# DLP Lab4 Report
# Conditional VAE for Video Prediction

Name: 丁祐承
ID: 0812212

## 1 Derivation of Conditional VAE

$$Claim:\ \mathcal{L}(X,c,q,\theta) = E_{Z\sim q(Z|X,c;\phi)}\log p(X \mid Z,c;\theta) - \mathrm{KL}(q(Z \mid X,c;\phi)\|p(Z \mid c))$$

Start from EM algorithm, we have

$$p(X,Z \mid c;\theta) = p(Z \mid X,c;\theta)p(X \mid c;\theta)$$
$$\Rightarrow \log p(X,Z \mid c;\theta) = \log p(Z \mid X,c;\theta) + \log p(X \mid c;\theta)$$
$$\Rightarrow \log p(X \mid c;\theta) = \log p(X,Z \mid c;\theta) - \log p(Z \mid X,c;\theta)$$
$$\Rightarrow q(Z \mid c)\log p(X \mid c;\theta) = q(Z \mid c)\log p(X,Z \mid c;\theta) - q(Z \mid c)\log p(Z \mid X,c;\theta)$$

$$\begin{aligned}\Rightarrow \int q(Z \mid c)\log p(X \mid c;\theta)dZ &= \int q(Z \mid c)\log p(X,Z \mid c;\theta)dZ - \int q(Z \mid c)\log p(Z \mid X,c;\theta)dZ \\ &= \int q(Z \mid c)\log p(X,Z \mid c;\theta)dZ - \int q(Z \mid c)\log q(Z \mid c)dZ \\ &+ \int q(Z \mid c)\log q(Z \mid c)dZ - \int q(Z \mid c)\log p(Z \mid X,c;\theta)dZ \\ &= \mathcal{L}(X,c,q,\theta) + \int q(Z \mid c)\frac{\log q(Z \mid c)}{\log p(Z \mid X,c;\theta)}dZ \\ &= \mathcal{L}(X,c,q,\theta) + \mathrm{KL}(q(Z \mid c)\|p(Z \mid X,c;\theta)\end{aligned}$$

Substitute every $q(Z \mid c)$ with $q(Z \mid X,c;\phi)$, we have

$$\Rightarrow \mathcal{L}(X,c,q,\theta) = \log p(X \mid c;\theta) - \mathrm{KL}(q(Z \mid X,c;\phi)\|p(Z \mid X,c;\theta))$$

For $\mathcal{L}(X,c,p;\theta)$, we have

$$\begin{aligned}\mathcal{L}(X,c,p;\theta) &= \int q(Z \mid X,c;\phi)\log p(X,Z \mid c;\theta)dZ - \int q(Z \mid X,c;\phi)\log q(Z \mid X,c;\phi)dZ \\ &= E_{Z\sim q(Z|X,c;\phi)}\log p(X,Z \mid c;\theta) - E_{Z\sim q(Z|X,c;\phi)}\log q(Z \mid X,c;\phi) \\ &= E_{Z\sim q(Z|X,c;\phi)}[\log p(X \mid Z,c;\theta) + \log p(Z \mid c)] - E_{Z\sim q(Z|X,c;\phi)}\log q(Z \mid X,c;\phi) \\ &= E_{Z\sim q(Z|X,c;\phi)}\log p(X \mid Z,c;\theta) + E_{Z\sim q(Z|X,c;\phi)}\log p(Z \mid c)) - E_{Z\sim q(Z|X,c;\phi)}\log q(Z \mid X,c;\phi) \\ &= E_{Z\sim q(Z|X,c;\phi)}\log p(X \mid Z,c;\theta) - \mathrm{KL}(q(Z \mid X,c;\phi)\|p(Z \mid c))l\end{aligned}$$

Thus we obtain Conditional VAE

$$\mathcal{L}(X,c,q,\theta) = E_{Z\sim q(Z|X,c;\phi)}\log p(X \mid Z,c;\theta) - \mathrm{KL}(q(Z \mid X,c;\phi)\|p(Z \mid c))$$

## 2 Introduction

In this lab, our task is to implement the video prediction using VAE-based model. The dataset we use is the pictures of a people doing different poses. There are two main parts of the dataset, the first part is the frame and the second part is the pose. Frames are used as source data for model and poses are used as label for model. For training, the main idea is to use the last generated frame as source to generate next frame using pose as constraint. For inference, the main idea is to use one source frame and rest of the poses to generate a video. The following figure is the training and inference architecture.

## 2.1 Model Architecture

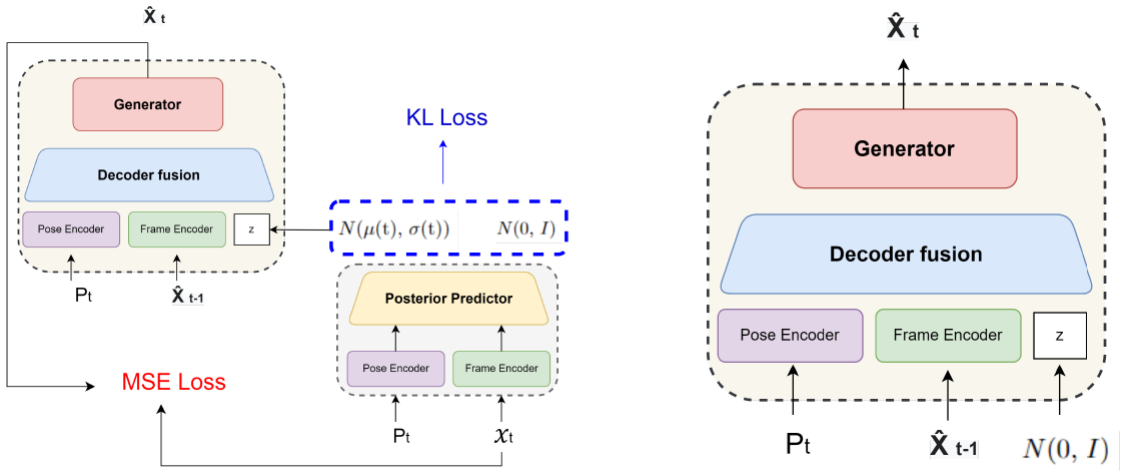



Figure 1: Model Architecture

## 2.2 Dataset

- Training dataset contains 23410 images and 23410 poses
- Validation dataset contains 630 images and 630 poses
- Testing dataset contains 5 video sequences with each contains one first frame and 630 poses

(a)

(b)

Figure 2: Frame (a), Pose (b) Example

# 3 Implementation details

## 3.1 How do you write your training protocol

For training, since the task is to do the video prediction, so the input of the model will be frame by frame. Briefly saying, if I set the video length as 16, then the main task is to predict the last 15 frames of the video. For example, we do not need to predict the first frame, so we start from predicting the second frame. We use $X_t$ to represent the frame of step $t$ and $P_t$ to represent the pose of step $t$. For one of the 15 training steps, we took $X_t$ and $P_t$ as input of the Encoder and generate the latent distribution. After reparameterization we have $z$, we use the $P_t$, $X_{t-1}$ which is generated from the last step and $z$ as inputs of the Decoder to generate the present frame. By repeating the 15 times, we will have the complete sequence of the predicted video. The following figures are the main idea of the training protocol and the actual implematation code.

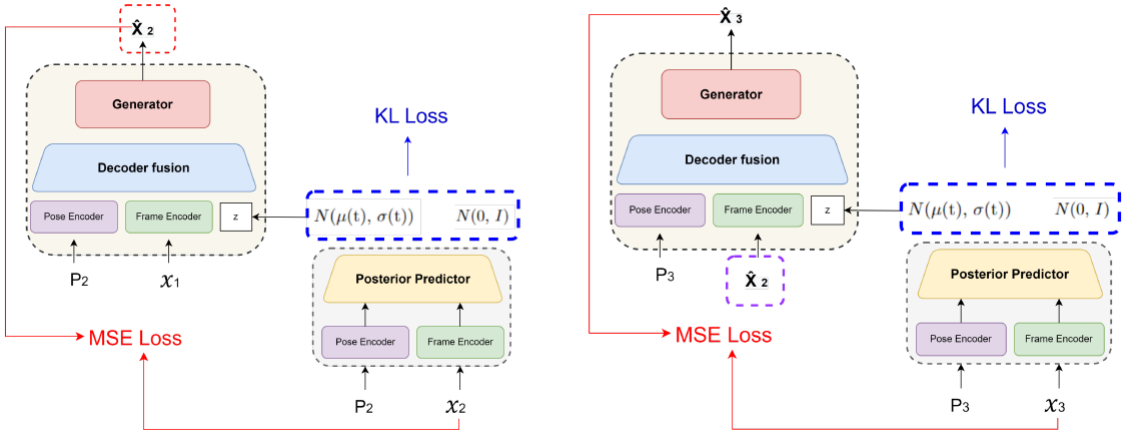



Figure 3: Training Protocol

```python
def training_one_step(self, img, label, adapt_TeacherForcing=0):
    self.optim.zero_grad()
    # Changing dimension -> (seq, batch, channel, height, width)
    img = img.permute(1, 0, 2, 3, 4)
    label = label.permute(1, 0, 2, 3, 4)

    kl_divergence = 0
    mse_loss = 0
    total_loss = 0
    pred_frame = [img[0]]
    beta = self.kl_annealing.get_beta()

    for frame in range(1, self.train_vi_len):
        img_en = self.frame_transformation(img[frame])
        label_en = self.label_transformation(label[frame])
        z, mu, logvar = self.Gaussian_Predictor(img_en, label_en)

        if adapt_TeacherForcing:
            x = img[frame - 1]
        else:
            x = pred_frame[-1]
        x = self.frame_transformation(x)
        decoder_output = self.Decoder_Fusion(x, label_en, z)
        pred = self.Generator(decoder_output)

        # Compute loss
        kl_divergence += kl_criterion(mu, logvar, self.batch_size)
        mse_loss += self.mse_criterion(pred, img[frame])

        pred_frame.append(pred)

    # Back propagation
    total_loss = mse_loss + beta * kl_divergence
    total_loss.backward()
    self.optimizer_step()

    return mse_loss, kl_divergence
```

### 3.2 How do you implementation reparameterization tricks

Before reparameterization trick, the Encoder encodes the input into a Normal distribution $N(\mu, \sigma^2)$ and we sample from it to use it as the input of the Decoder. Since the sampling process is not differentiable, so we introduce the reparameterization trick. Instead of sampling from the $N(\mu, \sigma^2)$, we sample a random noise $\epsilon$ from $N(0, 1)$ and let $z = \mu + \epsilon * \sigma$ as input of the Decoder. Therefore, we fullfill the end to end training for VAE model.

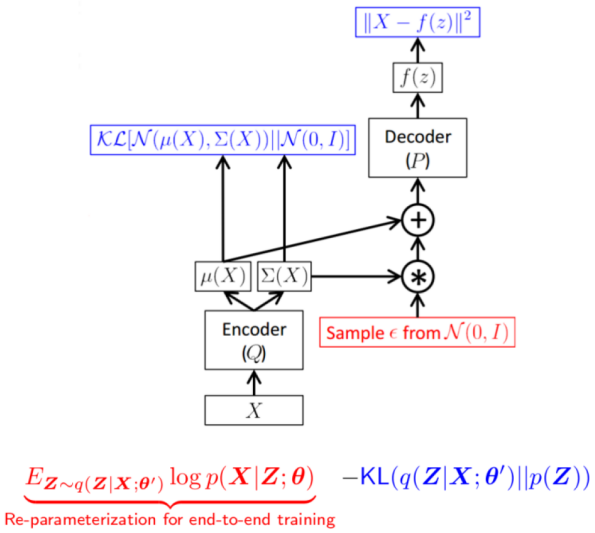



Figure 4: Reparameterization

```
class Gaussian_Predictor(nn.Sequential):
    def __init__(self, in_channel, out_channel):
        pass

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

### 3.3 How do you set your teacher forcing strategy

At the first stage of the training, model might not learn well so that the prediction will go very bad. To solve the problem, we introduce the teacher forcing strategy to prevent this condition. There will be a ratio for teacher forcing, if the teacher forcing is on, the input of the model will be the ground truth of the frame instead of the predicted frame from last step. If the teacher forcing is off, the input of the model will be the predicted frame from last step.

```
if adapt_TeacherForcing:
    x = img[frame - 1]
else:
    x = pred_frame[-1]
```

### 3.4 How do you set your kl annealing ratio

The total loss of the model is the sum of reconstruction and the regularization. For reconstruction is the MSE loss and for regularization is the KL divergence. What KL annealing does is to introduce a variable *beta* to the regularization term. There are two KL annealing ratio. The first one is the

Cyclical and the other one is the Monotonic. For Cyclical, the beta goes up and down as a cycle and for monotonic, it just linearly goes up.

$$\mathcal{L}_\beta = \mathcal{L}_E + \beta\mathcal{L}_R, \text{ with}$$
$$\mathcal{L}_E = -\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$$
$$\mathcal{L}_R = \mathrm{KL}(q_\phi(z|x)\|p(z))$$

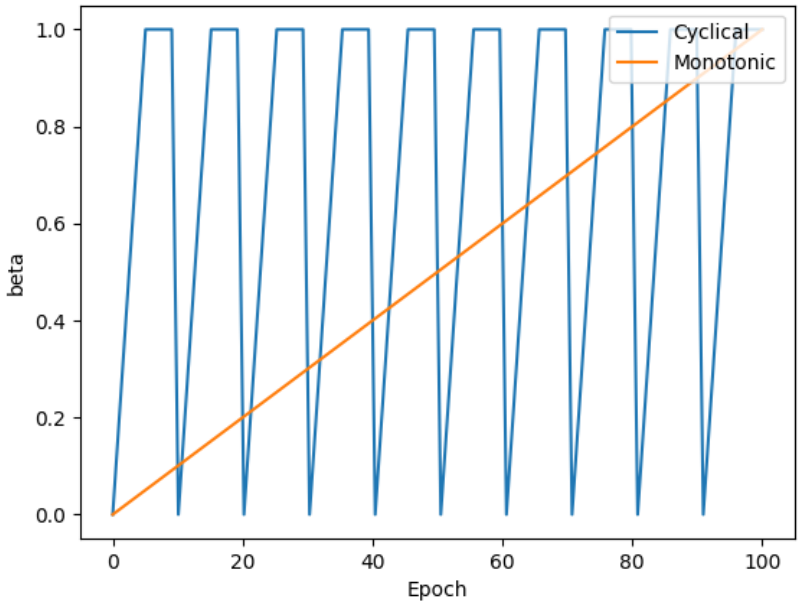
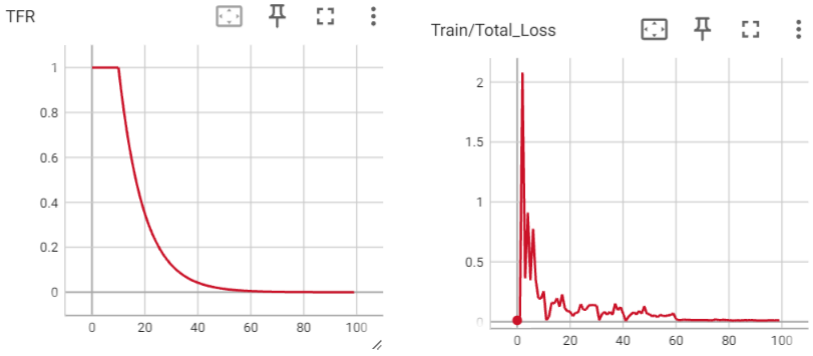



Figure 5: Cyclical and Monotonic KL annealing type

# 4 Analysis and Discussion

## 4.1 Plot Teacher forcing ratio

- Analysis and compare with the loss curve



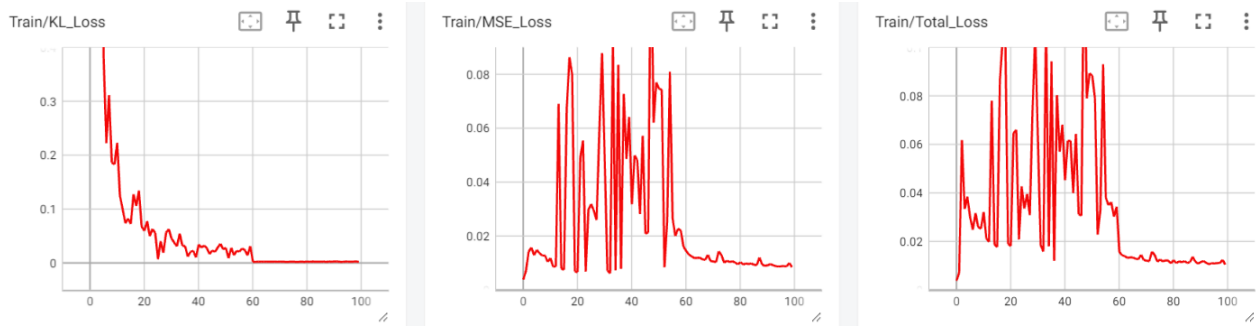

(a) Teacher forcing ratio (b) Total loss

Figure 6: Comparison between Teacher forcing ration and Total loss

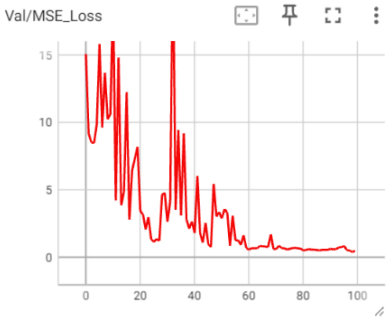
As the teacher forcing ratio goes down, the loss goes down. By providing the ground truth during training, the model converges faster because it does not have to learn from the wrong prediction. It also helps in stabilizing the training process by reducing the variance in predictions during the early training epochs. However, teacher forcing cannot step in too much or the model will not have a good generality.

## 4.2 Plot the loss curve while training with different settings

We can very easily see that if we apply Monotonic or even without KL annealing strategy, the total loss of the VAE model will not converge until around 60 epochs. For Cyclical type of KL annealing strategy, the VAE model will converge considerably fast at around 20 epoch. For KL divergence, both Monotonic and Cyclical type of KL annealing technique makes the KL divergence converge faster than without KL annealing technique. By doing so, it helps the model to first learn to reconstruct the frame, which can stabilize the early phases of training. It also can lead to better overall performance by preventing the KL divergence term from overwhelming the reconstruction loss at the beginning of training. However, it may takes longer time to train the model to converge. nditemize

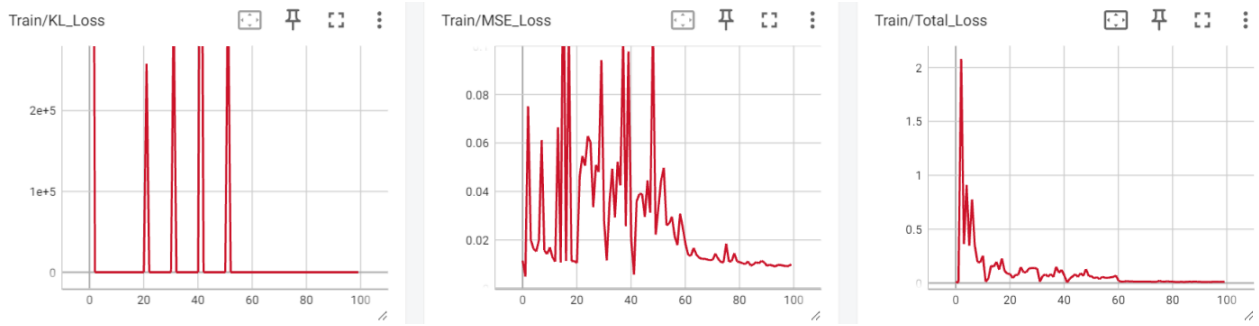



(a) Training Loss: KL Divergence (left) MSE Loss (mid) Total Loss (right)

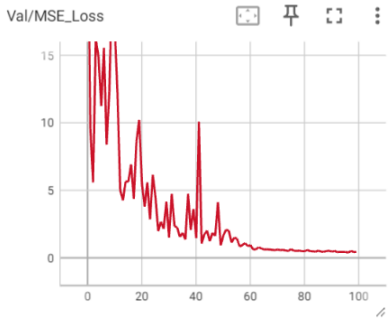



(b) Validation Loss: MSE Loss

Figure 7: With KL annealing (Monotonic)

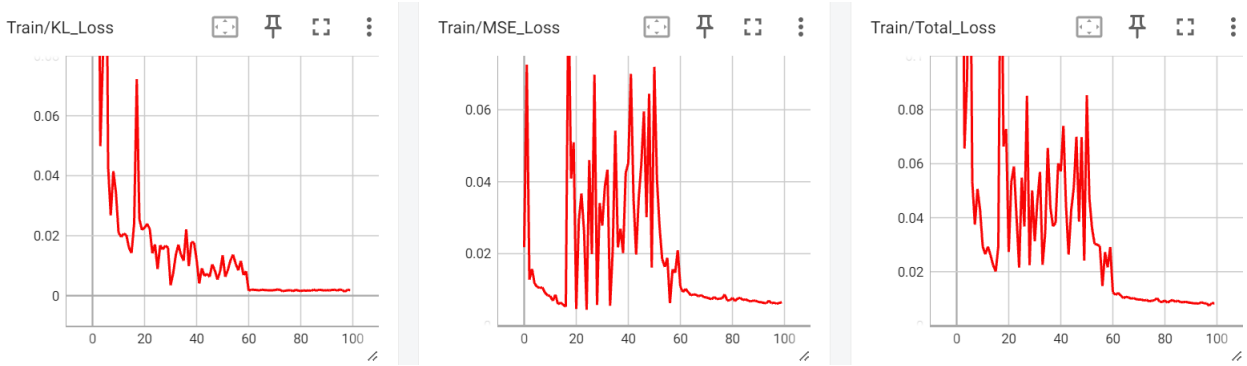



(a) Training Loss: KL Divergence (left) MSE Loss (mid) Total Loss (right)

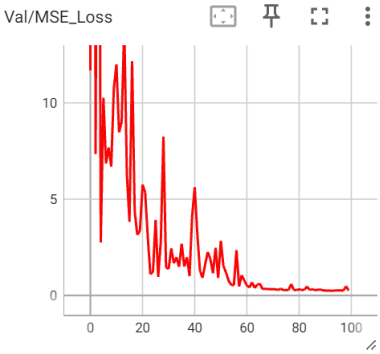



(b) Validation Loss: MSE Loss

Figure 8: With KL annealing (Cyclical)




(a) Training Loss: KL Divergence (left) MSE Loss (mid) Total Loss (right)



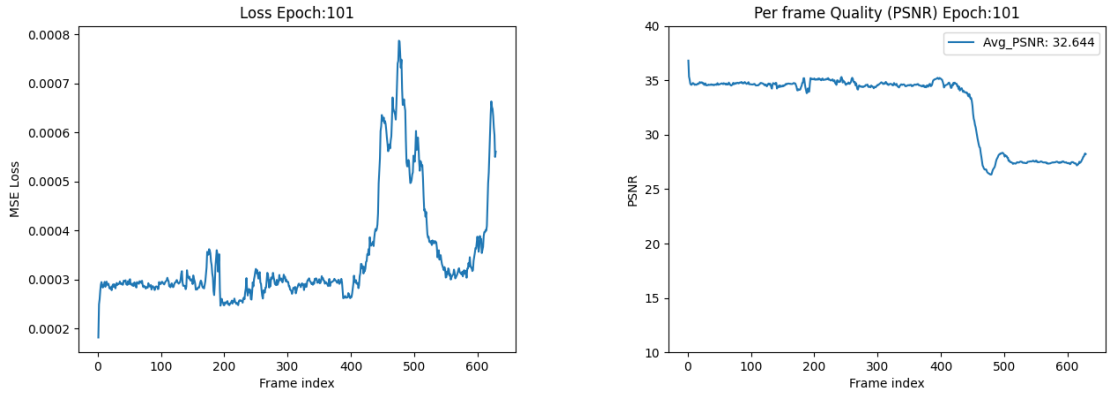

(b) Validation Loss: MSE Loss

Figure 9: Without KL annealing

### 4.3 Plot the PSNR-per frame diagram in validation dataset

Peak signal-to-noise ratio, also known as PSNR is a metric for quantify reconstruction quality for images and video subject to lossy compression.

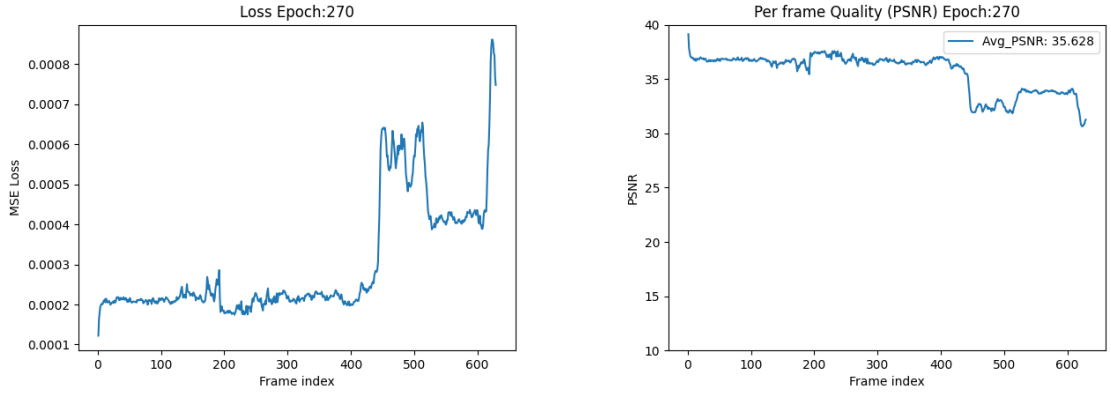
$$PSNR = 10 \cdot \log_{10}(\frac{MAX_I^2}{MSE}) = 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE)$$




(a) (b)

Figure 10: Loss per frame (left), PSNR per frame (right) of 101 Epoch

We can clearly see that the loss and PSNR curves per frame of the validation dataset is relative. As the loss value went up as frame index is higher, which means at the mid stage of the prediction, the PSNR went down. This shows that the prediction did well at the first stage of the prediction, which is around 400 frames. After 400 frames, the quality of the prediction started to go bad and the loss value goes up and PSNR goes down.




(a) (b)

Figure 11: Loss per frame (left), PSNR per frame (right) of 270 Epoch

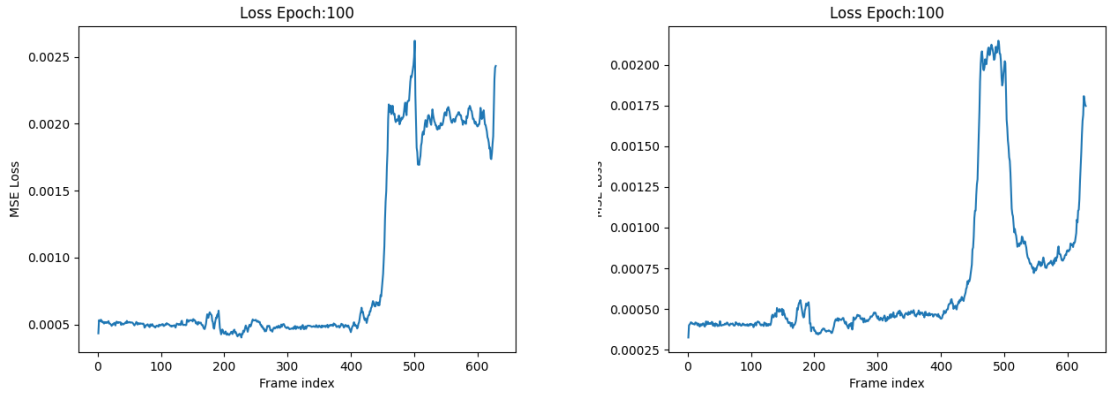
As we can see the Figure 11, after 270 epochs of training, the loss curve around 450 500 frames has a significant improved. For 101 epoch, the loss value around 450 500 was very high and for 270 epoch, it went flat a lot compared with the 101 epoch. For PSNR per frame, we can see the increasement of the PSNR from 32,644 to 35.628.

## 4.4 Other training strategy analysis

When training model, I gave the loss a weight every 4 frame to change the percentage of each frame's loss of the total loss.

```
def training_one_step(self, img, label, adapt_TeacherForcing=0):
    self.optim.zero_grad()

    .
    .
    .

    for frame in range(1, self.train_vi_len):

        .
        .
        .

        # Compute loss
        kl_divergence += kl_criterion(mu, logvar, self.batch_size)
        mse_loss += self.mse_criterion(pred, img[frame])
        if frame % 4 == 0:
            total_loss += weight_ratio * (self.mse_criterion(pred, img[frame])
                ↪  + beta * kl_criterion(mu, logvar, self.batch_size))
            weight_ratio *= weight_ratio
        else:
            total_loss += self.mse_criterion(pred, img[frame]) + beta *
                ↪ kl_criterion(mu, logvar, self.batch_size)
        pred_frame.append(pred)

    # Back propagation
    total_loss.backward()
    self.optimizer_step()

    return mse_loss, kl_divergence
```

Since we are dealing with the video prediction task and as frame goes bigger, the quality of prediction will go bad. So I manually adjust the proportion of each frame's loss to check if the result is better. However, the result seems to be quite the same or even worse.




(a) (b)

Figure 12: Without weight (left), With weight (right)