Implementation of a Person Counting Algorithm

GROUP 8 - ECSE415

Karine Mellata, *260741673* Jacob Hochtrasser, *260685632*
Melissa Hawley, *260730658* Pierre Robert-Michon, *260712449*

I. PART 1: DESCRIPTION OF PERSON DETECTOR

For part 1, we had chosen to use the YOLO (You Only Look Once) algorithm as the existing person detection technique. The OpenCV method was obtained from a tutorial on PythonCode [1]. YOLO is a real-time object detection algorithm using a deep convolutional neural network. The method works by splitting the input image into a set of grid cells and each grid cell has an associated vector in the output that tells us if there is an object in the grid cell, the class of that object and the predicted bounding box for that object.

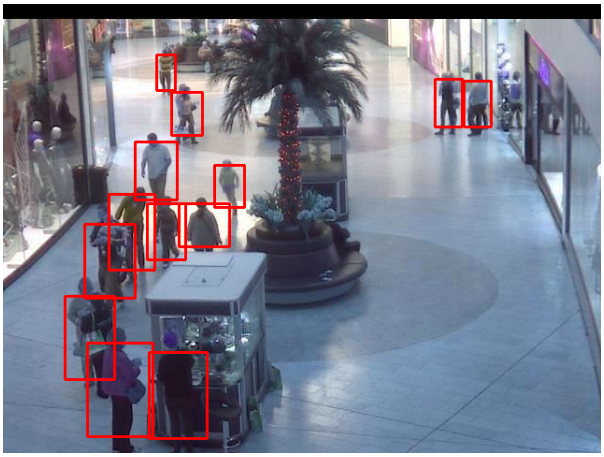

Fig. 1. Person detection using YOLO

II. PART 2: DESCRIPTION OF PERSON DETECTOR

A. Training Set Distribution

The second person detector algorithm was trained using rectangles of people detected from sample images using the YOLO algorithm in part 1. Class "0" represented images containing no person and class "1" represented images containing people. Using the YOLO algorithm, 31,153 persons were detected from the 2000 sample frames, which incidentally yields 31,153 sample images of class 1.

As expected, these 31,153 images were of various sizes, but needed to all be resized to the same dimensions in order to work with an SVM and sliding-window based detector. To achieve an optimal size which does not distort the majority of samples, the distributions of heights, widths and height/width ratios were calculated, as shown in Figure 2, over 250 samples. The mean height is 61.44, the mean width is 42.63 and the mean ratio is 1.54. Using this data, we chose to resize all images to a size of 60x40.

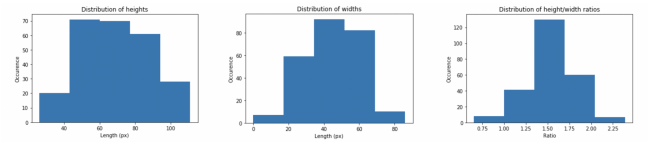

Fig. 2. Distributions of heights, widths and height/width ratios of a sample of 250 person boxes

Negative examples were taken by extracting randomly located patches of size 60x40 that do not intersect patches with the "people" detected in the frames. Using this method, 163,828 images were generated. However, this number yields a highly unbalanced data set, which is why only 30,000 images were kept in order to have an even data set[1]. Figure 3 shows sample images taken from the training set.

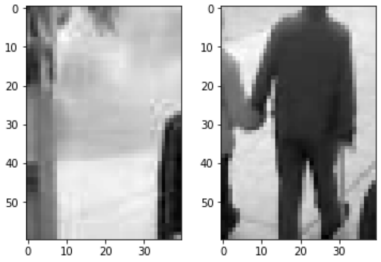

Fig. 3. Example of a negative sample (left) and a positive sample (right)

B. Features Pre-processing

A HoG (Histogram of Oriented Gradients) approach was used to generate the features to train the SVM. According to Dalal et Al., HoG descriptors significantly outperformed other feature descriptors when training an SVM-based human detector [2]. This method captures a gradient structure in a localized portion of the image that is very characteristic of the local shape, and it does so with a degree of invariance to local rotation and translation transformations. This is a particularly strong advantage for human detection as it allows for good performance although limbs and body segments change in appearance and move side-to-side [2]. This can be

[1] Although intuitively "negative" patches happen more often than "positive" patches in a frame, the SVM-based detector seemed to generate a lot of false negatives with the initial data set and yielded significantly better results using a balanced data set.

vaguely recognized in Figure 4, where a shape of a walking body is outlined in the HoG image.

The images were first all converted to grayscale in order to use this approach. The HoG features were computed using `scikit-learn`'s implementation with 9 orientation bins, cells of 8×8 pixels and blocks of 2x2 cells.

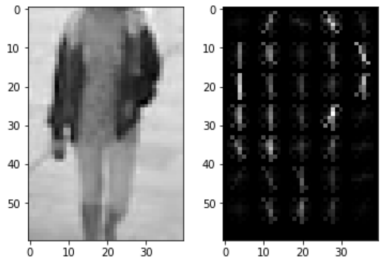

Fig. 4. Example of a positive sample (left) and its computed HoG image (right)

C. Support Vector Machine Detector

A non-linear SVM, implemented by `scikit-learn`, was used to classify negatives and positives. However, as the fitting process time complexity scales at best quadratically with the number of samples, the documentation stated that the implementation could not scale well beyond tens of thousands of samples [3]. This is why only 17% of the data set was used as a training set and 2% as a test set. After running a hyper-parameter search on the C value, an optimal value of `C=100` was found which yields an accuracy of 97.37%.

III. Description of the duplicate detection removal and person counting

A. Sliding-Window Approach

After the SVM was trained, we used a sliding window to count the number of people within a frame. The size of the sliding window was 60x40, and the window was analyzed by the SVM with jumps of 20 pixels over the y-axis and 10 pixels over x-axis [2].

In order to remove duplicates after detection, a confidence score was required for each detected box. As our classification was binary, this confidence score is rather straight-forward: it is the distance between the hyperplane and the data point. In other words, if the distance between the hyperplane and the data point is high, we have a higher confidence that this datapoint belongs in the predicted class. If it is very close to the decision boundary, we have a lower confidence of which class it belongs to. Given the `decision_function()` function implementation by `scikit-learn`, a negative number means a prediction of class 0 (no person detected) and a positive number means a class 1 (a person detected) and the sign of this confidence score indicates the side of the hyperplane on which the data point lies.

Once each "detected person box" was associated to a confidence score, it was possible to apply non-maximum suppression in order to remove overlapping duplicate boxes. This technique uses the highest confidence box and an "overlap" parameter which decides which degree of overlap with the highest confidence box constitutes another box being a duplicate. We used an overlap threshold (IoU) of 0.1. In other words, our algorithm did not allow for much overlap. A score threshold (which discards boxes with too low of a confidence score) of 1.0 was also used.

IV. Evaluation of Person Detector

The person detector algorithm yielded an accuracy of 97.37%. According to Figure 5, we get a similar amount of false positives as false negatives. Therefore, our error comes almost equally from missing "persons" as from falsely detecting "persons".

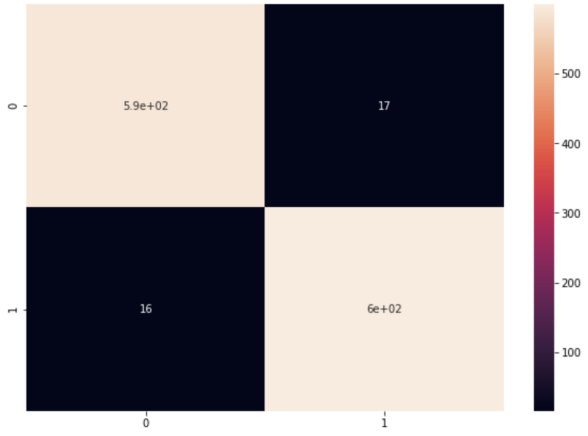

Fig. 5. Confusion Matrix for SVM Person Detector

V. Evaluation of Person Counting

When calculating the accuracy of our person counting algorithm using the YOLO results as our ground truth, we find a very low accuracy of 6.55%. This low number is due to multiple reasons. Firstly, when evaluating the YOLO results against the "ground truth" (Kaggle competition), we get a weighted mean error of 35.89%. This means that the YOLO algorithm implemented in the first place does not detect people perfectly, and from manual analysis, we have found out that this is because people are often missed by YOLO (high number of false negatives). This factor did not matter to creating our own person counting algorithm as we only needed "positive" boxes from the YOLO algorithm and as long as the positive boxes were indeed people, our person detector algorithm could be trained and did not necessarily need "all" persons. In other words, a high number of false negatives in Part 1 did not affect Part 2, but a high number of false positives would have.

Now, when evaluating our own person counting algorithm

[2]The jumps were included as the method took an extremely long time to run otherwise.

Fig. 6. Detection performance for IoU thresholds of 0.1, 0.5 and 0.9 respectively (top to bottom)

against the Kaggle ground truth, we found a weighted mean error of 17.45% (an accuracy of 82.55%) placing us at 9th position [3].

One last metric that we calculated was the average IoU of our samples (the average intersection of overlap per frame), which was equal to 63.16%. The IoU is calculated using the formula below,

$$IoU = \frac{TP}{TP + FP + FN} \tag{1}$$

The true positives (TP) is the number of correctly classified pixels, false positives (FP) is the number of pixels wrongly classified and false negatives (FN) is the number of pixels wrongly not classifed. The "ground truth" for these calculations were our YOLO results.

VI. Discussion

As discussed in the results, there are many ways our method could have been improved.

[3]We submitted a version with a small bug which yielded an initial accuracy closer to 39% and placed us at 15th position. The submission at 17.45% is considered a late submission.

Fig. 7. Example of persons detected by our person counting algorithm

A. False Positives

The first thing that was noticed is that our algorithm was fairly good at detecting persons but was very sensitive to false positives. Although the initial person detector did not indicate that our detection was particularly sensitive to false positives (see Figure 5), the final person counting approach seemed to pick up on a lot of random negatives. To counter this, we have included a confidence score threshold in the person counting approach, where detections with a score lower than 1 are not kept by the non-maximum suppression algorithm. A threshold higher than this one began to discard true positives. Also, the IoU threshold of NMS decreased the number false positives greatly, as can be seen in Figure 6. In addition, as our person detector algorithm yields a high accuracy (97%), we suspect our feature descriptor could not have been the cause.

B. False Negatives

There are also two major assumptions that we are currently making in our approach that are potentially false for many positives:

- The window size is fixed at a size of 60x40
- The windows analyzed are 20px apart on the Y-axis and 10px apart on the X-axis

These assumptions stem from the fact that we have resized all images to 60x40, therefore we only detect persons within a 60x40 window, and when running our sliding-window approach, we make jumps of 20px on the Y-axis and 10px on the X-axis. These estimates may very well explain the false negatives, where a person of smaller or larger size, which gets "cropped" by the sliding window approach, is not detected.

C. Difficulties

The main difficulties that we encountered were when implementing the person counting approach. As the images are of size 480x640, the sliding window needs to compute $(480/20) * (640/10) = 1536$ predictions per frame. In addition, each prediction includes the pre-processing step of computing the HoG feature descriptor. Therefore, this part was very long to compute and therefore very hard to debug.

D. Possible Different Approaches

As for a general pre-existing approach to estimate the number of people in an image, we decided to implement YOLO as the existing person detection technique as its full end-to-end training makes it appear as a ”cleaner” way of doing object detection when compared to Faster-RCNN for example. However, YOLO has some drawbacks. When detecting objects that are small and in close proximity to each other, it has difficulty as there are only two anchor boxes in a grid predicting one class of objects. It also has difficulty generalizing well when objects in the image show uncommon aspect ratios. Faster-RCNN does a better job at detecting small objects since it has nine anchors in a single grid. Another option could have been Fast-RCNN or SSD (Single Shot Detector). As shown in Figure 8, when comparing the algorithms mentioned, the one that obtains the best accuracy results is Faster-RCNN, although it is also one of the slowest [4]. The accuracy of these algorithms also depends on the size of objects being detected as the larger the objects, the larger the difference in accuracy between algorithms as well. So, in our case, switching from YOLO to Faster-RCNN would make a difference in accuracy results, although it may not have been a very large difference.

Fig. 8. Comparison of Accuracy vs. speed of YOLO, Faster-RCNN, Fast-RCNN and SSD [4]

REFERENCES

[1] Abdou Rockikz. How to perform yolo object detection using opencv and pytorch in python, Jan 2020.

[2] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, 2005.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[4] Zero to hero: Guide to object detection using deep learning: Faster r-cnn,yolo,ssd, Dec 2017.