Trespassing Detection

Aditya Nikhil, Ahmed Hasan, Amardeep Singh, Jane Pham,
Lilla Nagy, Nyandwi Jean de Dieu, Raamkishore P, Saikat Pandit

February 2020

1 Introduction

Trepassing detection is an app using OpenVINO pretrained model to detect human presence in a certain area. The model is built as a security tool to keep track of individuals who enter given premises, such as high security areas and rooms accessible only to specific personnel. Security and privacy in such matters has become quite important in order to prevent the incidence of unwanted actions such as thefts, property damage and much more.

The following model first tracks the movement and presence of individuals in the area and creates a registry at the backend. The backend is filled with details of authorized individuals who are allowed access to the area. At any point, if an unauthorized individual enters the premises, the model takes details of the same and alerts security about a possible breach in, by sending an e-mail to the user with an attached picture. In order to build the model, a dummy dataset with confidence percentage of people is created and fed into an existing OpenVino model for face and body detection.

Data for the same is contained in an xml file.

2 Theory and Literature

The problem of anomaly detection is not new, and a number of solutions have already been proposed over the years. However, before starting with the list of techniques, let's agree on a necessary premise: all anomaly detection techniques must involve a training set where no anomaly examples are encountered. The challenge consists of identifying suspicious events, even in the absence of examples.

We talk in this case of a training set formed of only "normal" events. The definition of "normal" is, of course, arbitrary. In the case of anomaly detection, a "normal" event refers just to the events represented in the training set. Here are some common approaches.

2.1 Statistical methods

Everything that falls outside of the statistical distribution calculated over the training set is considered an anomaly.

The simplest statistical method is the control chart. Here the average and standard deviation for each feature is calculated on the training set. Thresholds are then defined around the average value as `[k * std deviation]`, where k is an arbitrary coefficient, usually between 1.5 and 3.0, depending on how conservative we want the algorithm to be. During deployment, a point trespassing the thresholds in both directions is a suspicious candidate for an anomaly event.

Such methods are easy to implement and understand, fast to execute, and fit both static and time series data. However, they might be too simple to detect more subtle anomalies.

2.2 Clustering

Other proposed methods are often clustering methods. Since the anomaly class is missing from the training set, clustering algorithms might sound suitable for the task.

The concept here is clear. The algorithm creates a number of clusters on the training set. During deployment, the distance between the current data point and the clusters is calculated. If the distance is above a given threshold, the data point becomes a suspicious candidate for an anomaly event. Depending on the distance measure used and on the aggregation rules, different clustering algorithms have been designed and different clusters are created.

This approach, however, does not fit time series data since a fixed set of clusters cannot capture the evolution in time.

2.3 Supervised Machine Learning

Surprised? Supervised machine learning algorithms can also be used for anomaly detection. They would even cover all data situations since supervised machine learning techniques can be applied to static classification, as well as to time series prediction problems. However, since they all require a set of examples for all involved classes, we need a little change in perspective.

In the case of anomaly detection, a supervised machine learning model can only be trained on "normal" data — i.e., on data describing the system operating in "normal" conditions. The evaluation of whether the input data is an anomaly can only happen during deployment after the classification/prediction has been made.There are two popular approaches for anomaly detection relying on supervised learning techniques.

The first one is a neural autoassociator (or autoencoder). The autoassociator is trained to reproduce the input pattern onto the output layer. The pattern reproduction works fine as long as the input patterns are similar to the examples in the training set — i.e., "normal." Things do not work quite as well when a new, different shape vector appears at the input layer. In this case, the network

will not be able to adequately reproduce the input vector onto the output layer. If a distance is calculated between the input and the output of the network, the distance value will be higher for an anomaly rather than for a "normal" event. Again, defining a threshold on this distance measure should find the anomaly candidates. This approach works well for static data points but does not fit time series data.

The second approach uses algorithms for time series prediction. The model is trained to predict the value of the next sample based on the history of previous n samples on a training set of "normal" values. During deployment, the prediction of the next sample value will be relatively correct — i.e., close to the real sample value, if the past history comes from a system working in "normal" conditions. The predicted value will be farther from reality if the past history samples come from a system not working in "normal" conditions anymore. In this case, a distance measure calculated between the predicted sample value and the real sample value would isolate candidates for anomaly events.

3 Using Algorithms

The current algorithm uses the following components:

1. Capturing frames from the camera with a time in between
2. Each successive frame are converted to grayscale image (which is the measure of intensity of light).
3. Form these grayscale frames. We find the difference of the successive frames.
4. Later get the binary image of this differenced image.
5. We compute the change in intensity of light from this differenced images. From the value of change in intensity we can infer two things:
    - If the value peaks at some point (we can infer that a object has shown a sudden movement)
    - If the value is normal, i.e does not show any peak (we can infer that no object is moving.)
6. If the value is above threshold, the images are stored to make a not of the object which has trespassed the area under cover.

4 Basic Instructions to Run the Program

```
$ python app.py -m pedestrian-detection-adas-0002.xml -ct 0.6 -c BLUE
```

The program is in while loop. Press "ctrl+c" to break out of the program. A plot and images are the output of the program.

- Plot denotes at which frame what is the difference value (i.e the value peaks if there is a trespass in the region undercover)
- Images are stored for these peak values in the specified path.

Testing:

- The place under surveillance is left undisturbed for few seconds. If sudden movements are made in the place the images are captured.
- Tested this program for various lighting conditions in the room.
- Works fine with respect to various conditions.

5 Constraints and Difficulties

5.1 Video Feed

Naturally, to keep a look-out over a large area, we may require multiple cameras. Moreover, these cameras need to store this data somewhere; either locally, or to a remote location.

A higher quality video will take a lot more memory than a lower quality one. Moreover, an RGB input stream is 3x larger than a BW input stream. Since we can only store a finite amount of the input stream, the quality is often lowered to maximize storage. Therefore, a scalable surveillance system should be able to interpret low quality images. Hence, our Deep Learning algorithm must be trained on such low quality images as well.

5.2 Processing Power

Now that we have resolved the input constraint, we can answer a bigger question. Where do we process the data obtained from camera sources? There are two methods of doing this.

5.2.1 Processing on a centralized server

The video streams from the cameras are processed frame by frame on a remote server or a cluster. This method is robust, and enables us to reap the benefits of complex models with high accuracy. The obvious problem is latency; you need a fast Internet connection for limited delay. Moreover, if you are not using a commercial API, the server setup and maintenance costs can be high.

Memory consumption vs Inference GPU Time (milliseconds): Most high performance models consume a lot of memory.

5.2.2 Processing on the edge

By attaching a small microcontroller, we can perform real time inference on the camera itself. There is no transmission delay, and abnormalities can be reported faster than the previous method. Moreover, this is an excellent add on for bots that are mobile, so that they need not be constrained by range of WiFi/Bluetooth available. (such as microdrones).

FPS capability of various object detectors: The disadvantage is that, microcontrollers aren't as powerful as GPUs, and hence you may be forced to use models with lower accuracy. This issue can be circumvented by using on-board GPUs, but that is an expensive solution. An interesting solution would be to use software such as TensorRT, which can optimize your program for inference.

6 Image Preprocessing

When performing standard image classification, given an input image, we present it to our neural network, and we obtain a single class label and perhaps a probability associated with the class label as well.

This class label is meant to characterize the contents of the entire image, or at least the most dominant, visible contents of the image. We can thus think of image classification as:

- One image in
- And one class label out

Object detection, regardless of whether performed via deep learning or other computer vision techniques, builds on image classification and seeks to localize exactly where in the image each object appears.

When performing object detection, given an input image, we wish to obtain:

- A list of bounding boxes, or the (x, y)-coordinates for each object in an image
- The class label associated with each bounding box
- The probability/confidence score associated with each bounding box and class label

Figure 1 (right) demonstrates an example of performing deep learning object detection. Notice how both the person and the dog are localized with their bounding boxes and class labels predicted.
Therefore, object detection allows us to:

- Present one image to the network
- And obtain multiple bounding boxes and class labels out

The first method is not a pure end-to-end deep learning object detector. We instead utilize:

- Fixed size sliding windows, which slide from left-to-right and top-to-bottom to localize objects at different locations
- An image pyramid to detect objects at varying scales
- Classification via a pre-trained (classification) Convolutional Neural Network

At each stop of the sliding window + image pyramid, we extract the ROI (Region of Interest), feed it into a CNN, and obtain the output classification for the ROI.

If the classification probability of label L is higher than some threshold T, we mark the bounding box of the ROI as the label (L). Repeating this process for every stop of the sliding window and image pyramid, we obtain the output object detectors. Finally, we apply non-maxima suppression to the bounding boxes yielding our final output detections.

The second method to deep learning object detection allows you to treat your pre-trained classification network as a base network in a deep learning object detection framework (such as Faster R-CNN, SSD, or YOLO).

The benefit here is that you can create a complete end-to-end deep learning-based object detector.

The downside is that it requires a bit of intimate knowledge on how deep learning object detectors work — we'll discuss this more in the following section.

There are many components, sub-components, and sub-sub-components of a deep learning object detector, but the two we are going to focus on today are the two that most readers new to deep learning object detection often confuse:

- The object detection framework (ex. Faster R-CNN, SSD, YOLO).
- The base network which fits into the object detection framework.

You are likely already familiar with the base network,you just haven't heard it referenced as a "base network" before.

Base networks are your common (classification) CNN architectures, including:

- VGGNet
- ResNet
- MobileNet
- DenseNet

Typically these networks are pre-trained to perform classification on a large image dataset, such as ImageNet, to learn a rich set of discerning, discriminating filters.

Object detection frameworks consist of many components and sub-components.For example, the Faster R-CNN framework includes:

- The Region Proposal Network (RPN)
- A set of anchors
- The Region of Interest (ROI) pooling module
- The final Region-based Convolutional Neural Network

When using Single Shot Detectors (SSDs) you have components and sub-components such as:

- MultiBox
- Priors
- Fixed priors

7 Set Up Edge Device

7.1 Hardware Requirements

1. Raspberry Pi 3b + or higher
2. Intel NC2
3. Pi Camera
4. Charger
5. Keyboard, mouse, screen monitor for the initial setup
6. Optional: cases, heatsink, fan, etc.

7.2 Software Setup

7.2.1 Setting up the Raspberry and install the OS and python

- Follow this guide to setup the OS: https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up
- Update the software using sudo apt get or yum install (depends on the OS)
- Install Python 3
- Optional: setup ssh to monitor your device remotely

7.2.2 Setup Openvino enviroment for ML and computer vision

- follow this guide to install OpenVino on Pi: https://docs.openvinotoolkit.org

7.2.3 Ultilize Intel NC2 and start the model training

- install cmake using
  `$ sudo apt-get install cmake`

8 To Run the Program

1. Download all the files in the folder named "Complete folder to run the program". Have all these files in the same folder.
2. Install and setup OpenVINO environment to run the program. In the `send_mail.py` file, put your email id as a parameter where the *main()* function is called.
3. Run the below mentioned command in the shell to obtain results.

   ```
   python app.py -m pedestrian-detection-adas-0002.xml -ct 0.6 -c
   BLUE
   ```

4. As of now, you can pass a video file as input to detect trespassers. In real time, we can use camera to detect trespassers. Also, we can have an alarm system to alert security guards around premises.

9 Obtained Results

If a person is detected in a prohibited area, an alert mail is sent immediately to the concerned team with the image of the trespasser attached.