CS5658 Anomaly Detection

Homework4 – Video AD

Student ID: 109062211 Name: 張惇媛

1.

Result

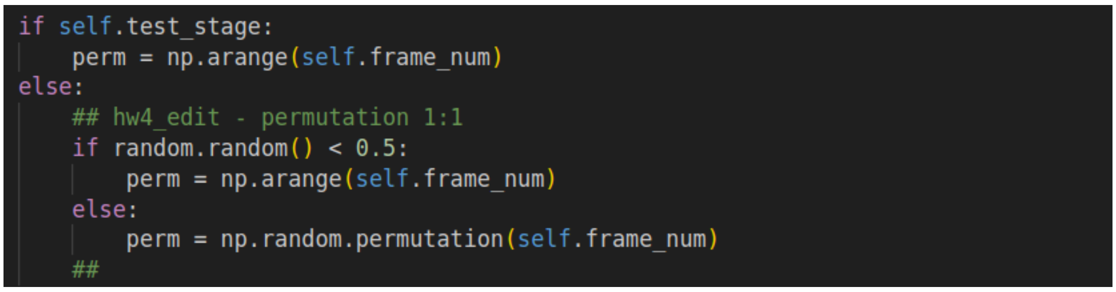
micro_auc: 0.7885

2. Binary Classification

Implementation

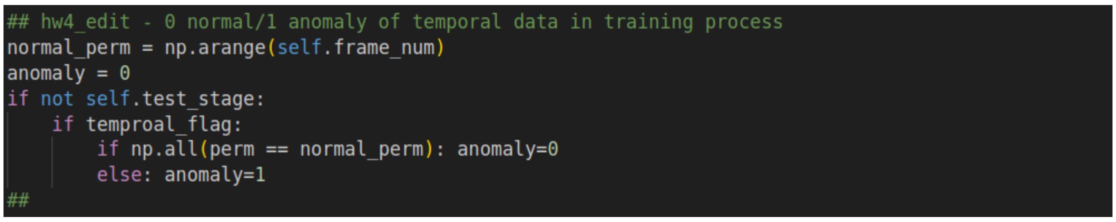
a. dataset.py

- function **__get_item__()**
  - modify the probability of permutation as 0.5 to get the ratio of normal:anomaly to 1:1

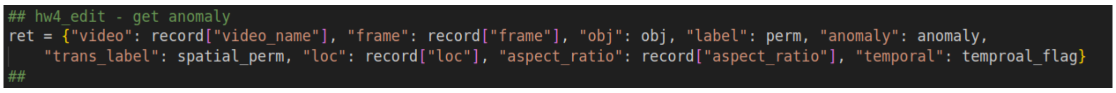


```python
if self.test_stage:
    perm = np.arange(self.frame_num)
else:
    ## hw4_edit - permutation 1:1
    if random.random() < 0.5:
        perm = np.arange(self.frame_num)
    else:
        perm = np.random.permutation(self.frame_num)
    ##
```

  - add the attribute of anomaly label, while 0 represents as normal and 1 represents as anomaly



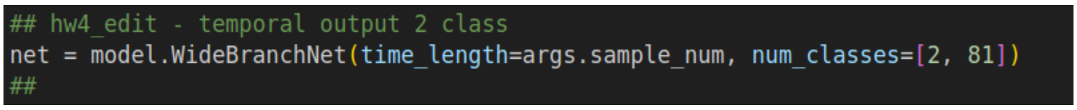
```python
## hw4_edit - 0 normal/1 anomaly of temporal data in training process
normal_perm = np.arange(self.frame_num)
anomaly = 0
if not self.test_stage:
    if temproal_flag:
        if np.all(perm == normal_perm): anomaly=0
        else: anomaly=1
##
```



```python
## hw4_edit - get anomaly
ret = {"video": record["video_name"], "frame": record["frame"], "obj": obj, "label": perm, "anomaly": anomaly,
    "trans_label": spatial_perm, "loc": record["loc"], "aspect_ratio": record["aspect_ratio"], "temporal": temproal_flag}
##
```

b. main.py

- function **train()**
  - modify the number of classes for temporal permutation prediction model as 2



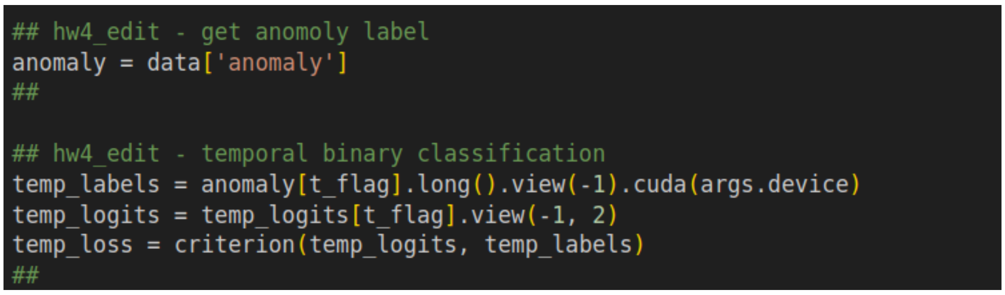
```python
## hw4_edit - temporal output 2 class
net = model.WideBranchNet(time_length=args.sample_num, num_classes=[2, 81])
##
```

- modify the temporal logits, labels, and loss:

  - For temporal logits, reshape as (-1, 2) because of binary classification

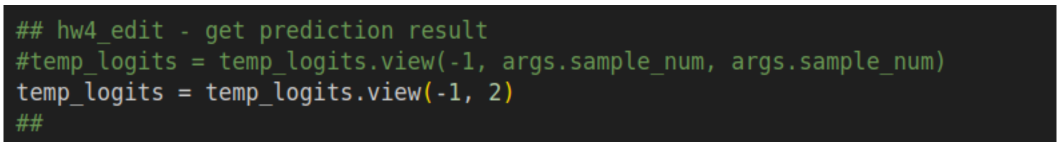
  - For temporal labels, get the anomaly labels from data's attributes

  - For temporal loss, use the new temporal logits and labels for the calculation of the cross entropy loss



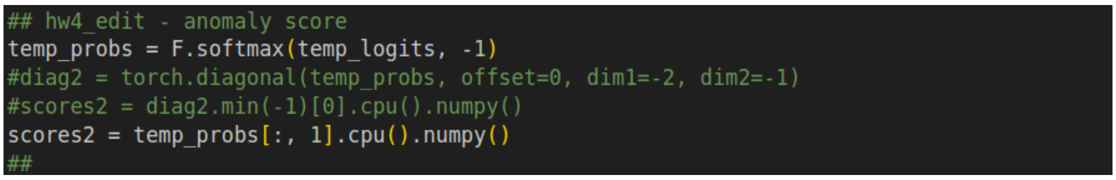
```
## hw4_edit - get anomoly label
anomaly = data['anomaly']
##

## hw4_edit - temporal binary classification
temp_labels = anomaly[t_flag].long().view(-1).cuda(args.device)
temp_logits = temp_logits[t_flag].view(-1, 2)
temp_loss = criterion(temp_logits, temp_labels)
##
```

- function **val()**
  - modify the temp_logits.shape as (-1, 2) to get the binary classification temporal prediction result



```
## hw4_edit - get prediction result
#temp_logits = temp_logits.view(-1, args.sample_num, args.sample_num)
temp_logits = temp_logits.view(-1, 2)
##
```

  - modify the computation for scores2(t_score), using the probability of anomaly as the anomaly score for temporal permutation prediction



```
## hw4_edit - anomaly score
temp_probs = F.softmax(temp_logits, -1)
#diag2 = torch.diagonal(temp_probs, offset=0, dim1=-2, dim2=-1)
#scores2 = diag2.min(-1)[0].cpu().numpy()
scores2 = temp_probs[:, 1].cpu().numpy()
##
```

Result

micro_auc: 0.6380

3. Multiclass Classification

Implementation

a. table.py (Adding)

Design this python file for finding the index of permutation. The permutation index is defined as the following:

Permutation #0: 01234

Permutation #1: 01243

…

Permutation #119: 43210

- function **find_index()**
  - return the index of permutation

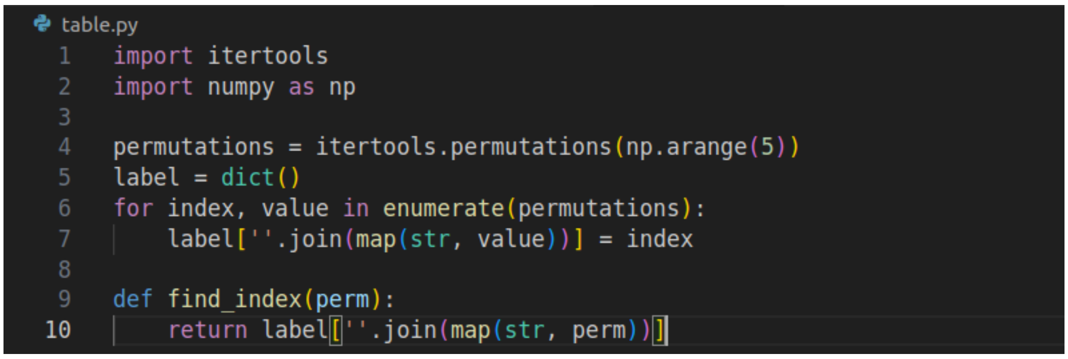
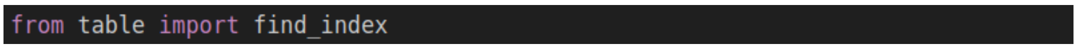


table.py
```python
import itertools
import numpy as np

permutations = itertools.permutations(np.arange(5))
label = dict()
for index, value in enumerate(permutations):
    label[''.join(map(str, value))] = index

def find_index(perm):
    return label[''.join(map(str, perm))]
```

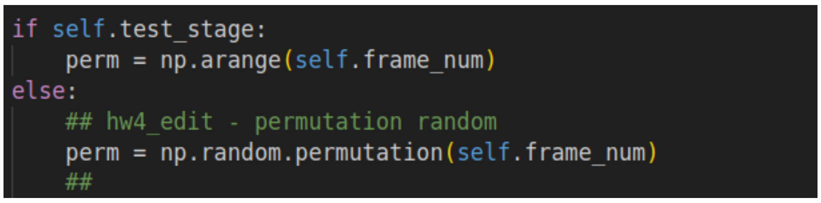
b. dataset.py

- import adding function **find_index()**



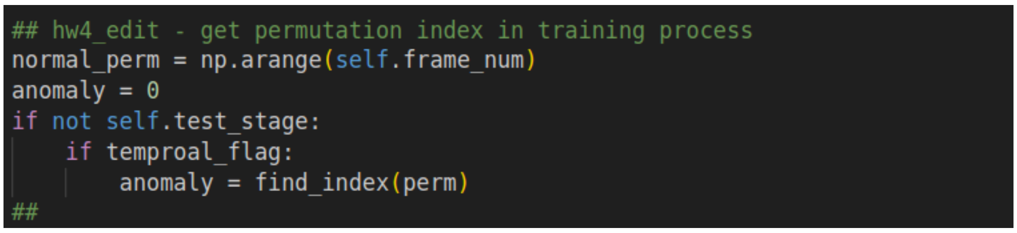
```python
from table import find_index
```

- function **__get_item__()**
  - ■ modify the permutation as random (each probability of permutation is equivalently equal)

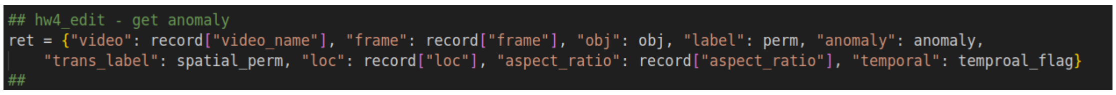


```python
if self.test_stage:
    perm = np.arange(self.frame_num)
else:
    ## hw4_edit - permutation random
    perm = np.random.permutation(self.frame_num)
    ##
```

  - ■ add the attribute of anomaly label, while the label represents as the permutation index derived from callable function **find_index()**



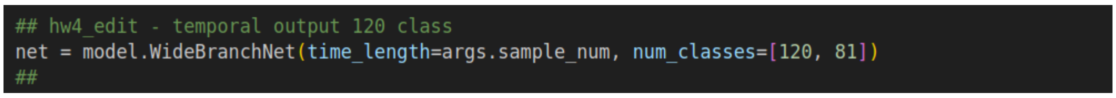
```python
## hw4_edit - get permutation index in training process
normal_perm = np.arange(self.frame_num)
anomaly = 0
if not self.test_stage:
    if temproal_flag:
        anomaly = find_index(perm)
##
```



```python
## hw4_edit - get anomaly
ret = {"video": record["video_name"], "frame": record["frame"], "obj": obj, "label": perm, "anomaly": anomaly,
    "trans_label": spatial_perm, "loc": record["loc"], "aspect_ratio": record["aspect_ratio"], "temporal": temproal_flag}
##
```

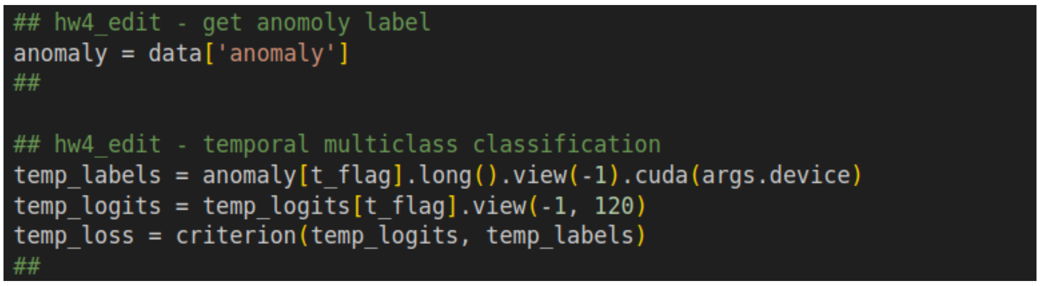
c. main.py

- function **train()**
  - ■ modify the number of classes for temporal permutation prediction model as 5! = 120



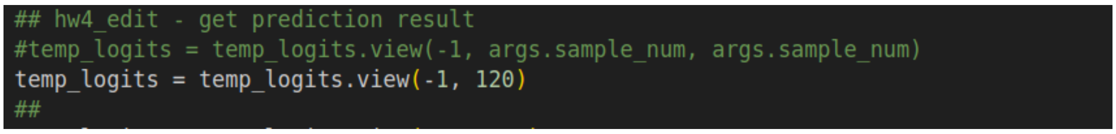
```
## hw4_edit - temporal output 120 class
net = model.WideBranchNet(time_length=args.sample_num, num_classes=[120, 81])
##
```

- modify the temporal logits, labels, and loss:
  - For temporal logits, reshape as (-1, 120) because of multiclass classification
  - For temporal labels, get the anomaly labels from data's attributes
  - For temporal loss, use the new temporal logits and labels for the calculation of the cross entropy loss



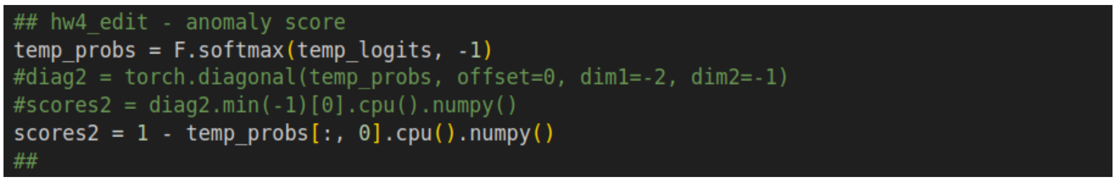
```
## hw4_edit - get anomoly label
anomaly = data['anomaly']
##

## hw4_edit - temporal multiclass classification
temp_labels = anomaly[t_flag].long().view(-1).cuda(args.device)
temp_logits = temp_logits[t_flag].view(-1, 120)
temp_loss = criterion(temp_logits, temp_labels)
##
```

- function **val()**
  - modify the temp_logits.shape as (-1, 120) to get the multiclass classification temporal prediction result



```
## hw4_edit - get prediction result
#temp_logits = temp_logits.view(-1, args.sample_num, args.sample_num)
temp_logits = temp_logits.view(-1, 120)
##
```

  - modify the computation for scores2(t_score), using 1 - Probability of Permutation #0 (01234) as the anomaly score for temporal permutation prediction



```
## hw4_edit - anomaly score
temp_probs = F.softmax(temp_logits, -1)
#diag2 = torch.diagonal(temp_probs, offset=0, dim1=-2, dim2=-1)
#scores2 = diag2.min(-1)[0].cpu().numpy()
scores2 = 1 - temp_probs[:, 0].cpu().numpy()
##
```

<u>Result</u>

micro_auc: 0.6336

4. Observation/Conclusion

Problem 1 version (probability matrix prediction) has the best performance than others version (direct classification) due to rich information content, handling ambiguity, and improved generalization. By providing a detailed probability distribution over all possible classes, the model offers detailed insights into prediction. This helps in effectively managing ambiguous cases, as the probability distribution reflects uncertainty rather than forcing a potentially incorrect classification. Moreover, focusing on probability estimation helps the model to learn about the overall data distribution, leading to better generalization to unseen data. As a result, Problem 1 version has a higher AUROC.

Based on the above AUROC records, Problem 2 version (binary classification) has better performance compared to Problem 3 version (multiclass classification). The possible reason is that binary classification has a simpler decision boundary and lower dimensionality. As the number of possible permutations increases, the problem becomes significantly more complex due to the number of classes the model needs to distinguish. In a multiclass classification scenario, this complexity results in higher dimensionality, making it more challenging for the model to accurately learn and generalize across all possible permutations. Consequently, the decision boundaries become more complicated and harder to optimize effectively. On the other hand, binary classification has lower dimensionality, focusing only on distinguishing between normal and anomaly sequences. This simplification allows the model to establish clearer decision boundaries and perform better. As a result, Problem 2 version has a higher AUROC.