Indian Institue of Technology Bombay
Electrical Engineering

B.TECH PROJECT - II REPORT
ANOMALOUS HUMAN ACTIVITY DETECTION

Parth Patil 170070011
Guide:
Rajbabu Velmurugan
Biplab Banerjee

June 30, 2021

1 Abstract

Anomaly detection refers to the act of identifying behaviour in data which is different than normal. This can range from detecting malicious activity in a video to detecting earthquake in a seismic data. Anomaly detection methods based on convolutional neural networks (CNNs) typically leverage proxy tasks, such as reconstructing input video frames, to learn models describing normality without seeing anomalous samples at training time, and quantify the extent of abnormalities using the explicit reconstruction error at test time. The main drawbacks of these approaches are that they do not consider the diversity of normal patterns. Also, many existing approaches requires large number of normal samples from a particular scene before it could detect anomalies. This makes deployment in real world impractical. In this report we will have a look at approach which tackles both the issues. We will also look at the detail code which can enable any anomaly detection model to be adapted for a new scene using few frames. The code is available at https://github.com/Parth1811/Few-Shot-Protypical-Anomaly-Detection

2 Introduction & Background

2.1 Anomaly Detection

The activities of a human being can be broadly classified into normal activities or anomalous activities. A human being's deviation from normal behavior causing harm to the surrounding or to himself is categorized as an anomalous activity. The extensive research of human activity recognition and its applications has thrown light upon anomaly detection. The existing approaches for anomalous human activity recognition are built based on the type and speed of object movements along with how the objects of interest interact with each other.

It is extensively used for prediction of time series data in which certain regularities are checked in the data dimension. This can be used to predict stock market prices, weather forecasting and time series prediction. It is also applied in the area of surveillance. This involves identifying abnormal activities in videos which otherwise is heavily dependant on manual monitoring and is therefore both human resource intensive and time consuming. Some existing approaches to go about this task includes learning the latent space representation corresponding to normal activities.

2.2 Memory Networks

There are a number of attempts to capture long-term dependencies in sequential data. Long short-term memory (LSTM) addresses this problem using local memory cells, where hidden states of the network record information in the past partially. The memorization performance is, however, limited, as the size of the cell is typically small and the knowledge in the hidden state is compressed. To overcome the limitation, memory networks have recently been introduced. It uses a global memory that can be read and written to, and performs a memorization task better than classical approaches. The memory networks, however, require layer-wise supervision to learn models, making it hard to train them using standard backpropagation. More recent works use continuous memory representations or key-value pairs to read/write memories, allowing to train the memory networks end-to-end. Several works adopt the memory networks for computer vision tasks including visual question answering, one-shot learning, image generation, and video summarization. We would also discuss a memory module but for anomaly detection with a different memory updating strategy, which record various patterns of normal data to individual items in the memory, and consider each item as a prototypical feature.

2.3 Prototypes and Criticisms

A prototype is a data instance that is representative of all the data. A criticism is a data instance that is not well represented by the set of prototypes. The purpose of criticisms is to provide insights together with prototypes, especially for data points which the prototypes do not represent well. Prototypes and criticisms can be used independently from a machine learning model to describe the data, but they can also be used to create an interpretable model or to make a black box model interpretable.

The expression "data point" is used to refer to a single instance, to emphasize the interpretation that an instance is also a point in a coordinate system where each feature is a dimension. The following figure shows a simulated data distribution, with some of the instances chosen as prototypes and some as criticisms. The small points are the data, the large points the criticisms and the large squares the prototypes. The prototypes are selected to cover the centers of the data distribution and the criticisms are points in a cluster without a prototype.

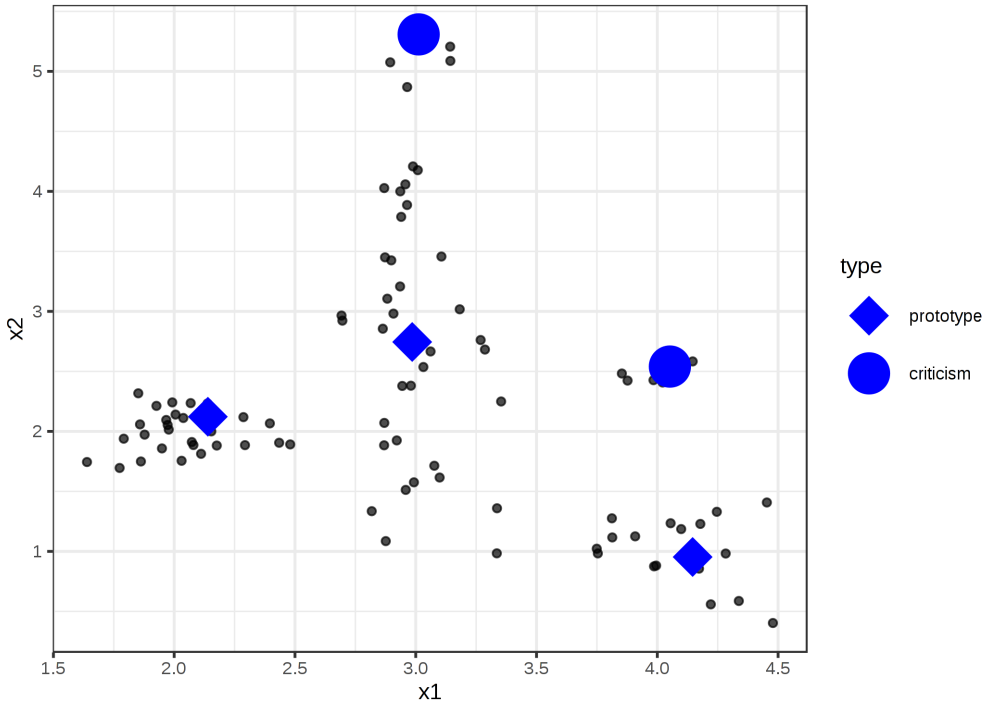


Figure 1: Prototypes and Criticisms

2.4 Few Shot Learning (FSL)

Few-shot learning is the problem of making predictions based on a limited number of samples. Few-shot learning is different from standard supervised learning. The goal of few-shot learning is not to let the model recognize the images in the training set and then generalize to the test set. Instead, the goal is to learn. "Learn to learn" sounds hard to understand. You can think of it in this way. I train the model on a big training set. The goal of training is not to know what an elephant is and what a tiger is. Instead, the goal is to know the similarity and difference between objects.

After training, you can show the two images to the model and ask whether the two are the same kind of animals. The model has similarities and differences between objects. So, the model is able to tell that the contents in the two images are the same kind of objects. Take a look at our training data again.

3 Literature Review

Recently there has been a lots of research in the field of detecting anomalous activities in videos. Most of these depend on reconstruction of a normal future frame in order to

detect anomalies, using a lot of data to train the model. Here we discuss two papers, one highlighting the prototypical model used, and another discussing the process to enable few shot learning in anomaly detection.

3.1 Learning Memory-guided Normality for Anomaly Detection

In this paper the authors tried to address the problem that most of the approach for anomaly detection do not consider the diversity of normal patterns explicitly, and the powerful representation capacity of CNNs allows to reconstruct abnormal video frames while predicting future frames. They presented an unsupervised learning approach to anomaly detection that considers the diversity of normal patterns explicitly, while lessening the representation capacity of CNNs, using memory module with a new update scheme where items in the memory record prototypical patterns of normal data. They also proposed novel feature compactness and separateness losses to train the memory, boosting the discriminative power of both memory items and deeply learned features from normal data.

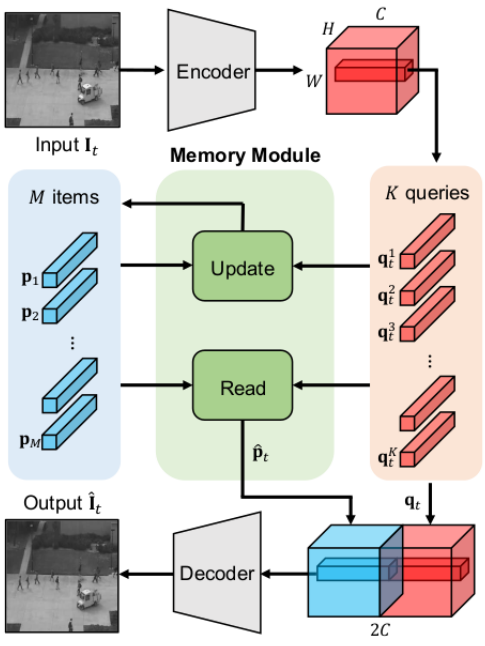


Figure 2: Overview of the framework for reconstructing a video frame.

The model mainly consists of three components: an encoder, a memory module, and a decoder. The encoder inputs a normal video frame and extracts query features. The

features are then used to retrieve prototypical normal patterns in the memory items and to update the memory. The query features and memory items aggregated (i.e., read) are feed to the decoder for reconstructing the input video frame. The model is trained using reconstruction, feature compactness, and feature separateness losses end-to-end. At test time, weighted regular score is used in order to prevent the memory from being updated by abnormal video frames.

3.2 Few-Shot Scene-Adaptive Anomaly Detection

In this paper the authors address the problem that most of the Anomaly detection models requires huge database to be trained on in order to learn normal patterns. Hence such models can be difficult to be deployed in real life, for example using anomaly detection in CCTV videos placed at different points. They propose a novel few-shot scene-adaptive anomaly detection problem to address the limitations of previous approaches. They introduce the process of Meta-Training and Meta-Testing in order to achieve few-shot learning.

3.2.1 Meta-Training

Consider a pre-trained anomaly detection model $f_\theta : x \imath y$ with parameters θ. We adapt to a task T_i by defining a loss function on the training set D_i^{tr} of this task and use one gradient update to change the parameters from θ to θ_i':

$$\theta_i' = \theta - \alpha \nabla_\theta \mathscr{L}_{T_i}(f_\theta;\ D_i^{tr}) \quad , where \tag{1}$$

$$\mathscr{L}_{T_i}(f_\theta;\ D_i^{tr}) = \sum_{(x_j, y_j) \epsilon D_i^{tr}} L(f_\theta(x_j),\ y_j) \tag{2}$$

where α is the step size. Here $L(f_\theta(x_j),\ y_j)$ is the loss function that measures the difference between the predicted frame $f_\theta(x_j)$ and the actual future frame y_j.

The updated parameters θ' are specifically adapted to the task T_i. Intuitively we would like θ' to perform on the validation set D_i^{val} of this task. We measure the performance of θ' on D_i^{val} as:

$$\mathscr{L}_{T_i}(f_\theta';\ D_i^{val}) = \sum_{(x_j, y_j) \epsilon D_i^{val}} L(f_\theta'(x_j),\ y_j) \tag{3}$$

The goal of meta-training is to learn the initial model parameters θ, so that the scene-adapted parameters θ' obtained via Eq. 1 will minimize the loss in Eq. 3 across all tasks.

We minimize the loss over all the scenes sampled.

3.2.2 Meta-Testing

After meta-training, we obtain the learned model parameters θ. During meta-testing, we are given a new target scene S_{new}. We simply use Eq. 1 to obtain the adapted parameters θ' based on K examples in S_{new}. Then we apply θ' on the remaining frames in the S_{new} to measure the performance. We use the first several frames of one video in S_{new} for adaptation and use the remaining frames for testing. This is similar to real-world settings where it is only possible to obtain the first several frames for a new camera.

3.2.3 Algorithm

Algorithm 1: Meta-training for few-shot scene-adaptive anomaly detection

Input: Hyper-parameters α, β
Initialize θ with a pre-trained model $f_\theta(\cdot)$;
while *not done* **do**
  Sample a batch of scenes $\{S_i\}_{i=1}^{N}$;
  for *each* S_i **do**
    Construct $\mathcal{T}_i = (\mathcal{D}_i^{tr}, \mathcal{D}_i^{val})$ from S_i;
    Evaluate $\bigtriangledown_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta; \mathcal{D}_i^{tr})$ in Eq. 1;
    Compute scene-adaptative parameters $\theta_i' = \theta - \alpha \bigtriangledown_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta; \mathcal{D}_i^{tr})$;
  end
  Update $\theta \leftarrow \theta - \beta \sum_{i=1}^{N} \bigtriangledown_\theta \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}; \mathcal{D}_i^{val})$ using each $\mathcal{D}_i^{val}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Eq. 3;
end

Figure 3: Meta-Training and Meta-Testing Algorithm, Source: Yiwei Lu et al. Few-Shot Scene-Adaptive Anomaly Detection

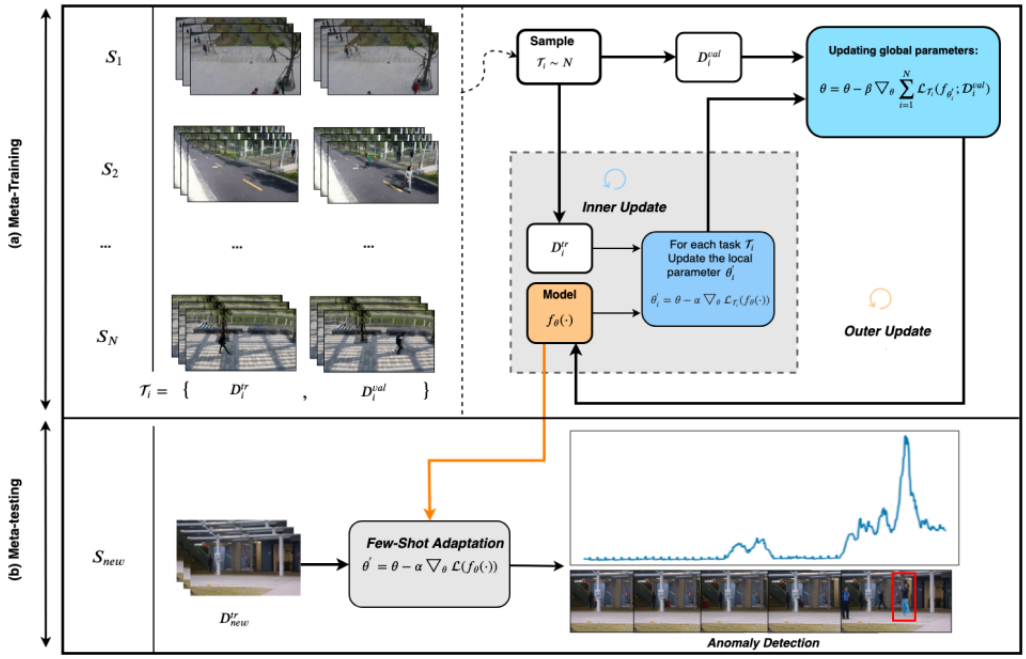


Figure 4: Flow chart of Meta-Training and Meta-Testing Algorithm, Source: Yiwei Lu et al. Few-Shot Scene-Adaptive Anomaly Detection

4 Our approach and Results

We used the Meta-Training process laid out in the above to train the prototypical memory model for anomaly detection. Hence hoping to achieve the benefits of the prototypical model using only few frames, and hence cane be used to deploy in real life.

here are some of the results we achieve using this approach on different datasets.

N	No. of iterations	AUC Score		
		USCD Ped2	CHUK Avenue	IITB Dataset
	10	91.74	79.05	67.89
4	100	72.01	82.07	70.09
	500	68.215	81.55	65.43
	10	87.308	79.45	70.08
5	100	68.95	82.86	69.65
	500	64.011	75.53	68.03
	10	85.32	80.04	70.54
5	100	88.487	82.9	66.62
	500	51.219	74.69	66.62

Table 1: AUC score on different dataset for the model trained on Shanghai dataset

5 Code Documentation [Link to Code]

This project had an important task of writing the code for the proposed approach. There were many hurdles that had to be over-come for this. In the initial attempt we tried to refactor the code provided by Yiwei Lu et al [2]. But there were many issues while running the code, hence we decided to instead take the model from Hyunjong Park et al.[1], and the write the Meta-Training and Meta-Tessting algorithms from scratch.

These were some of the challenges which were addressed:-

1. Designing a method to sample N scenes and then sample few frames from each scene in shuffled manner and without repeating.
2. Do the inner and outer update in PyTorch, without exhausting resources.
3. Handle the fact that different scenes have different number of frames available for training.
4. Make the code general and usable with any dataset.
5. Enable easy swapping of the models for the Meta-Training and Meta-Testing process.

5.1 Scene Loader

Data Loader is a helper class which takes in the path of the dataset where the videos are stored, and then creates an iterator to go over all the frames present in dataset in a shuffled order.

The original Data Loader class which was present with the Hyunjong Park et al.[1] proposed model had mant flaws. Most significantly its inability to be used for multi-scene dataset, and lack of easy way to sample scene. Here is a list of all the existing problems with the Data Loader:

1. It takes videos from all scenes into a single array and then shuffles and draws samples from it.
2. It had hard coded values for time-steps, etc.
3. It couldn't handle different starting number for frames.
4. It stores lot of unnecessary information, for e.g. path of each frame; this could be auto generated as it a sequential.

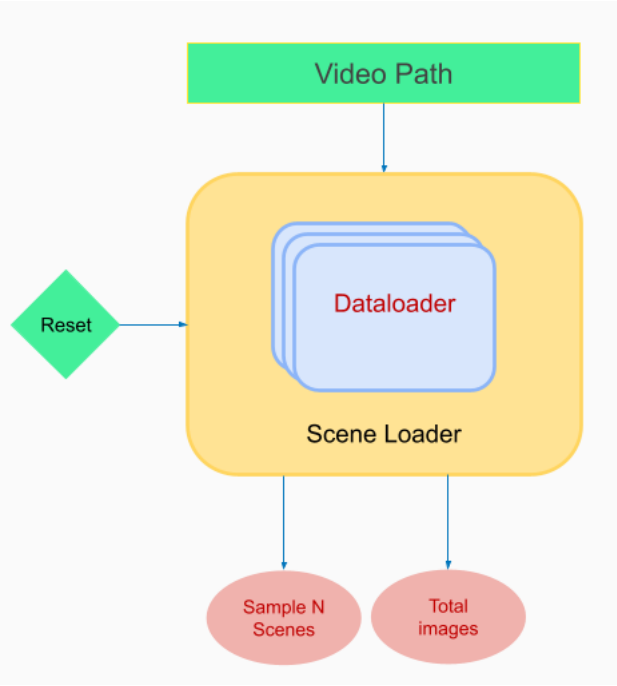


Figure 5: Scene Loader Flowchart

Scene Loader is a wrapper, over the previous DataLoader, while also improving all the problems with the previous DataLoader. It takes in the vidoe path of the dataset like the previous DataLoader, but instead creates an separate iterator for each scene in the dataset. The *SceneLoader* class contains helper functions which enables easy sampling of random samples from random scenes for each iteration of the training loop.

```
class SceneLoader(self, scenes_folder, transform, resize_height,
 ↪  resize_width, k_shots=4, time_step=4, num_pred=1, num_workers=2,
 ↪  shuffle=True, drop_last=True, single_scene=False)
```

Here are some of the helper functions and their uses:-

5.1.1 Setup Single/Multi Scene

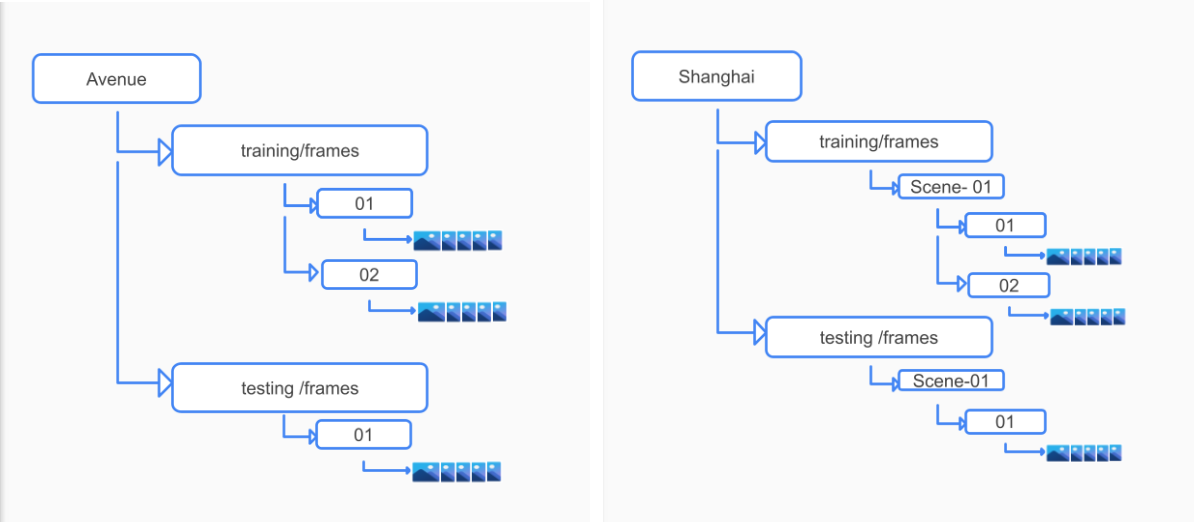
```
def setup_singlescene(self, scenes_folder, transform, resize_height,
 ↪  resize_width, k_shots, time_step, num_pred, num_workers, shuffle,
 ↪  drop_last)

def setup_multiscene(self, scenes_folder, transform, resize_height,
 ↪  resize_width, k_shots, time_step, num_pred, num_workers, shuffle,
 ↪  drop_last)
```

These are the two setup functions, which are run based on the flag which tells whether the dataset is single scene or multi scene. Input of the constructor are passed as it to these function which then appropriately sets up all the internal variables of the class.

The old Data Loader expected the data to be in the following structure as shown in the below figure. This made it hard to be used for the mutli scene datasets. Hence the Scene loader was designed to support both the structure types, which enables it to work with wide range of dataset, without changing the code.

Also there are a lot options which can be changed in the Scene Loader, Like the transforms which has to applied to the image before processing. Also scene Loader can handle video frame folders which do not start with 0000.jpg. Any sequence is valid as far as it continuous without break in between.



(a) Old Data Loader expected dataset structure **(b)** Scene Loader expected dataset structure

5.1.2 Sample N random scenes

```
def get_dataloaders_of_N_random_scenes(self, N)
```

This function samples N random scenes from the avaible array of scenes, and then result the iterator for these scene. The training and validation set then can be further drevied from these iterators.

5.1.3 Process Label List

```
def process_label_list(self, label_list)
```

This function takes in the raw label list at the time of evaluation and removes the first $T-1$ frames from each each video, where T is the time step.

5.2 Logging Module & Progress Bar

Extensive logs are saved for each step during both testing and training process. Every input variable is stored in the log for further reference. During the testing (i.e. evaluation of model), the score for each video is stored separately in a '.npy' file which then can be given to the visualization program to plot the time series plot against the video. Along with the logging, I also added progress bar and Estimated Time of Completion utilities in order to get accurate status of the training,

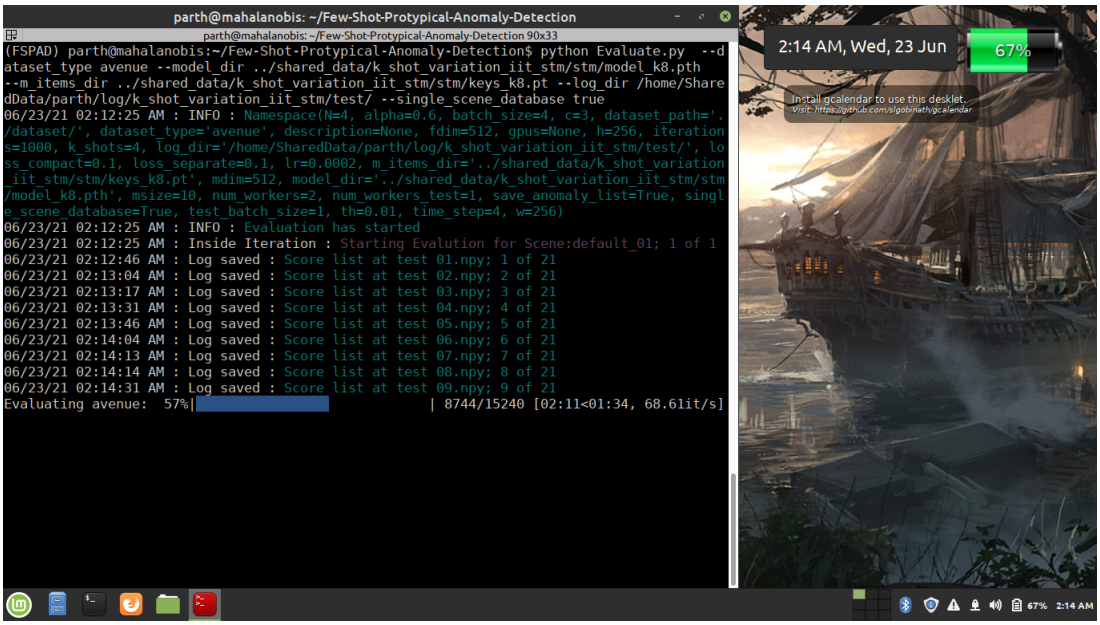


Figure 7: Logging Output

5.3 Visualization Tools

Apart from the Training and Testing, I also developed a visualization tool using Pygames. This enable me to playback the videos from continuous frames and also overlay the evaluated scores for each frame on it, in order to get a better understanding of the model.

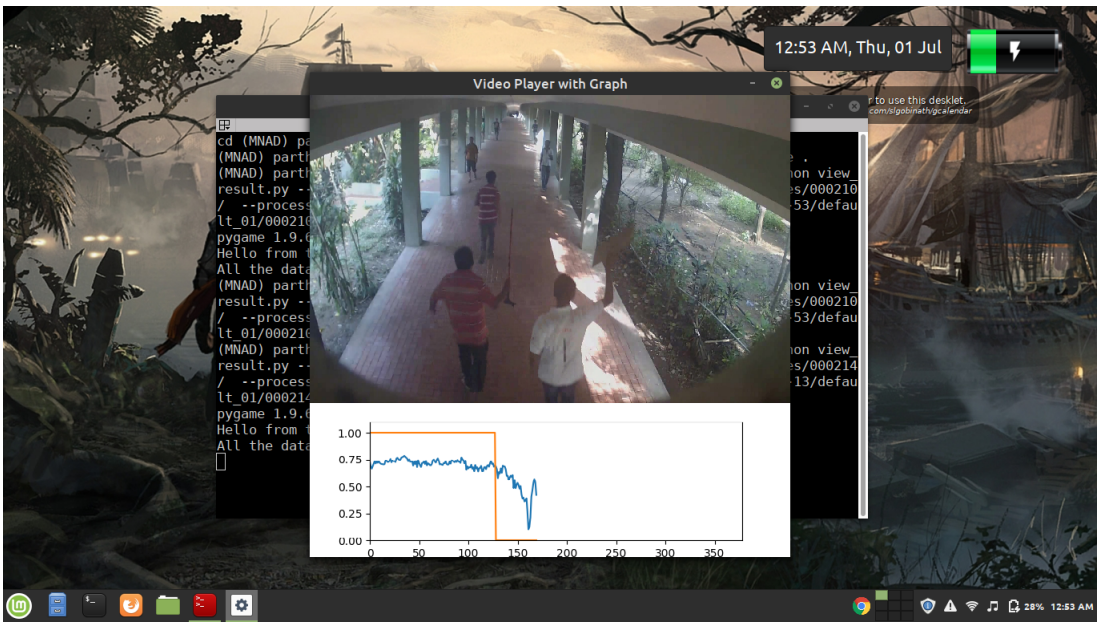


Figure 8: Visualization Output

6 References

1. Hyunjong Park, Jongyoun Noh, Bumsub Ham; Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 14372-14381

2. Yiwei Lu1, Frank Yu1, Mahesh Kumar Krishna Reddy1, and Yang Wang; Few-Shot Scene-Adaptive Anomaly Detection. 2020

3. Yashswi Jain, Anomalous Human Activity Detection, Master of Technology Thesis, 2019

4. Shreyas, D.G., Raksha, S. & Prasad, B.G. Implementation of an Anomalous Human Activity Recognition System. SN COMPUT. SCI. 1, 168 (2020). https://doi.org/10.1007/s42979-020-00169-0

5. Christoph Molnar, Interpretable Machine Learning, A Guide for Making Black Box Models Explainable. https://christophm.github.io/interpretable-ml-book/proto.html

6. Dhanya Thailappan.An Introduction to Few-Shot Learning https://www.analyticsvidhya.com/blog/2021/05/an-introduction-to-few-shot-learning/