Graph Convolutional Label Noise Cleaner:
Train a Plug-and-play Action Classifier for Anomaly Detection
Supplementary Materials (Appendix)

Jia-Xing Zhong, Nannan Li, Weijie Kong, Shan Liu, Thomas H. Li, Ge Li
School of Electronic and Computer Engineering, Peking University
Institute of Intelligent Video Audio Technology, Longgang Shenzhen
Tencent America

jxzhong@pku.edu.cn lnnsiat@gmail.com weijie.kong@pku.edu.cn
shanl@tencent.com tli@aiit.org.cn geli@ece.pku.edu.cn

1. Processing Speed in the Test Phase

<table>
<thead>
<tr>
<th>Input Size (Pixel)</th>
<th>Speed (FPS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C3D 112 × 112</td>
<td>123.08</td>
</tr>
<tr>
<td>TSN-RGB 224 × 224</td>
<td>30.96</td>
</tr>
<tr>
<td>TSN-Optical Flow</td>
<td>150.15</td>
</tr>
</tbody>
</table>

Table 1: Testing speed (FPS) of our models on a Titan-XP GPU. Note that the reported result includes all pre-processing operations, such as resizing, 10-crop oversampling, zero-centering, etc.

Our approach of directly utilizing action classifiers for anomaly detection has great computational efficiency. As shown in Table 1, we report the frame per second (FPS) performance of the two types of action classifiers. Although the time-consuming pre-processes (e.g., 10-crop oversampling) are taken into consideration, the three action classifiers still have the real-time or even the super real-time performance.

2. Implementation of Label Noise Cleaner

At the first cleaning step, we select the 30% and 60% highest-confidence snippets as $H$ for two-stream and C3D networks respectively if not specified, and increase the cardinality of $H$ by 30% at each step. To learn an unbiased model, we also include normal videos in training data. To generate the label assignments of action classifiers, we concentrate the output probability into a single anomaly category with a min-max normalization. The output dimensions of the first two fully connected layers are 512 and 128 respectively, at the 60% dropout rate. Both the graph modules have two convolutional layers: a 32-unit hidden layer activated by ReLu and the last 1-unit output layer. Due to the limited memory of GPUs, we at most sample 1,600 high-confidence snippets with not more than 8 neighbours respectively in a video. We implement our noise cleaner upon Pytorch with the following hyper-parameters: base_learning_rate = 0.0001, momentum = 0.9 and weight_decay = 0.0005. In preliminary experiments, we observe that three iterations are sufficient in most cases. Therefore we repeat the alternate optimization until the 3rd step and compare the last (not always the best) results with other methods.

3. More Comparisons on UCSD-Peds

Several unary-classification works in 2018 also conduct experiments on UCSD-Peds. As shown in Table 3 and the main body of our paper, their default implementations are not directly comparable with ours because of different data splits. For some open-source works, we hereby reproduce experiments on the data split in [1] as ours, while the results in their original papers are also provided within square parentheses “[ ]” for reference as reported in Table 2. Since UCF-Crime is released at Github on June 10th 2018 lately, except the official reference [11] and its comparisons, neither public reporting of results nor source codes can be found, and we hope that our work can fill in the blanks.

4. Vectorized Feature Similarity Module

Following the main body of our paper, we denote the feature similarity graph as $F = (V, E, X)$, where $V$ is the vertex set, $E$ is the edge set, and $X$ is the attribute of vertexes. In particular, $V$ is a video, $E$ describes the feature similarity amongst snippets, and $X \in \mathbb{R}^{N \times d}$ represents the $d$-dimensional feature of these $N$ snippets. The adjacency matrix $A \in \mathbb{R}^{N \times N}$ of $F$ is defined as:

$$A_{i,j} = \exp(X_i \cdot X_j - \max(X_i \cdot X)),$$  
(1)
<table>
<thead>
<tr>
<th>Method</th>
<th>Publication</th>
<th>AUC (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP [9]</td>
<td>WACV 2018</td>
<td>No source codes [88.4]</td>
</tr>
<tr>
<td>Frame Prediction [6]</td>
<td>CVPR 2018</td>
<td>92.6 ± 1.1 95.4</td>
</tr>
<tr>
<td>C2ST [7]</td>
<td>BMVC 2018</td>
<td>81.4 ± 2.8 87.5</td>
</tr>
</tbody>
</table>

**Unary-classification Paradigm**

<table>
<thead>
<tr>
<th>Method</th>
<th>Publication</th>
<th>AUC (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours-TSN</td>
<td>J-Mult. Nov. 2018</td>
<td>90.1</td>
</tr>
<tr>
<td>Ours-TSNGray-scale</td>
<td>–</td>
<td>93.2 ± 2.3</td>
</tr>
<tr>
<td>Ours-TSNOpticalFlow</td>
<td>–</td>
<td>92.8 ± 1.6</td>
</tr>
</tbody>
</table>

**Table 2: Comparison on UCSD-Peds in 2018.** The results of their original papers under data split [5] are reported within “[ ]”.

<table>
<thead>
<tr>
<th>Splitting Approach</th>
<th>Train Normal</th>
<th>Abnormal</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Following [5]</td>
<td>16</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>Following [1]</td>
<td>4</td>
<td>6</td>
<td>18</td>
</tr>
</tbody>
</table>

**Table 3: Difference in splitting UCSD-Peds.** The random selection is repeated 10 times in [1].

where the element \( A_{(i,j)} \) measures the feature similarly between the \( i^{th} \) and \( j^{th} \) snippets. Here is an equivalent vectorization of Equation 1:

\[
A = \exp(XX^T - \text{torch}.max(XX^T, \text{dim} = 1)), \tag{2}
\]

where the \text{torch}.max function takes the maximum value over dimension 1.

The nearby vertexes are driven to have the same anomaly label via the graph-Laplacian operation approximated with a renormalization trick [3]:

\[
\hat{A} = \hat{D}^{-\frac{1}{2}}A\hat{D}^{-\frac{1}{2}}, \tag{3}
\]

where the self-loop adjacency matrix \( \hat{A} = A + I_n \), and \( I_n \in \mathbb{R}^{N \times N} \) is the identity matrix; \( \hat{D} \) is the corresponding degree matrix:

\[
\hat{D}_{(i,i)} = \sum_j \hat{A}_{(i,j)}. \tag{4}
\]

The vectorization of Equation 4 is implemented with the vectorized summation and the broadcasting diagonal functions of Pytorch:

\[
\hat{D} = \text{torch}.diag(\text{torch}.sum(\hat{A}, \text{dim} = 1)). \tag{5}
\]

Finally, the output \( H \) of a feature similarity graph module layer is computed as:

\[
H = \sigma(\hat{A}XW), \tag{6}
\]

where \( W \) is a trainable parametric matrix, and \( \sigma \) is an activation function.

Since the whole computational procedure is differentiable, our feature similarity graph module can be trained in an end-to-end fashion. Therefore, neural networks are capable of seamlessly incorporating the single or multiple stacked modules. The temporal similarity module can be also rewritten as its corresponding vectorized implementation in a similar manner.

**5. Details of Indirectly Supervised Loss Term**

Our indirectly supervised term of the loss function can be viewed as a temporal ensembling strategy [4]. The pseudo code is shown in Algorithm 1. In practice, we set \( \gamma = 0.5 \) in all of the experiments. Since we have already obtained a set of rough predictions from the action classifier, the “cool start” initialization and the bias correction of the original temporal ensembling method [4] are not required as illustrated on the 1st and the 8th statements.

**6. Reorganization of ShanghaiTech**

<table>
<thead>
<tr>
<th>Normal Videos</th>
<th>Anomaly Videos</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>175</td>
<td>63</td>
<td>238</td>
</tr>
<tr>
<td>155</td>
<td>44</td>
<td>199</td>
</tr>
<tr>
<td>330</td>
<td>107</td>
<td>437</td>
</tr>
</tbody>
</table>

**Table 4: The number of videos on our reorganized ShanghaiTech.**

In total, there are 437 videos on ShanghaiTech. As shown in Table 4, we split the data into two subsets: the training set is made up of 238 videos, and the testing set contains 199 videos. In each scene, the numbers of normal and anomaly videos w.r.t. the two subsets are depicted in Figure 1 and Figure 2, respectively. The new
data split is available at https://github.com/jx-zhong-for-academic-purpose/GCN-Anomaly-Detection.

7. Discuss the Formulation

Following the reviewer’s suggestion, we discuss our noisy-labeled problem formulation and the EM-like optimization mechanism under this formulation in more detail.

7.1. Concept: MIL vs Noisy-labeled Learning

Conceptually, the two formulations mainly differ in their emphases. Given a positive bag \( Y = 1 \), the MIL usually focuses on positive instances \( y_i = 1 \), whereas the noisy-labeled training pays attention to noisy labels \( y_i = 0 \) and the remaining ones are \( y_i = 1 \). The two conceptions are complementary and have transformational relations.

7.2. Practice: EM-like MIL vs Ours

Practically, in terms of selection criteria on “seed examples”, the EM-like MIL focuses on the most-likely positive instances, while our noisy-labeled optimization prefers the most-likely reliable predictions. Take the three MIL models the reviewer mentioned for examples. If the 10-crop prediction of a snippet within an anomalous video is \{0.2, 0.2, ..., 0.2\}, He et al. [1] will not update their “anchor dictionary” with it for its low anomaly score (mean value=0.2), Hou et al. [2] will exclude it because it is “non-discriminative” (without “the same label” as the corresponding video), Zhang et al. [12] will neglect it since their E-step is to seek the most “responsible” instance to the bag annotation, but we will select it to supervise our GCN because it is highly certain and noiseless (predictive variance=0).

7.3. Terminology: EM-like vs EM-based

As pointed out in the main body of this paper, our updating method is “EM-like” instead of “EM-based”. The resemblance between our optimization mechanism and the EM-based approach is that they both alternately repeat update-and-fix processes. However, our method is not “EM-based” since we do not explicitly estimate mathematical expectation in the training process.

References

**Algorithm 1 Indirectly Supervised Loss Term.**

Note that the practical computational processes are incrementally implemented, while in this pseudo code all of them are calculated from the 1\textsuperscript{st} epoch for clarity.

**Input:**
- $V = \{v_i\}_{i=1}^N$: a video with $N$ snippets
- $\tilde{Y} = \{\tilde{y}_i\}_{i=1}^N$: the rough snippet-wise anomaly probabilities from the last action classifier
- $p_\theta(v_i)$: the GCN predictions of video clips $v_i$ with trainable parameters $\theta$
- $\alpha(v_i)$: the stochastic augmentation (such as dropout and random cropping) function of input snippets $v_i$
- $\gamma$: a hyper-parametric discount factor within the range of $(0, 1)$

**Output:**
- $\mathcal{L}_j^I$: the indirectly supervised loss at the $j\textsuperscript{th}$ epoch

1: Initialize the smooth target $\bar{p}_i \in 1, 2, ..., N = \bar{y}_i \in 1, 2, ..., N$  
2: repeat  
3: Initialize the epoch counter $j = 0$  
4: for each video $V$ in the training set do  
5: Obtain the GCN predictions of augmented snippets: $p_i = p_\theta(\alpha(v_i))$  
6: Compute the loss under indirect supervision: $\mathcal{L}_j^I = \frac{1}{N} \sum_{i=1}^N |p_i - \bar{p}_i|$  
7: Optimize the parameters $\theta$ of the GCN  
8: Update the smooth target: $\bar{p}_i \in 1, 2, ..., N = \gamma \bar{p}_i \in 1, 2, ..., N + (1 - \gamma)p_i \in 1, 2, ..., N$  
9: Update the epoch counter: $j = j + 1$  
10: until $j == \text{current epoch number}$

---

**Figure 1:** *Training set on the reorganization of ShanghaiTech.*

**Figure 2:** *Testing set on the reorganization of ShanghaiTech.*