- Problem 1

  雖然 Problem 1 是將環境架好並且執行指令就應該要可以正常訓練的，但是因為 code 裡面關於 filter ratio 以及 frame number 的參數，有一些地方是直接設成定值，導致參數在 terminal 輸入進去的時候也沒有辦法更改到，使得 detect object 的數量不一樣，而且 bug 的訊息很難察覺出是關於這兩個參數的問題，讓第一個問題反而是花最久時間的。

  以下是結果：

```
Load 5 videos 5527 frames, 21558 objects, excluding 244 inside objects and 0 small objects.
(smoothing: True): dataset = avenue, auc = 0.8694396519224543  aver_result: [0.8376166549468416]
(smoothing: True): dataset = avenue, auc = 0.8975648321040329  aver_result: [0.9114282819628279]
(smoothing: True): dataset = avenue, auc = 0.8920493908199426  aver_result: [0.8924235151355644]
cur max: 0.9308253535994182 in 2024-05-23-10-59-19
[19:93/233]     loss: 0.809623 t_loss: 0.480788 s_loss: 0.328835        time: 79.774110
[19:113/233]    loss: 0.845328 t_loss: 0.486121 s_loss: 0.359207        time: 12.926501
[19:133/233]    loss: 0.799119 t_loss: 0.424996 s_loss: 0.374123        time: 12.982999
[19:153/233]    loss: 0.886529 t_loss: 0.509828 s_loss: 0.376701        time: 13.028500
[19:173/233]    loss: 0.815399 t_loss: 0.512264 s_loss: 0.303135        time: 13.055999
[19:193/233]    loss: 0.848488 t_loss: 0.496589 s_loss: 0.351899        time: 13.076501
[19:213/233]    loss: 0.734683 t_loss: 0.441243 s_loss: 0.293440        time: 13.096502
[19:233/233]    loss: 0.568186 t_loss: 0.295135 s_loss: 0.273050        time: 12.498997
```

- Problem 2

```python
def __getitem__(self, idx):
    temproal_flag = idx % 2 == 0
    record = self.objects_list[idx]
    if self.test_stage:
        temp_perm = np.arange(self.frame_num)
        temp_label = 0
    else:
        if random.random() < 0.5: # normal and abnormal ratio
            temp_perm = np.arange(self.frame_num)
            temp_label = 0
        else:
            temp_perm = np.random.permutation(self.frame_num)
            temp_label = 1
            if np.array_equal(temp_perm, np.arange(self.frame_num)):
                temp_label = 0
    obj = self.get_object(record["video_name"], record["frame"], record["object"])
```

  首先更改 dataset.py 內的__getitem__，將 temporal 排列的比例設成 1:1，並且區段多增加 temporal label，讓回傳的時候把 temp_perm 改成 temp_label

```python
net = model.WideBranchNet(time_length=args.sample_num, num_classes=[1, 81])
```

```python
criterion_bce = nn.BCEWithLogitsLoss(reduction='mean')
```

```python
temp_labels = temp_labels[t_flag].long().view(-1).cuda(args.device).float()
```

```python
temp_logits = temp_logits[t_flag].view(-1)
```

```python
temp_loss = criterion_bce(temp_logits, temp_labels)
```

  隨後回到 main.py，先行修改網路架構最後的輸出，因為是輸出 anomaly probability，所以為 1，並且因為機率落在 0~1 之前，使用 BCE 這個 loss

比原本的 CE 來的更好一點。訓練過程中，模型產生的 logits 原本會 reshape 成最後一維變成 frame num，但因為目前 label 也會是 1 個數字，所以不需要 reshape，最後就是套用 BCE 計算 loss。

```
temp_scores = F.sigmoid(temp_logits).squeeze().cpu().numpy()
```

在 val 更新的地方就是將 logits 套用 sigmoid 讓他變成 anomaly probability，隨後計算 micro AUROC。

以下是結果：

```
Load 5 videos 5527 frames, 21558 objects, excluding 244 inside objects and 0 small objects.
(smoothing: True): dataset = avenue, auc = 0.8540462684117853  aver_result: [0.801853290272067]
(smoothing: True): dataset = avenue, auc = 0.6300230188896823  aver_result: [0.4715014429876069]
(smoothing: True): dataset = avenue, auc = 0.7952019586014805  aver_result: [0.6778324543213733]
cur max: 0.7952019586014805 in 2024-05-23-17-54-17
[19:93/233]     loss: 0.328313 t_loss: 0.049003 s_loss: 0.279310        time: 81.690471
[19:113/233]    loss: 0.289360 t_loss: 0.008939 s_loss: 0.280421        time: 13.388000
[19:133/233]    loss: 0.319007 t_loss: 0.006082 s_loss: 0.312925        time: 13.350501
[19:153/233]    loss: 0.236193 t_loss: 0.007343 s_loss: 0.228851        time: 13.424500
[19:173/233]    loss: 0.300207 t_loss: 0.027834 s_loss: 0.272373        time: 13.406500
[19:193/233]    loss: 0.307707 t_loss: 0.013576 s_loss: 0.294131        time: 13.709501
[19:213/233]    loss: 0.266313 t_loss: 0.016762 s_loss: 0.249551        time: 13.785001
[19:233/233]    loss: 0.240012 t_loss: 0.006050 s_loss: 0.233961        time: 13.258500
```

- Problem 3

```
self.all_temp_perms = np.array(list(itertools.permutations(np.arange(self.frame_num))))
```

```
def __getitem__(self, idx):
    temproal_flag = idx % 2 == 0
    record = self.objects_list[idx]
    temp_label = np.zeros(120, dtype=float)
    if self.test_stage:
        temp_perm = np.arange(self.frame_num)
        temp_label[0] = 1
    else:
        if random.random() < 0.5: # normal and abnormal ratio
            temp_perm = np.arange(self.frame_num)
            temp_label[0] = 1
        else:
            temp_perm = np.random.permutation(self.frame_num)
            match_index = np.where(np.all((self.all_temp_perms == temp_perm), axis=1))[0]
            temp_label[match_index] = 1
    obj = self.get_object(record["video_name"], record["frame"], record["object"])
```

一樣先修改 dataset.py，先生成一個 120 種全排列可能的 array 用於後續比對，隨後在 temporal 區段內將 label 設定成長度為 120 的陣列，將產生的 permutation 與全排列去做比對，找到對應的 index，將 label array 內的該 index 設定為 1

```
net = model.WideBranchNet(time_length=args.sample_num, num_classes=[120, 81])
```

```
temp_labels = temp_labels[t_flag].long().view(-1, 120).cuda(args.device).float()
```

```
temp_logits = temp_logits[t_flag].view(-1, 120)
```

回到 main.py，一樣修改網路架構將輸出改成 120，即為每一種排列的 probability，隨後同樣將最後一維 reshape 成對應輸出維度。

```
temp_probs = F.softmax(temp_logits, dim=-1)
temp_scores = 1 - temp_probs[:, 0].cpu().numpy()
```

在 val 裡面，因為要求 anomaly score 為 1 – 正常排列的機率，所以首先以 softmax 計算全部機率，再做 anomaly score 的計算。

以下是結果：

```
Load 5 videos 5527 frames, 21558 objects, excluding 244 inside objects and 0 small objects.
(smoothing: True): dataset = avenue, auc = 0.8441128856994593  aver_result: [0.7926090687171692]
(smoothing: True): dataset = avenue, auc = 0.6643832578603521  aver_result: [0.4992611543352604]
(smoothing: True): dataset = avenue, auc = 0.793640532760892  aver_result: [0.6742674249198798]
cur max: 0.799369087272195 in 2024-05-23-19-33-35
[19:93/233]     loss: 0.802635 t_loss: 0.371822 s_loss: 0.430813        time: 81.133516
[19:113/233]    loss: 0.715775 t_loss: 0.259878 s_loss: 0.455897        time: 12.938500
[19:133/233]    loss: 0.945216 t_loss: 0.405487 s_loss: 0.539729        time: 12.996003
[19:153/233]    loss: 0.884406 t_loss: 0.358517 s_loss: 0.525888        time: 13.046499
[19:173/233]    loss: 0.939530 t_loss: 0.530593 s_loss: 0.408937        time: 13.075498
[19:193/233]    loss: 0.735643 t_loss: 0.353411 s_loss: 0.382231        time: 13.096000
[19:213/233]    loss: 0.807238 t_loss: 0.435859 s_loss: 0.371379        time: 13.203500
[19:233/233]    loss: 0.684451 t_loss: 0.190362 s_loss: 0.494088        time: 12.745500
```

- Problem 4

  從三種版本來看，第一個沒有任何修改的原 paper 的 code 表現最好，有 0.938，而第二個版本因為只有簡單應用了是或否的二元分類，任務目標可能對於該模型來說太簡單，所以 t_loss 降到很低，然而表現效果沒有很好，為 0.792，有可能存在 overfitting 的情況。最後第三個版本是預測 120 種排列的機率，在任務目標中應該是最難的，因為要考慮到時間的連續性，表現上與第二個版本差不多，為 0.799，但從 t_loss 來看震盪較大，可能還在處於 underfitting 的情況下，若是多 train 幾個 epoch 可能表現會再更好一些。