In a world becoming ever more attuned to potential security threats, the need to deploy sophisticated surveillance systems is increasing. An intellectual system that functions as an intuitive “robotic eye” for accurate, real-time detection of unattended baggage has become a critical need for security personnel at airports, stations, malls, and in other public areas. This article discusses inferencing a Microsoft Common Objects in Context (MS-COCO) detection model for detecting unattended baggage in a train station.

1. Evolution of Object Detection Algorithms

Image classification involves predicting the label of an image among predefined labels. This assumes that there is a single object of interest in the image and it covers a significant portion of the image. Detection is about not only finding the class of the object but also localizing the extent of the object in the image. The object can be lying anywhere in the image and can be any size (scale). So object classification is not helpful when there are multiple objects in an image, the objects are small, and the exact location and image are desired.

Traditional methods of detection involved using a block-wise orientation histogram (SIFT or HOG) feature that could not achieve high accuracy in standard data sets such as PASCAL VOC. These methods encode low-level characteristics of the objects and therefore cannot effectively distinguish among the different labels. Methods based on deep learning (convolutional networks) have become the state-of-the-art in object detection in images. Various network topologies have evolved over time, as shown in Figure 1.
2. Installation

2.1 Building and Installing Caffe* Optimized for Intel® Architecture

Caffe can be installed and used with several combinations of development tools and libraries on a variety of platforms. Here we describe the steps to build and install Caffe* optimized for Intel® architecture with the Intel® Math Kernel Library 2017 on Ubuntu*-based systems. Please refer to the git* clone https://github.com/intel/caffe.

1. Clone the Caffe optimized for Intel architecture and pull down all the dependencies.

Navigate to the local caffe directory and copy the makefile.config.example and rename it to makefile.config.

2. Make sure the following lines are uncommented in the makefile.config.

```plaintext
# BLAS choice:
# atlas for ATLAS (default)
# mkl for MKL
# open for OpenBlas
BLAS ::= mkl

# CPU-only switch (uncomment to build without GPU support)
CPU_ONLY := 1
```

3. Install OpenCV*.

For computer vision and image augmentation, install the OpenCV 3.2 version.

```bash
sudo apt-get install python-opencv
```

Remember to enable OPENCV_VERSION := 3 in Makefile.config before running make when using OpenCV 3 or higher.

4. Build the local Caffe.

Navigate to the local caffe directory.

```bash
NUM_THREADS=41
make -j $NUM_THREADS
```

5. Install and load the Python* modules.

```bash
make pycaffe
pip install pandas
pip install scipy
import sys
CAFFE_ROOT = 'path/to/caffe'
sys.path.append(CAFFE_ROOT)
import caffe
caffe.set_mode_cpu()
```

3. Solution Architecture and Design

Our solution aims at identifying unattended baggage in public areas like railway stations, airports and so on and then triggering an alarm. Detections are done in surveillance videos using the business rules defined in section 3.3. Network Topology

Of the different detection techniques mentioned in Figure 1, we decided to choose the Single Shot multibox Detector (SSD) optimized for Intel architecture [2]. Researchers say that it has promising performance even in embedded systems and high-end devices and hence is likely to be used for real-time detections.

![Figure 2. Input Image and Feature Maps](image)
SSL only needs an input image and ground truth (GT) boxes for each object during training. In a convolutional fashion, a small set (four, in our example) of default boxes of different aspect ratios at each location in several feature maps with different scales \([8 \times 8 \text{ and } 4 \times 4\) in (8) and (c)] is evaluated (see Figure 2). The SSL leverages the Faster R-CNN [3] Region Proposal Network (RPN) [4], using it to directly classify object inside each prior box instead of just scoring the object confidence.

For each default box, the network predicts both the shape offsets and the confidences for all object categories \([c_1, c_2, ..., c_p]\). At training time, the default boxes are first matched to the ground truth boxes. For example, two default boxes—one with a cat and one with a dog—are matched, which are treated as positives. The boxes other than default boxes are treated as negatives. The model loss is a weighted sum between localization loss (example: Smooth L1) and confidence loss (example: Softmax).

Since our use case involves baggage detection, either the SSL network needs to be trained with different kinds of baggage or we can use a pretrained model like SSD300 trained on an MS-COCO data set. We decided to use the pretrained model, which is available for download at https://github.com/weiliu89/caffe/tree/ssd#models

3.1 Design and Scope

The scope of this use case limited to the detection of baggage that stays un-attended for a period of time. Identifying the exact owner and tracking the baggage is beyond the scope of this use case.

Because of the large number of boxes generated during the model inference, it is essential to perform non-maximum suppression (NMS) efficiently during inference. By using a confidence threshold of 0.01, most boxes can be filtered out. The NMS can then be applied with a Jaccard overlap of 0.45 per class, keeping the top 400 detections per image. Figure 3 shows the flow diagram for running detection on a surveillance video.

![Figure 3. Detection flow diagram.](image)

The surveillance video is broken down into frames using OpenCV with a configurable frames per second. As the frames are generated, they are passed to the detection model, which localizes the different objects in the form of four coordinates \((x_{min}, x_{max}, y_{min}, y_{max})\) and provides a classification score to the different possible objects. By applying the NMS threshold and setting confidence thresholds, the number of predictions can be reduced and kept to the prediction that is the most optimal. OpenCV is used to draw a rectangular box with various colors around the detected baggage and the person.

3.2 Defining the Business Rules

Abandoned luggage in our context is defined as items of luggage that have been abandoned by their owner. Each item of luggage has one owner, and each owns at most one item of luggage. Luggage is defined as including all types of baggage that can be carried by hand. Examples: trunks, bags, rucksacks, backpacks, parcels, and suitcases.

The following rules apply to attended and unattended luggage:

- A luggage is owned and attended to by a person who enters the scene with the luggage until the point at which the luggage is not in physical contact with the person.
- At this point the luggage is attended to by the owner ONLY when they are within a distance of 20 inches (spatial rule). All distances are measured between Euclidean distances.
- A luggage item is unattended when the owner is farther than \(b\) meters (where \(b \geq a\) from the luggage. In this case the system applies the spacio-temporal rule to detect whether this item of luggage has been abandoned (triggering an alarm event).
- The spacio-temporal rule to determine abandonment: an item of luggage that has been left unattended by its owner for a period of \(t\) consecutive seconds during which time the owner has not re-attended to the luggage nor has the luggage been attended to by a second party (instigated by physical contact, in which case a theft/tampering event may be raised). The image below (Figure 7) shows an item of luggage left unattended for \(t =10\) seconds, at which point the alarm event...
is triggered. Here we relate the time $t$ with the number of frames $f$ per second. If we have $n$ frames per second in our input video, $t$ seconds would be defined as $(t*f)$ frames. In short, a bag that has been unattended in $(t*f)$ consecutive frames triggers the alarm.

### 3.3 Inferencing the MS-COCO Model

Implementation or inferencing is done using Python 2.7.6 and OpenCV 3.2. The following steps are performed (code snippets are included for reference):

1. **Read the input video as follows:**

   ```python
   CAFFE_ROOT = '/home/979648/SSD/caffe/
   # -> Reading the video file and storing in a directory
   TEST_VIDEO = cv2.VideoCapture(os.getcwd() +
   'InputVideo/SurveillanceVideo.avi')
   MODEL_DEF = 'deploy.prototxt'
   CAFFE_ROOT = '/home/979648/SSD/caffe/
   # -> Reading the video file and storing in a directory
   TEST_VIDEO = cv2.VideoCapture(os.getcwd() +
   'InputVideo/SurveillanceVideo.avi')
   MODEL_DEF = 'deploy.prototxt'
   ```

2. **Load the network architecture.**

   ```python
   net = caffe.Net(MODEL_DEF, MODEL_WEIGHTS, caffe.TEST)
   ```

3. **Read the video by frame and inference each frame against the model to obtain a detection and classification score.**

   ```python
   success, image = TEST_VIDEO.read()
   if (success):
       refObj = None
       imageToNet = cv2.resize(image, (300, 300))
       image_convert = np.swapaxes(np.swapaxes(imageToNet, 1, 2), 0, 1)
       net.blobs['data'].data[...] = image_convert
       # Forward pass.
       detections = net.forward()['detection_out']

       # Parse the outputs.
       det_label = detections[0, 0, :, 1]
       det_conf = detections[0, 0, :, 2]
       det_xmin = detections[0, 0, :, 3]
       det_ymin = detections[0, 0, :, 4]
       det_xmax = detections[0, 0, :, 5]
       det_ymax = detections[0, 0, :, 6]

       # Get detections with confidence higher than 0.6.
       top_indices = [i for i, conf in enumerate(det_conf) if conf >= CONFIDENCE]
       top_conf = det_conf[top_indices]
       top_label_indices = det_label[top_indices].tolist()
       top_labels = get_labelname(labelmap, top_label_indices)
       top_xmin = det_xmin[top_indices]
       top_ymin = det_ymin[top_indices]
       top_xmax = det_xmax[top_indices]
       top_ymax = det_ymax[top_indices]

       colors = plt.cm.hsv(np.linspace(0, 1, 21)).tolist()
       currentAxis = plt.gca()

       # print('Detected Size : ', top_conf.shape[0])
       detectionDF = pd.DataFrame()
   if (top_conf.shape[0] != 0):
       for i in xrange(top_conf.shape[0]):
           xmin = int(round(top_xmin[i] * image.shape[1]))
           ymin = int(round(top_ymin[i] * image.shape[0]))
           xmax = int(round(top_xmax[i] * image.shape[1]))
           ymax = int(round(top_ymax[i] * image.shape[0]))
           score = top_conf[i]
           label = int(top_label_indices[i])
           label_name = top_labels[i]
           display_txt = '%s: %.2f' % (label_name, score)
           detectionDF = detectionDF.append(
               {'label_name': label_name, 'score': score, 'xmin': xmin, 'ymin': ymin, 'xmax': xmax, 'ymax': ymax},
               ignore_index=True)
       detectionDF = detectionDF.sort('score', ascending=False)
   ```
4. For calculating the distance between objects in an image, a reference object has to be used. A reference object has two main properties:

a) The dimensions of the object in some measurable unit, such as inches or millimeters. In this case we consider the dimensions to be in inches.

b) We can easily find and identify the reference object in our image.

Also, an approximate width of the reference object has to be assumed. In this case we assume the width of the suitcase (args['width']) to be 27 inches.

```python
pip install imutils
if refObj is None:
    # unpack the ordered bounding box, then compute the
    # midpoint between the top-left and top-right points,
    # followed by the midpoint between the top-right and
    # bottom-right
    (tl, tr, bl, br) = box
    (tlblX, tlblY) = midpoint(tl, bl)
    (trbrX, trbrY) = midpoint(tr, br)

    # compute the Euclidean distance between the midpoints,
    # then construct the reference object
    D = dist.euclidean((tlblX, tlblY), (trbrX, trbrY))
    refObj = (box, (cX, cY), D / args['width'])
    continue

5. Once the reference object is obtained, the distance between the reference object and the other objects in the image is calculated. The business rules are applied, and then the appropriate alarm will be triggered. In this case, a red box will be highlighted on the object.

```python
if refObj != None:
    D = dist.euclidean((objBotX, objBotY), (int(tlblX), int(tlblY))) / refObj[2]
    (mX, mY) = midpoint((objBotX, objBotY), (tlblX, tlblY))

    //apply spacio temporal rule
    // Highlight with Green/Yellow /Red
```

6. Save the processed images, and then append them to the output video.

4. Experimental Results

The following detection (see Figures 4—7) was obtained when the inference use case was run on a sample YouTube* video available at https://www.youtube.com/watch?v=fpTG4ELZ3bE

![Figure 4](image1.png)  ![Figure 5](image2.png)

Figure 4: Person enters the scene with the baggage, which is currently safe (highlighted with green).

Figure 5: The owner is moving away from the baggage.

![Figure 6](image3.png)

Figure 6: The system raises a warning signal.
5. Conclusion and Future Work

We observed that the system can detect baggage accurately in medium- to high-quality images. The system is also capable of detecting more than one baggage in the case of multiple owners. However, the system failed to detect the baggage in a low-quality video. The distance calculation does not include focal length, angle of the camera, and the plane, and hence the current calculation logic has its own limitations. The current system is also not capable of tracking the baggage.

The model was inferenced using the Intel® Xeon® processor E5-2699 v4 @ 2.20 GHz with 22 cores and 64 GB free memory. Future work will include enhancement to the current use case by identifying the owner of the baggage and also tracking the baggage. Videos with different angles and focal lengths will also be inferenced to judge the effectiveness of the system. The next phases of our work will also consider efforts to parallelize the inference model.

6. References and Links

- SPPNet 2014
- MultiBox, 2014
- Fast RCNN 2015
- YOLO, 2015
- Faster RCNN 2015
- SSD, 2016
- Mask R-CNN
- Single shot multibox detector (SSD)
- Faster R-CNN: Toward Real-Time Object Detection with Region Proposal Networks
- https://medium.com/@phelixlau/speed-accuracy-trade-offs-for-modern-convolutional-object-detectors-bbad4e4e0718
- https://github.com/Nikasa1889/HistoryObjectRecognition