A. Introduction

本次實驗實作 CVAE，和一般 VAE 不同的是，CVAE 可以給 decoder 其他額外資訊來生成特定圖片，本次實驗及透過給 decoder 過去的資訊來生成現在的圖片。而在訓練上則有 Techer forcing 和 KL annealing 等方法來實現，並以 PSNR score 當作評分標準。

B.Implementation detail

i. training/testing protocol

```
def forward(self, img, pre_img, label, batch_size):
    pre_img_vector = self.frame_transformation(pre_img)
    label_vector = self.label_transformation(label)
    z, mu, sig = self.Gaussian_Predictor(pre_img, label_vector)
    re_gen_img = self.Generator(self.Decoder_Fusion(pre_img_vector, label_vector, z))
    mse_loss = self.mse_criterion(re_gen_img, img)
    kl_loss = kl_criterion(mu, sig, batch_size)
    return mse_loss, kl_loss, re_gen_img
```

forward 當中先將 input 轉換並丟進 Gaussian Predictor，得到這些 input 在 latent space 的分布情況(mu, sig)，接著只要在這個分布當中抽一個點，並加上 condition(pre_img_vector, label_vector) 丟給 decoder 和 Generator，產生模型輸出，而 mse_loss 是模型輸出和實際的差異，kl_loss 則是 latent space 分布和常態分佈的差異。

```python
def training_one_step(self, img, label, adapt_TeacherForcing):
    self.train()
    beta = self.kl_annealing.get_beta()
    mse_loss, kl_loss = 0, 0
    batch_size = img.shape[0]
    frames = img.shape[1]
    x_hat = img[:, 0]
    for f in range(1, frames):
        now_img = img[:, f]
        pre_img = img[:, f-1] if adapt_TeacherForcing else x_hat.detach()
        loss_temp, kl_loss_temp, re_gen_img = self(now_img, pre_img, label[:, f], batch_size)
        mse_loss += loss_temp
        kl_loss += kl_loss_temp
        x_hat = re_gen_img
    loss = mse_loss + beta * kl_loss
    self.optim.zero_grad()
    loss.backward()
    self.optimizer_step()
    return loss.detach()
```

training 當中以第 0 幀當起點，丟給模型 forward 回傳 mse_loss 和 kl_loss 以及第 1 幀的 predict，反覆循環直到所有幀被預測完，之後再做反向傳遞，Techer forcing 是判斷下一幀的輸入是否要是模型上一幀的輸出還是實際幀。

```python
def val_one_step(self, img, label):
    self.eval()
    assert img.shape[0] == 1, "Batch size should be 1 in validation stage"
    batch_size = img.shape[0]
    frames = img.shape[1]
    beta = self.kl_annealing.get_beta()
    mse_loss, kl_loss = 0, 0
    x_hat = img[:, 0]
    gif = [img[0, 0].permute(1,2,0).detach().cpu().numpy()]
    score = []
    for f in range(1, frames):
        now_img = img[:, f]
        pre_img = x_hat.detach()
        loss_temp, kl_loss_temp, re_gen_img = self(now_img, pre_img, label[:, f], batch_size)
        mse_loss += loss_temp
        kl_loss += kl_loss_temp
        x_hat = re_gen_img
        gif.append(re_gen_img[0].permute(1,2,0).detach().cpu().numpy())
        score.append(Generate_PSNR(x_hat, img[:, f]).item())
    self.make_gif(gif, "val.gif")
    # loss = mse_loss + beta * kl_loss
    loss = mse_loss
    return loss.detach(), np.mean(score)
```

testing 當中和 training 類似，差異在為了評估模型

的能力，需要強制使用模型上一幀的輸出當作下一幀的輸入，而圖片中 loss 的計算只有 mse loss，其意思代表評估方式最終看重的是模型輸出的預測能力而非考慮 latent space 的分布。

ii. reparameterization tricks

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

先前提到要在 latent space 中抽一個點，但如果真的隨機抽樣的話會無法做反向傳播(back propagation 會斷在此處)，因此透過 mu + sig * eps 的方式，eps 是一個常態分布，整體代表要選一個點而這個點會偏離以 mu 為中心 eps 的標準差，這就和在 N(mu, sig)中抽樣行為類似，且此方法也可微分做反向傳播。

iii. teacher forcing strategy

```
def teacher_forcing_ratio_update(self):
    if self.current_epoch < self.tfr_sde:
        return
    self.tfr = max(0, self.tfr - self.tfr_d_step)
```

在一定 epoch 之前保證 tfr 為預設值(1.0)，之後每

一次 epoch 衰減一定值。

iv. kl annealing ratio

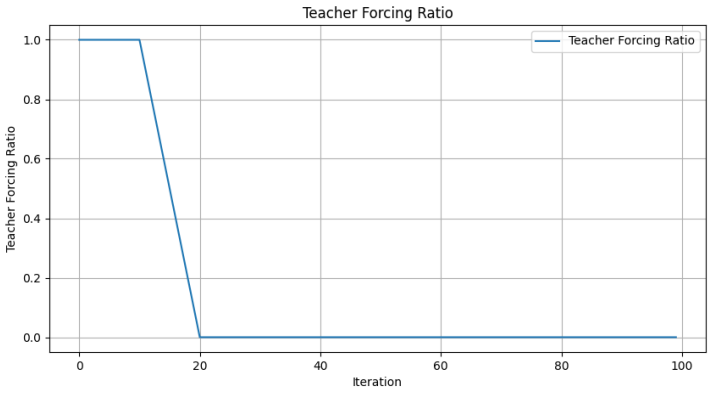
```python
class kl_annealing():
    def __init__(self, args, current_epoch=0):
        self.iteration = current_epoch
        self.args = args
        if args.kl_anneal_type == "Cyclical":
            self.frange_cycle_linear(args.num_epoch, start=0.0, stop=1.0, n_cycle=args.kl_anneal_cycle, ratio=args.kl_anneal_ratio)
        elif args.kl_anneal_type == "Monotonic":
            self.frange_cycle_linear(args.num_epoch, start=0.0, stop=1.0, n_cycle=args.num_epoch, ratio=args.kl_anneal_ratio)

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

    def get_beta(self):
        if self.args.kl_anneal_type == "None": return 1.0
        return round(self.beta[self.iteration], 2)

    def frange_cycle_linear(self, n_iter, start=0.0, stop=1.0,  n_cycle=1, ratio=1):
        self.beta = []
        cycle_num = n_iter // n_cycle
        for i in range(cycle_num+1):
            for j in range(n_cycle):
                val = start + (stop - start) * ratio * (j+1)
                self.beta.append(min(stop, val))
```

如果是 Cyclical，則 beta 週期性的從 start->stop，

如果是 Monotonic，則週期為 1 的版本，如果 beta 在遞增的途中超過 stop，則會停留在 stop 不動。

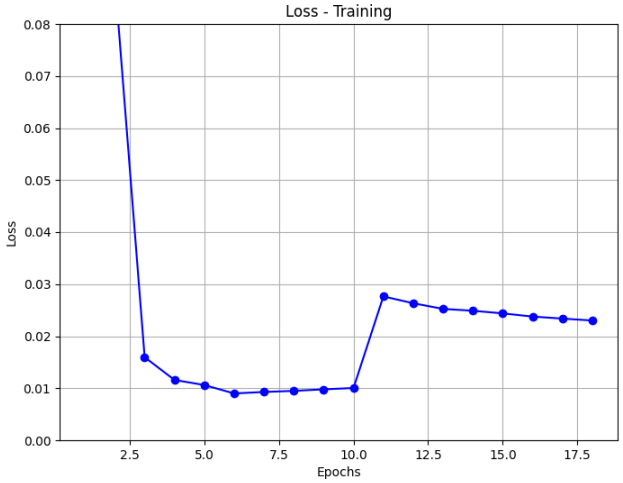
C.Analysis and Discussion

i. teacher forcing ratio

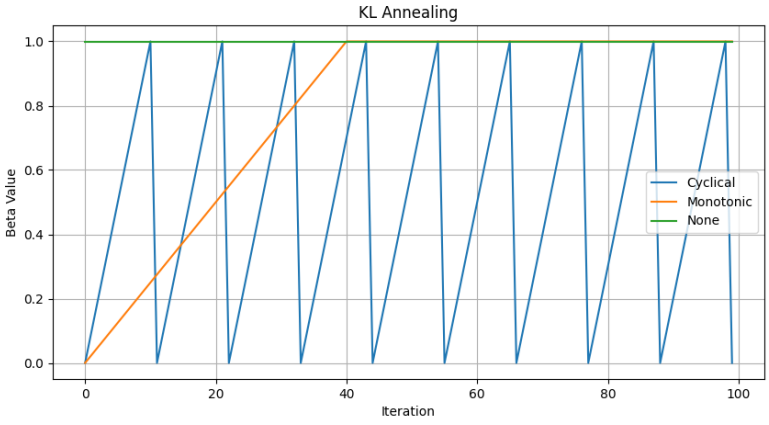


tfr 的幫助能夠讓模型很好的學習下一幀的圖片，

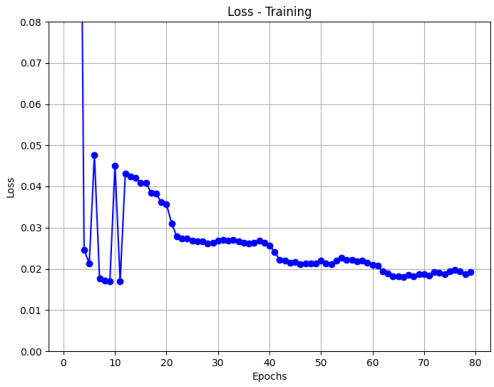
但是仍有缺點，如下圖所示，training loss 在第 10 個 epochs 的時候突然上升，這說明即使對模型學習下一幀圖面的生成能力有幫助，但卻沒有在每一幀跌代時將部分錯誤導正的能力，也就是如果模型在某一幀開始偏離 ground truth，模型接下來沒有能力彌補錯誤。

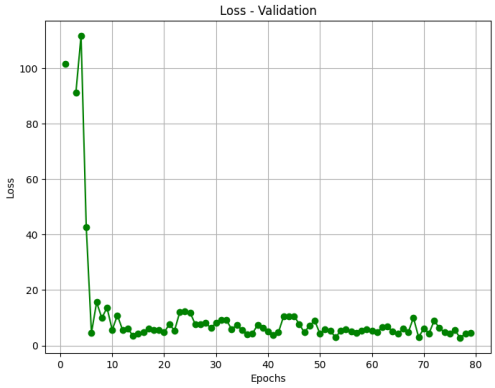


ii. KL annealing

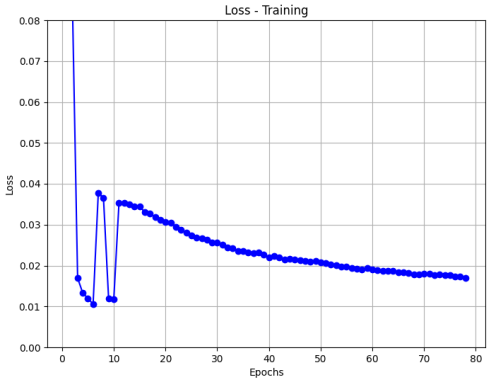


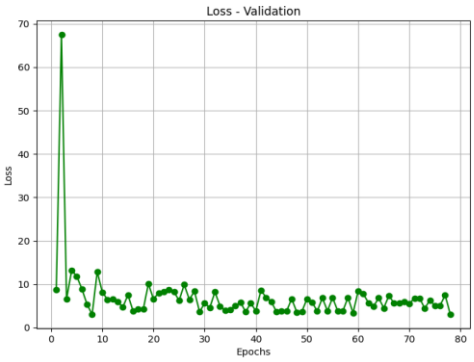
I. Cyclical



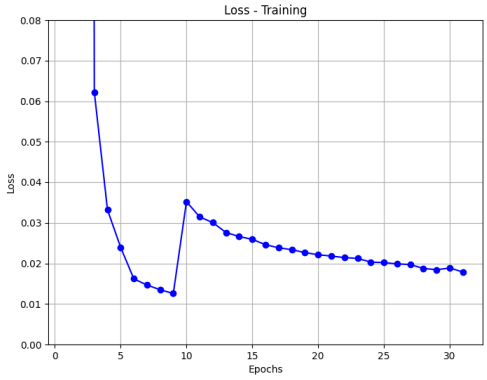


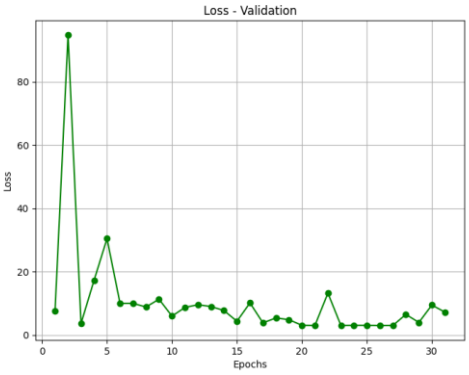
II. Monotonic





III. None



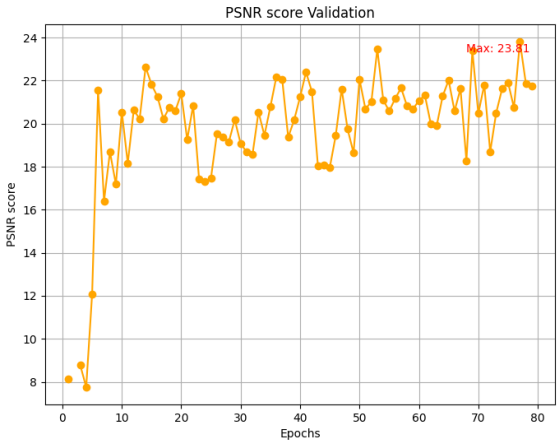


IV. Little Conclusion

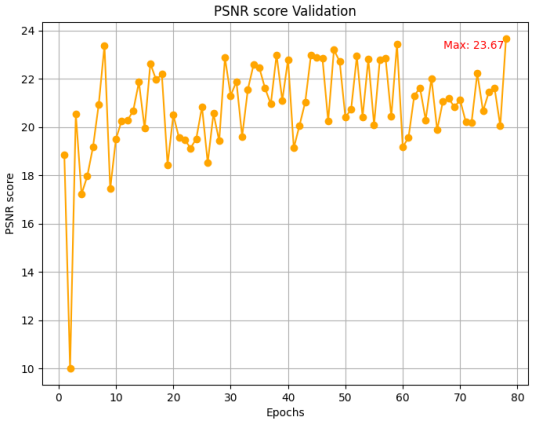
實驗顯示，使用不同方式的 KL annealing，對於最終訓練成果的影響不大，但是相比之下，使用 Cyclical 和 Monotonic 的 loss 曲線相對穩定。

iii. PSNR

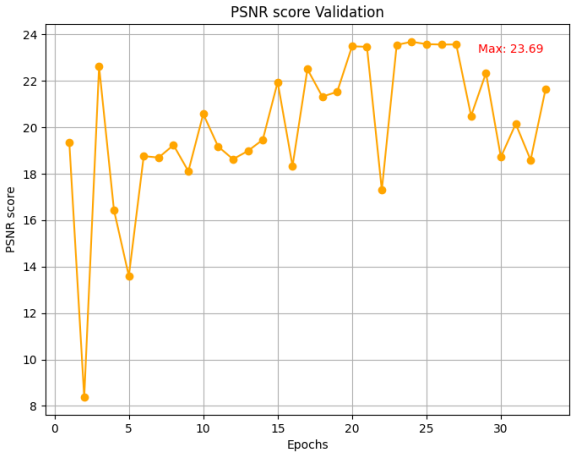
I. Cyclical



II. Monotonic



III. None

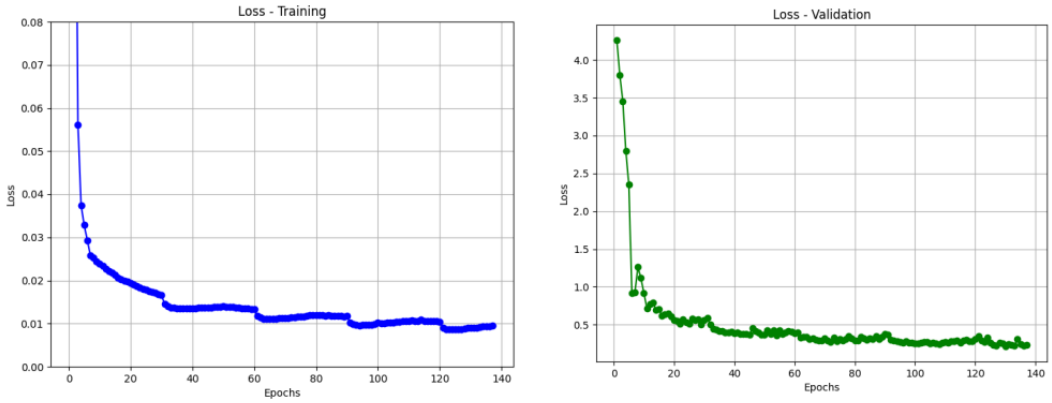


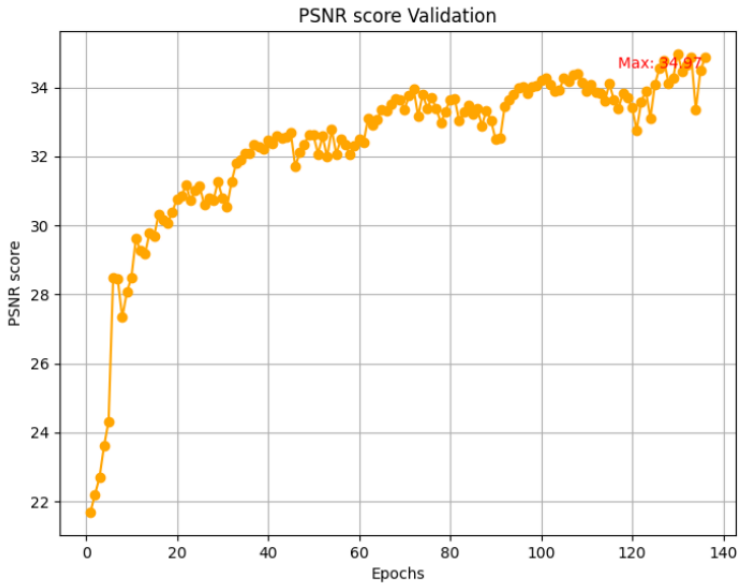
iv. Other training strategy analysis

I. 將 tfr 設為 0

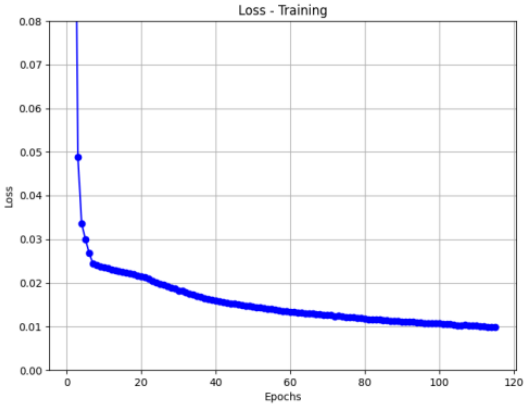
既然 tfr 有可能會讓模型失去導正錯誤的能力，在此就不使用 tfr 的方法。結果顯示，如果將 tfr 移除，則不管使用哪一種 KL annealing 的方式，所有的模型成效皆會**大幅提升**。

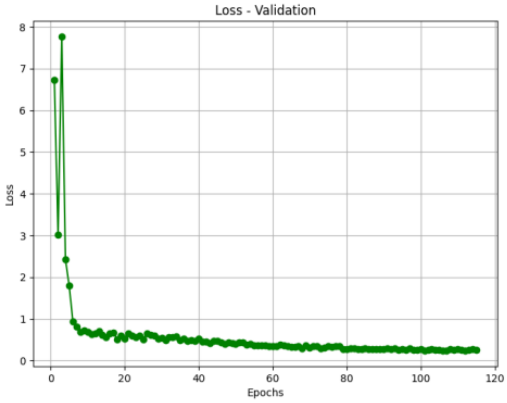
Cyclical:

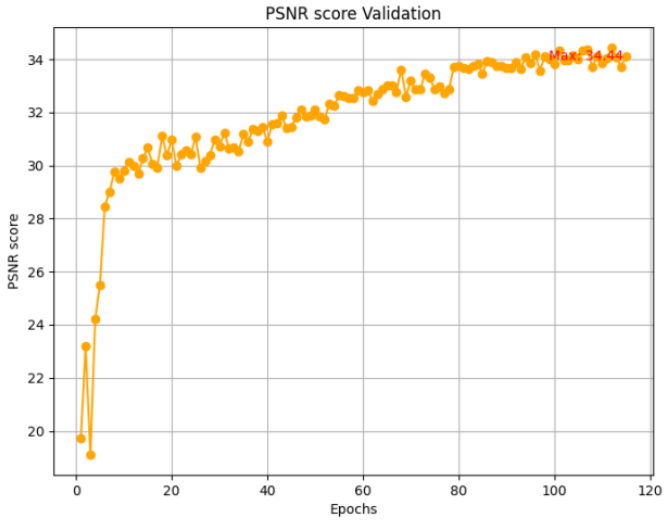




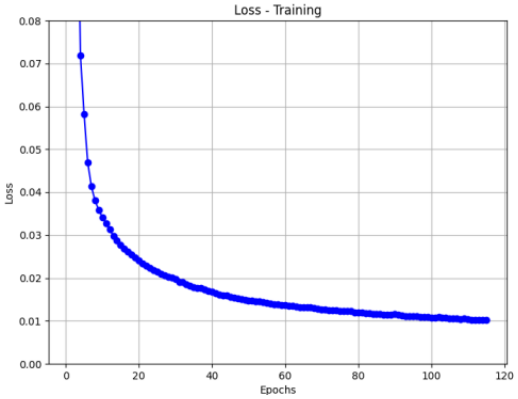
Monotonic:

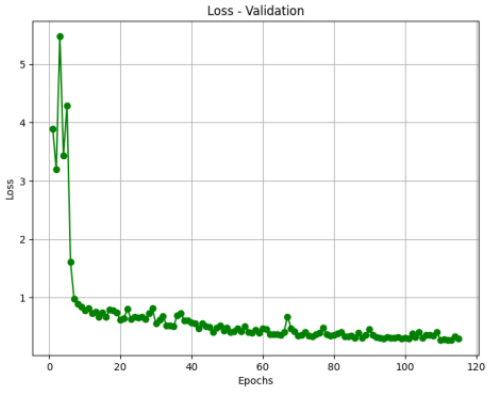


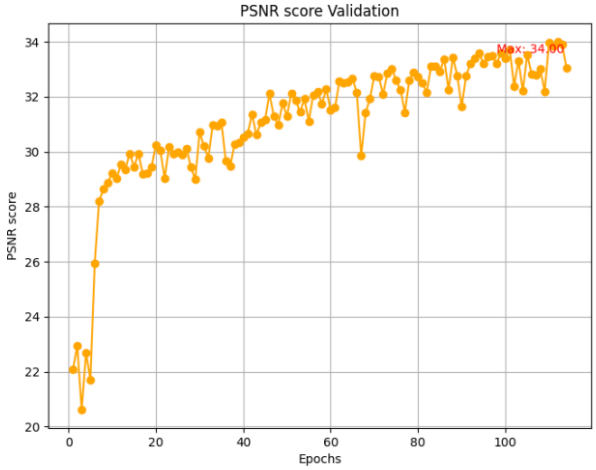




None:







II. Pretrain

```python
def pretrain_one_step_decoder(self, img, label):
    self.train()
    mse_loss = 0
    frames = img.shape[1]
    for f in range(1, frames):
        now_img = img[:, f]
        pre_img = img[:, f-1]
        pre_img_vector = self.frame_transformation(pre_img)
        label_vector = self.label_transformation(label[:, f])
        z = torch.randn(pre_img_vector.shape[0], self.args.N_dim, pre_img_vector.shape[2], pre_img_vector.shape[3]).to(self.args.device)
        re_gen_img = self.Generator(self.Decoder_Fusion(pre_img_vector, label_vector, z))
        mse_loss += self.mse_criterion(re_gen_img, now_img)
    self.optim.zero_grad()
    mse_loss.backward()
    self.optimizer_step()
    return mse_loss.detach()
```

這邊使用了其他方法，為了避免在訓練初期，encoder 所產出的 output 品質不好，連帶影響到 decoder，因此先把 decoder 訓練好，在 pretrain 當中假設 encoder 已經是有能力區分不同圖片的 latent space 分布，且其分布接近標準常態分布的情況下去訓練的。

D. execute code

i. Training

程式會把所模型和所有分析存在 save_root

python ./Lab4_template/Trainer.py --save_root ./save_path

ii. Testing

python ./Lab4_template/Tester.py --save_root ./save_path --ckpt_path ./save_path/best_score.ckpt