DL_LAB2_Report _ Conditional VAE for Video Prediction

學號：312554004　姓名：林垣志

(1) Introduction:

In this lab, the goal is to implement a Conditional Variational Autoencoder (VAE) for the purpose of video prediction. The primary objective is to predict future video frames based on past frames as input. The process involves encoding the previous frame (x_t - 1) to generate a latent code (h_t - 1) using an encoder. A fixed prior distribution is then used to sample a noise variable (z_t). The encoder output, along with the noise variable and the condition (action and position), are fed into the decoder to generate the predicted next frame ($x\hat{}_t$).

There will be 16 frames as a training video sequence {x1, x2, …… , x16}, and 16 label frames as conditional signals {P1, P2, …… , P16}. First frame is the past frame which is provided to predict the consecutive 15 future frames. Example is provided shown below.

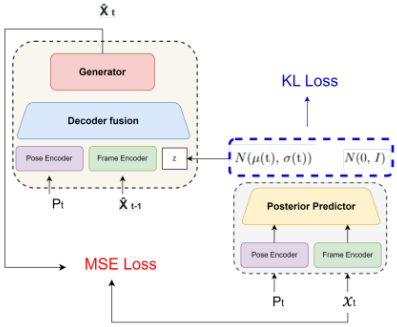


Figure 2. (a)

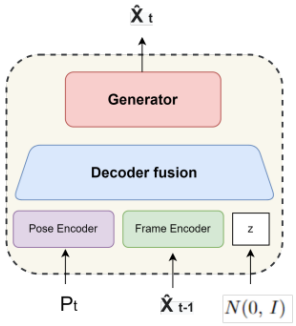


Figure 2. (b)

TA provided dataset in both training, validating, and testing data are shown below.

- Training dataset (same quantity):
  i. train_img & train_label ➔ 23410
- Validating dataset (same quantity):
  i. val_img & val_label ➔ 630
- Testing dataset:
  ⇨ 6 video sequences. Each video sequence contains 1 first frame and 630 label frames.

(2) Implementation details:

(a) Video prediction protocol in (train) stage:

The training protocol involves using a Posterior Predictor (Gaussian Predictor) that takes the current frame which needs to undergo the judgment of adapt_TeacherForcing to determine whether to use the ground truth or the image by using the model generated previous reconstruction, and the label as input to generate a distribution, and a generator that uses the current label, last generated frame, and noise sampled from the distribution to generate the predicted frame.

The figures (code) are shown below that includes the forward process, train & valid process for one step, and the training stage, which have recorded some informations to csv files, and even to calculate the valid PSNR helps me to determinate the best training model can used in testing stage.

```python
def forward(self, img, label, adapt_TeacherForcing):
    # Initialize frame
    reconstructed_img = [img[:,0]]

    for i in range(1, img.size(1)):
        if adapt_TeacherForcing:
            previous_frames = img[:, i-1]  # Use ground truth image for teacher forcing
        else:
            # Generate image using the model's own previous reconstruction
            previous_frames = reconstructed_img[-1]

        # Image transformation
        previous_features = self.frame_transformation(previous_frames)
        ground_truth_features = self.frame_transformation(img[:, i])

        # Label transformation
        label_features = self.label_transformation(label[:, i])

        # Predict latent parameters (sample latent variable)
        z, mu, logvar = self.Gaussian_Predictor(ground_truth_features, label_features)

        # Fusion for decoder input
        decoder_input = self.Decoder_Fusion(previous_features, label_features, z)

        # Generate output
        generated_output = self.Generator(decoder_input)
        reconstructed_img.append(generated_output)

    reconstructed_img = stack(reconstructed_img, dim = 1)

    return reconstructed_img, mu, logvar
```

```python
def training_one_step(self, img, label, adapt_TeacherForcing):
    self.optim.zero_grad()

    # Forward pass
    reconstructed_img, mu, logvar = self.forward(img, label, adapt_TeacherForcing)

    # Calculate reconstruction loss (MSE)
    mse_loss = self.mse_criterion(reconstructed_img, img)

    # Calculate KL divergence
    kl_loss = kl_criterion(mu, logvar, img.size(0))

    # Apply annealed KL weight
    total_loss = mse_loss + self.kl_annealing.get_beta() * kl_loss

    # Backpropagation
    total_loss.backward()
    self.optimizer_step()

    return total_loss
```

```python
def val_one_step(self, img, label):
    # Forward pass
    reconstructed_img, mu, logvar = self.forward(img, label, False)

    # Calculate reconstruction loss (MSE)
    mse_loss = self.mse_criterion(reconstructed_img, img)

    # Calculate KL divergence
    kl_loss = kl_criterion(mu, logvar, img.size(0))

    total_loss = mse_loss + self.kl_annealing.get_beta() * kl_loss

    return total_loss, reconstructed_img.squeeze()
```

```python
def training_stage(self):
    train_Loss, tf_adapt, tfr_arr, beta_record, val_Loss, valid_psnr = [],[],[],[],[],[]
    num = []

    for i in range(self.args.num_epoch):
        tot_loss = 0.0
        num.append(i+1)
        train_loader = self.train_dataloader()
        adapt_TeacherForcing = True if random.random() < self.tfr else False

        for (img, label) in (pbar := tqdm(train_loader, ncols = 120)):

        tot_loss /= len(train_loader.dataset)
        print(f"training loss: ", tot_loss)

        if self.current_epoch % self.args.per_save == 0:

        train_Loss.append(round(tot_loss, 5))
        tf_adapt.append(adapt_TeacherForcing)
        tfr_arr.append(round(self.tfr, 10))
        beta_record.append(beta)

        """ eval stage """
        epoch_psnr, vloss = self.eval()
        val_Loss.append(round(vloss, 5))
        valid_psnr.append(round(epoch_psnr, 4))

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

    df = {'Epoch': num, 'train_loss': train_Loss, 'teacherForce(T/F)': tf_adapt, 'teacher_forcing_ratio': tfr_arr,

    new_df = pd.DataFrame(df)
    new_df.to_csv(self.args.save_root + '\\' + self.args.kl_anneal_type + ".csv", index = False)
```

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

        for i in range(1, len(generated_frame)):
            PSNR = Generate_PSNR(img[0][i], generated_frame[i])
            psnr_list.append(PSNR.item())

    return sum(psnr_list)/(len(psnr_list)-1), float(loss.detach().cpu())
```

(b) Reparameterization tricks:

Utilize the reparameterization trick, ensuring that the output of the trick provides the log variance rather than directly presenting the variance.

```python
def reparameterize(self, mu, logvar):
    std = torch.exp(0.5 * logvar)
    epsilon = torch.randn_like(std)
    latent_z = mu + epsilon * std

    return latent_z
```

(c) Setting teacher forcing strategy:

I let teacher forcing ratio decay linearly (using tfr_sde), so when last epoch, it will decay to teacher forcing lower bound (0.1).

```python
def teacher_forcing_ratio_update(self):
    if self.current_epoch % self.tfr_sde == 0:
        self.tfr = max(0.1, (self.tfr - self.tfr_d_step * 2))
```

(d) Setting kl annealing ratio:

If the “kl_anneal_type” is **cyclical**, the current update of “beta” will be obtained using the 【frange_cycle_linear】function. If the “kl_anneal_type” is **monotonic**, the “beta” updated each time will start from 0.1 and increase to 1.0, with 1.0 being the maximum value for `beta`. If “kl_anneal” is not used, the current “beta” remains set at 1.0.

```python
class kl_annealing():
    def __init__(self, args, current_epoch = 0):

        self.current_epoch = current_epoch
        self.kl_anneal_type = args.kl_anneal_type
        self.beta = 0.0

    def update(self):
        self.current_epoch += 1

        if self.kl_anneal_type == "Monotonic":
            self.beta = min(1.0, self.beta + 1.0 / args.kl_anneal_cycle)

        elif self.kl_anneal_type == 'Cyclical':
            self.beta = self.frange_cycle_linear(args.num_epoch, start = 0.0, stop = 1.0,
                                                 n_cycle = 4.0, ratio = args.kl_anneal_ratio)
        else:
            self.beta = 1.0    # Default, Without KL annealing

    def get_beta(self):
        return self.beta

    def frange_cycle_linear(self, n_iter, start = 0.0, stop = 1.0,  n_cycle = 10, ratio = 1.0):
        period = n_iter / n_cycle
        step = self.current_epoch % period

        return start + step * (stop - start) / (period * ratio) if step < (period * ratio) else stop
```

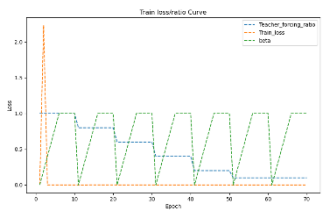
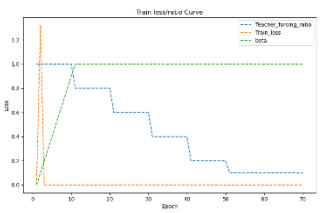
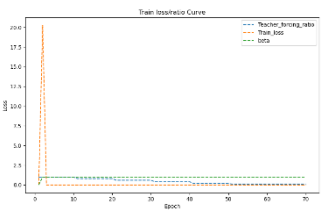
(3) Analysis & Discussion:

(a) Plot Teacher forcing ratio:

The follow hyperparameters are same in every different settings.

lr	Beginning tfr	Epochs	Optimizer	Loss function
0.001~1e-5	1.0	70	Adam	MSE_loss + KL_loss

I need to use our predicted frame to predict next time step frame, but when training the model, it can't let tfr as 0 at the beginning, because predicted input data may have bias or may be too blurred, it will influence model to reconstruct correct frame. To avoid this situation, I set teacher forcing ratio as 1 at first 10 epochs, and every decay decrease 0.1 to lower bound (0.1). This way can let model learn correct reconstruction given ground truth input data at first stage, then learning reconstruction given bias input data. The loss in every different settings can be lower to 0.0005，the detail will be discussed at 3.b.

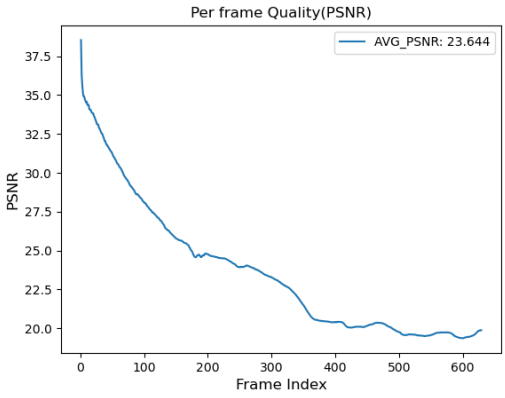
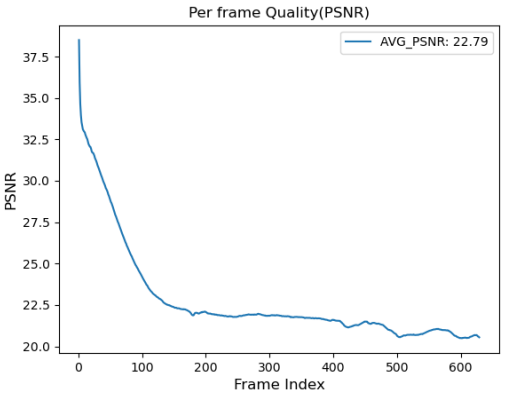
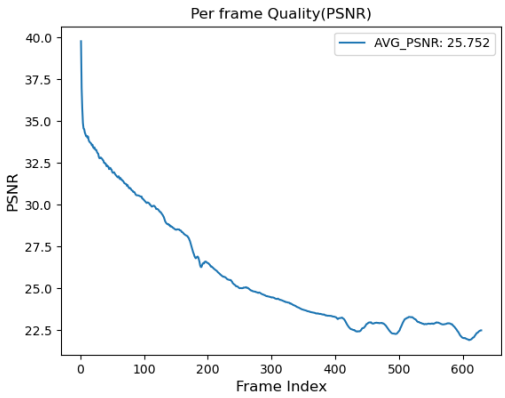
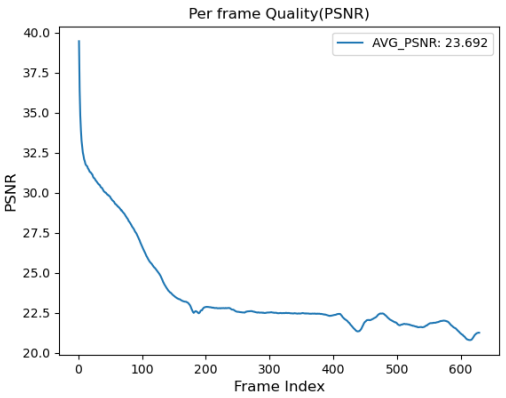
Cyclical	Monotonic	w/o Monotonic & Cyclical
		

(b) Plot the loss curve while training with different settings:

As section 3.a shown the graph and mentions. At the start of the training process, our aim is for the model to prioritize the MSE loss. Additionally, given the significantly higher value of the KL loss (approximately 1000 ~ 300) compared to the MSE loss (approximately 0.02), there is a risk that the overall loss could be dominated by the KL loss component. KL annealing cyclical as finetuning the model cycle-by-cycle, and as we can see in the figure shows, the loss suddenly increase to roughly 2.x in first epoch, but w/o KL annealing the loss will increase to above 20, it's really higher than the others. Even though this KL loss value is considerably higher than that of epoch 1, it remains lower than the initial KL loss value (about 0.0005).

In next section (3.c) will see the PSNR-per frame diagram in validation dataset，and the avg psnr value in testing dataset. It's interesting to note that the validation PSNR (in Cyclical (ratio = 0.5)) is relatively modest, yet during testing, the calculated PSNR spikes to 24. The other observed that use the same model to testing stage to calculated PSNR, the output avg PSNR will have a small different.

(c) Plot the PSNR-per frame diagram in validation dataset:

Cyclical (ratio = 0.5)	Cyclical (ratio = 1.0)
	
Monotonic	**w/o Monotonic & Cyclical**
	

(d) The Avg PSNR (take 6 sequences) in testing dataset:

Cyclical (ratio = 0.5)	Cyclical (ratio = 1.0)
========== Your Result ========== PSNR for testing sequence1: 23.757 PSNR for testing sequence2: 25.796 PSNR for testing sequence3: 21.068 PSNR for testing sequence4: 24.907 PSNR for testing sequence5: 24.793 PSNR for testing sequence6: 23.776 ---------- AVG: 24.016	========== Your Result ========== PSNR for testing sequence1: 22.528 PSNR for testing sequence2: 21.892 PSNR for testing sequence3: 20.925 PSNR for testing sequence4: 22.077 PSNR for testing sequence5: 22.822 PSNR for testing sequence6: 23.379 ---------- AVG: 22.270
Monotonic	**w/o Monotonic & Cyclical**
========== Your Result ========== PSNR for testing sequence1: 24.521 PSNR for testing sequence2: 23.974 PSNR for testing sequence3: 20.848 PSNR for testing sequence4: 21.822 PSNR for testing sequence5: 24.825 PSNR for testing sequence6: 24.210 ---------- AVG: 23.367	========== Your Result ========== PSNR for testing sequence1: 23.241 PSNR for testing sequence2: 20.705 PSNR for testing sequence3: 21.293 PSNR for testing sequence4: 21.835 PSNR for testing sequence5: 22.996 PSNR for testing sequence6: 23.369 ---------- AVG: 22.240

(4) Extra:

(a) Derivate conditional VAE formula:

◎ Derivation of Conditional VAE (refer to L13-LFM. slide 23)

Start → $\log p(x|c;\theta) = \log p(x,z|c;\theta) - \log p(z|x,c;\theta)$

we next introduce an arbitrary distribution $q(z|x,c)$ on both sides and integrate over Z.

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

Note:

$p(x) = \int p(z)p(x|z)dz$

$L = \sum_x \log p(x)$

$\log p(x) = \int q_\phi(z|x)\log p(x)dz$

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

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

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

$$\Rightarrow \int q(z|x,c)\log\left(\frac{p(x,z|c;\theta)}{q(z|x,c)}\right)dz + \int q(z|x,c)\log\left(\frac{q(z|x,c)}{p(z|x,c;\theta)}\right)dz$$

$$\downarrow \qquad\qquad \downarrow$$

$$L_b(x,c,q,\theta) \qquad + KL\left(q(z|x,c)\,||\,p(z|x,c;\theta)\right)$$

☆ 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 if and only if $q(z|x,c) = p(z|x,c;\theta)$

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

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

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

$$= E_{z\sim q(z|x,c;\phi)}\log p(x|z,c;\theta) - KL(q(z|x,c;\phi)\,||\,p(z|c)) \quad *$$