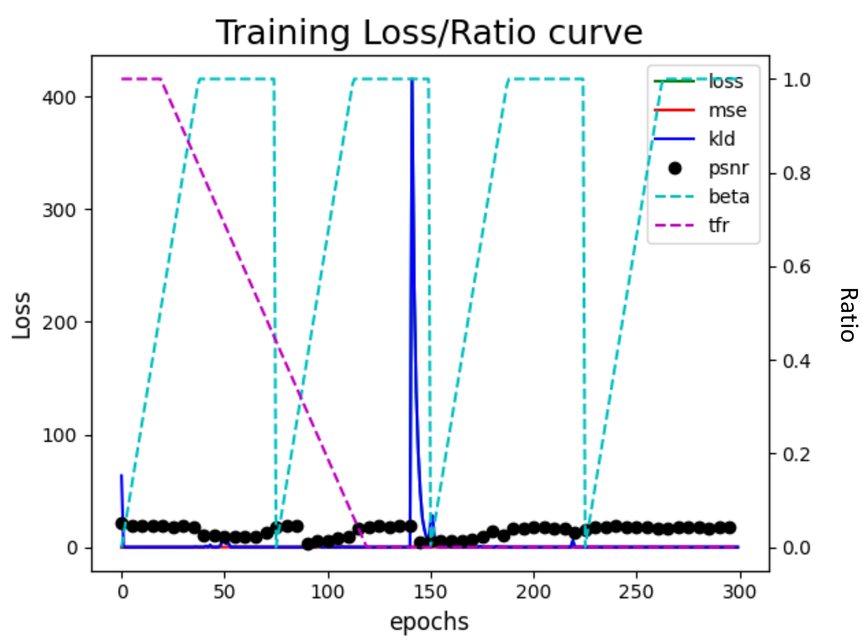
Cyclical



Monotonic

- Discuss the results according to your settings. (15%)
  看得出來一開始 kld loss 會根據 beta 變大而變少，而 tfr 一開始會引導模型學習，讓模型的編碼和解碼器先學習到一定的程度，後來就可以讓模型自行預測，然後再提升預測的準確度，PSNR 都相對平穩慢慢提升。
  執行過 Cyclical 和 Monotonic，發現兩個 PSNR 差別不太大，但 Cyclical 的 beta 從 1 變 0 的那段時間觀察到 kld loss 會有所提升，接著 PSNR 也會在這段時間跳動，感覺上會對模型學習有幫助。

  我嘗試了跑 4 個 Cyclical(如下圖)，但效果不太好，也突然有一段 kld loss 突然很大，就我推斷認為是和抽樣有關，剛好抽樣到很極端的 sample 導致有這樣的結果，然而 4 個 Cyclical 沒有想像中的好，也就沒有再試下去了。



Notice: This part mainly focuses on your discussion, if you simply just paste your results, you will get a low score