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

Jia-Xing Zhong[1,2] Nannan Li[3,1,2] Weijie Kong[1,2] Shan Liu[4] Thomas H. Li[1] Ge Li ✉[1,2]
[1]School of Electronic and Computer Engineering, Peking University [2]Peng Cheng Laboratory
[3]Institute of Intelligent Video Audio Technology, Longgang Shenzhen [4]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

	Input Size (Pixel)	Speed (FPS)
C3D	112×112	123.08
TSN-RGB	224×224	30.96
TSN-Optical Flow	224×224	150.15

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 [10] 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 [8] 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 3^{rd} 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 10^{th} 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 $F = (V, E, \mathbf{X})$, where V is the vertex set, E is the edge set, and $\mathbf{X}$ is the attribute of vertexes. In particular, V is a video, E describes the feature similarity amongst snippets, and $\mathbf{X} \in \mathbb{R}^{\mathbf{N}\times\mathbf{d}}$ represents the d-dimensional feature of these N snippets. The adjacency matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$ of F is defined as:

$$\mathbf{A}_{(i,j)} = \exp(\mathbf{X}_i \cdot \mathbf{X}_j - max(\mathbf{X}_i \cdot \mathbf{X})), \quad (1)$$

Method	Publication	AUC (%)
Unary-classification Paradigm		
...	...	...
TCP [9]	WACV 2018	No source codes [88.4]
Frame Prediction [6]	CVPR 2018	**92.6 $\pm$ 1.1 [95.4]**
C2ST [7]	BMVC 2018	81.4 $\pm$ 2.8 [87.5]
Binary-classification Paradigm		
AL [1]	J-Mult. Nov. 2018	90.1
Ours-TSN$^{Gray-scale}$	–	**93.2 $\pm$ 2.3**
Ours-TSN^{OpticalFlow}	–	92.8 $\pm$ 1.6

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

Splitting Approach	Train		Test
	Normal	Abnormal	
Following [5]	16	0	12
Following [1]	4	6	18

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

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

$$\mathbf{A} = \exp(\mathbf{X}\mathbf{X}^T - torch.max(\mathbf{X}\mathbf{X}^T, dim = 1)), \quad (2)$$

where the $torch.max$ 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]:

$$\widehat{\mathbf{A}} = \widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}, \quad (3)$$

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

$$\widetilde{\mathbf{D}}_{(i,i)} = \sum_j \widetilde{\mathbf{A}}_{(i,j)}. \quad (4)$$

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

$$\widetilde{\mathbf{D}} = torch.diag(torch.sum(\widetilde{\mathbf{A}}, dim = 1)). \quad (5)$$

Finally, the output $\mathbf{H}$ of a feature similarity graph module layer is computed as:

$$\mathbf{H} = \sigma(\widehat{\mathbf{A}}\mathbf{X}\mathbf{W}), \quad (6)$$

where $\mathbf{W}$ is a trainable parametric matrix, and σ 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 γ as 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 1^{st} and the 8^{th} statements.

6. Reorganization of ShanghaiTech

	Training Set	Test Set	Total
Normal Videos	175	155	330
Anomaly Videos	63	44	107
Total	238	199	437

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

[1] Chengkun He, Jie Shao, and Jiayu Sun. An anomaly-introduced learning method for abnormal event detection. *Multimedia Tools and Applications*, 77(22):29573–29588, Nov 2018. 1, 2, 3

[2] L. Hou, D. Samaras, T. M. Kurc, Y. Gao, J. E. Davis, and J. H. Saltz. Patch-based convolutional neural network for whole slide tissue image classification. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2424–2433, June 2016. 3

[3] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017. 2

[4] Samuli Matias Laine and Timo Oskari Aila. Temporal ensembling for semi-supervised learning, Apr. 12 2018. US Patent App. 15/721,433. 2

[5] W. Li, V. Mahadevan, and N. Vasconcelos. Anomaly detection and localization in crowded scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36:18–32, 2014. 2

[6] W. Liu, W. Luo, D. Lian, and S. Gao. Future frame prediction for anomaly detection a new baseline. In *CVPR*, June 2018. 2

[7] Yusha Liu, Chun-Liang Li, and Barnabás Póczos. Classifier two-sample test for video anomaly detections. In *BMVC*, 2018. 2

[8] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 1

[9] M. Ravanbakhsh, M. Nabi, H. Mousavi, E. Sangineto, and N. Sebe. Plug-and-play cnn for crowd motion analysis: An application in abnormal event detection. In *WACV*, 2018. 2

[10] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. 1

[11] Waqas Sultani, Chen Chen, and Mubarak Shah. Real-world anomaly detection in surveillance videos. In *CVPR*, June 2018. 1

[12] Qi Zhang and Sally A Goldman. Em-dd: An improved multiple-instance learning technique. In *Advances in neural information processing systems*, pages 1073–1080, 2002. 3

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^{st} epoch for clarity.

Input:

$V = \{v_i\}_{i=1}^{N}$: a video with N snippets

$\widetilde{Y} = \{\widetilde{y}_i\}_{i=1}^{N}$: 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 θ

$\alpha(v_i)$: the stochastic augmentation (such as dropout and random cropping) function of input snippets v_i

γ: a hyper-parametric discount factor within the range of $(0, 1)$

Output:

$\mathcal{L}_I^j$: the indirectly supervised loss at the j^{th} epoch

```
Initialize the smooth target p̄_{i∈1,2,..,N} = ỹ_{i∈1,2,..,N}
repeat
    Initialize the epoch counter j = 0
    for each video V in the training set do
        Obtain the GCN predictions of augmented snippets: p_i = p_θ(α(v_i))
        Compute the loss under indirect supervision: L_I^j = (1/N) Σ_{i=1}^{N} |p_i − p̄_i|
        Optimize the parameters θ of the GCN
        Update the smooth target: p̄_{i∈1,2,..,N} = γ p̄_{i∈1,2,..,N} + (1 − γ) p_{i∈1,2,..,N}
    Update the epoch counter: j = j + 1
until j == current epoch number
```

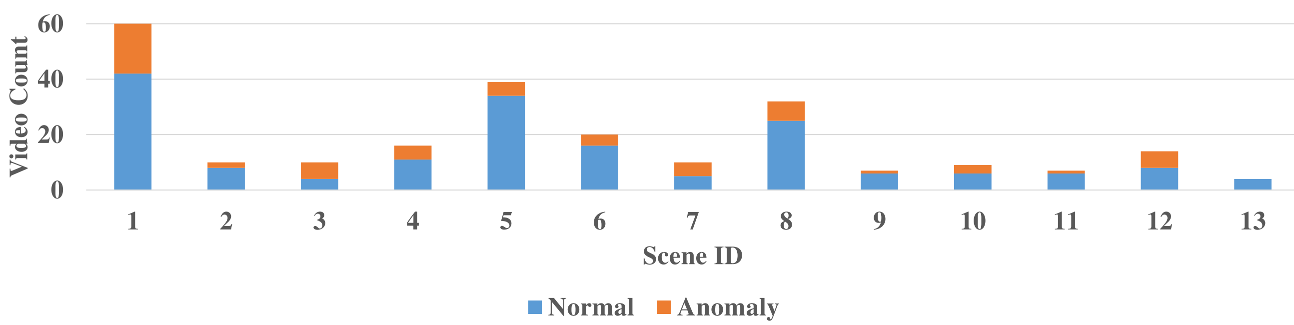

Figure 1: *Training set on the reorganization of ShanghaiTech.*

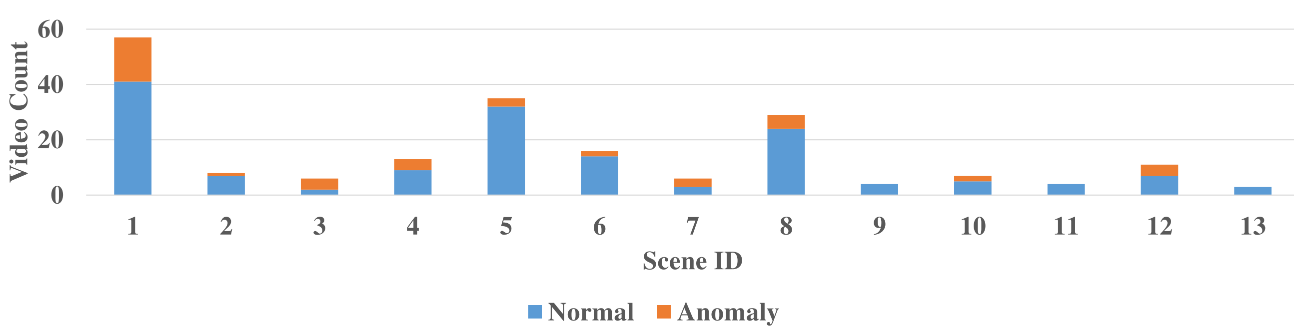

Figure 2: *Testing set on the reorganization of ShanghaiTech.*