Modern Convolutional Object Detector Under Security-related Usage

Shantao Yi
Department of Computer Science and Engineering
University of California San Diego
La Jolla, CA 92092
shy146@ucsd.edu

Cheng Gong
Department of Computer Science and Engineering
University of California San Diego
La Jolla, CA 92092
c5gong@ucsd.edu

Abstract

The full potential of recent advances in Computer Vision has not been fully exploited in the field of physical security yet. Object detection, for example, can greatly reduce the resource and time put into menial tasks performed at airport security checks. In this article, we propose an implementation of a baggage detection algorithm using start-of-the-art object detectors as baseline, and achieved improvements via fine-tuning.

1 Introduction

Object detection is a topic under the field of Computer Vision that allows the computer to distinguish and recognize object instances within images or videos. Nowadays, many industries have already applied object detection in computer vision applications, such as face recognition and autonomous driving. Compared to simple image classifications, object detection has the more detailed benefit of knowing the exact location, size, rotation, and count of different objects, creating exponential possibilities of previously unexplored use cases.

Since many other fields have already integrated such utilization, we want to highlight the field of security, a mostly neglected field with valuable usage potential. As a matter of life and death, the consequence of security is unquestionable around the globe. When looking into solutions of this issue, we notice the common use of baggage in committing crimes. In both civic and international crimes, no matter if it is illicit object trafficking, or weapon or bomb usage for military attacks, perpetrators need containers to conceal and carry malicious items. Therefore, it is of paramount importance to be able to swiftly and accurately identify containers in security photos or a real-time video footage. These problems, however, may be able to be solved by object detection algorithms. Therefore, we proposed and fine-tuned a baggage detection algorithm that will trigger at the sight of any possible briefcases, handbags or anything of the sort (In our current implementation, we focused solely on backpacks).

To list a few of its benefits, our algorithm will lead to automate and expedite the security process, minimizing human errors and engage human only when necessary. At the security gate of airports, high-end hotels, large conventions, our algorithm will accurately and promptly identify baggage, given the pre-installation of multi-angle cameras. With an integrated security system, we can prevent

Preprint. Under review.

the entrance of people if they have not put their baggage through the X-rays scanner, for example. Only then can the persons walk through the metal detectors. The algorithm will also be useful in surroundings surveillance. With camera parameter set up, guards will be alerted almost instantly when suspicious persons carrying bags walk into the parameter. This has huge civil police uses and some military uses, in addition to the human-detecting thermal imaging.

2 Related Work

2.1 YOLO: You Only Look Once[3]

Other region-proposal based frameworks detect objects by re-using classifiers on the local regions of the images. YOLO treats this as a regression problem and uses a single-shot Neural Network to predict the bounding boxes and the class probabilities. The prediction of the bounding boxes on one input image are done through looking at the features of the whole image. By design, in back-propagation, the neural network can learn conveniently about object co-occurrence, relation between the sizes and locations of different objects, etc.

YOLO divides the image into an S by S grid, and each grid cell gives a conditional probability distribution of the classes we care about. However, each grid is responsible for predicting bounding boxes for the class with the highest probability. Thus, regardless of the grid size, every grid can only predict one object.

Since each grid cell only predicts a few (2 in YOLOv1) boxes and can only predict one class, the limited spatial usage of the bounding box of YOLO limits the number of adjacent objects able to predict. Unlike Faster R-CNN, which has no constraints where the proposed regions can be, YOLO struggles with small objects that appear in groups, such as flocks of birds.

On the bright side, the spatial constraints on the grid cell proposals of YOLO helps mitigate multiple detections of the same object, and produces fewer bounding boxes to process, only 98 per image compared to about 2000 from Selective Search in R-CNN and Fast R-CNN, but not in Faster R-CNN.

2.2 Faster R-CNN[2]

Faster R-CNN is the state-of-the-art region-based CNN network. To provide some background, CNNs before R-CNN mainly use sliding windows to generate regions to look at individually with CNN-classifiers, producing a set of probabilities for all the possible object classes. A general region-based CNN has the same approach, but avoiding to select a huge number of regions to examine by using Selective Search. This Selective Search independently generates about 2000 regions of interest.

Instead of using Selective Search, Faster R-CNN utilizes CNN to generate convolutional feature map directly from the image. The feature map is then feed into a Region Proposal Network to generate regions of interest. Then the regions are reshaped into the same size with a RoI pooling layer. Lastly both the regions of the same shape and the feature map are feed into a classifier, where it classifies the image within the region and provide the region offsets.

3 Methodology

Almost all state-of-the-art object detection algorithms are performing "comprehensive detection", that is, given an image or video clip, the algorithm will try to detect, locate, and name all possible instances that appear in it. However, when facing a specific task, most of the information the algorithm provides will not be useful. Not to mention that comprehensive detection algorithms needs multi-class detection and classification, which requires huge amount of training data covering different instances, as well as tremendous amount of resources and time for training.

For our project, we will develop a task-specific, single-class detector designed exclusively for detecting luggages under specific circumstances, real-life security camera footage snapshots in specific. To achieve that, we will first pick a high performance CNN structure from industry-standard object detectors, and our choice here will be darknet from YOLO. The primary advantage for darknet is its speed factor. Real-life security problems usually deal with monitoring via security cameras, which requires real-time computation results when detection algorithm is applied. Darknet carries

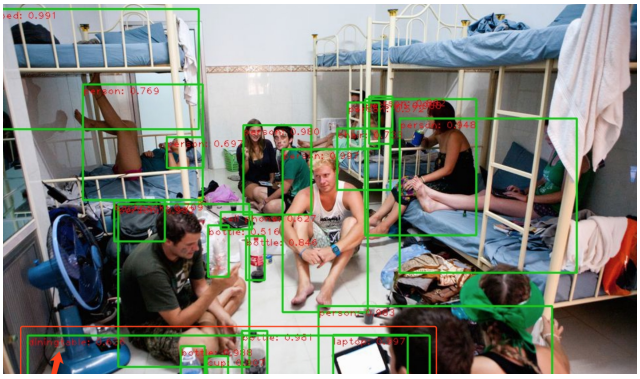

Figure 1: Comprehensive detection example (massive and crowded information pool)

the reputation of such speed while still being able to provide relatively good accuracy. Therefore, we used the state-of-the-art implementation of YOLOv3, adapting its configuration for single-class detection. Then, we trained and fine-tuned our detector iteratively to increase its performance. We test the model over our own handpicked custom dataset, which is explained in detail later, as well as compare it against the unmodified luggage (backpack) prediction from Faster R-CNN, the other main-stream object detector. Note that with limited compute power and time, we only selected the class "backpack" to detect and train from.

4 Experiment

4.1 Choosing and Filtering Datasets

First of all, since our model requires large amount of images with the appearances of backpacks, we chose our training set from established deep learning image platforms that support separation of data via categories. We have attempted to use data from between COCO and Google Open Image Dataset, which are discussed below.

COCO[4] After downloading the entire COCO dataset, we used COCO's build-in API to get all images, labels, and annotations with the "backpack" tag. However, after feeding this dataset into our detector's default settings and trained for a decent amount of time, our detector behaved very poorly, with low accuracy, low spacial precision (the bounding box is board and not in place), while classifying human as backpack. Therefore, we decided to stir clear of COCO dataset as baseline.

The undesired COCO performance has a few potential reasons, one of them being the vast diversity of COCO data. COCO contains very diverse data of vastly different backpack images, causing the training loss to significantly fluctuate. Given the limited resources and time, we only train for limited epochs. Thus, our snapshot could have stopped at a point of undesired fluctuation, resulting in a low test score. In conclusion, if deciding to proceed with COCO, a highly selective training set and increase training time will be needed.

Google Open Image Dataset[5] In order to use Google Open Image Dataset, we selectively downloaded images with "backpack" tag, and sanitized their corresponding labels and annotations to simple-class only (delete other classes' information). This dataset turns out to work just fine and thus chosen as our training and validation dataset.

4.2 Training and Improvement

4.2.1 Training

To train our network, the preprocessed dataset is split randomly into 90% as training data and 10% as validation data. Then we will apply the standard darknet training process to both datasets. We trained

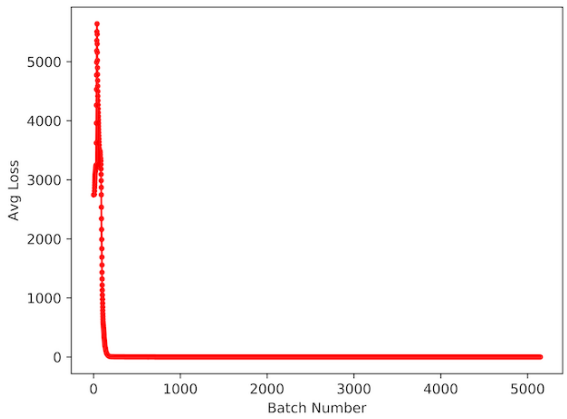

Figure 2: Similar detector's training loss plot from Dr. Nayak's post

our network for 1000 - 1500 batch iterations, for it is resource and time-efficient yet able to minimize the loss to almost minimum (as shown in Figure 2).

4.2.2 Improvement: Hyper-parameter Tuning

After the initial training with default setting from darknet, we used it as the baseline. We started fine-tuning a variety of hyper-parameters to suit it to the task at hand, increasing the performance of our detector. All hyper-parameter modification was done by updating the comprehensive darknet's configuration profile. The performance fluctuation affected by modifying hyper-parameters is discussed in the performance section.

CNN Structure darknet is a robust and complicated CNN-based structure. After our effort of understanding the purpose of each layer, we had multiple attempts to modify the structure itself, such as adding a layer, deleting a layer, and changing the channel size. However, every attempt has always broken or deactivate part of the normal functionality of darknet. Therefore, we disregarded the notion of structure modification, and instead focus on other hyper-parameter tuning.

Activation Functions The default darknet uses leaky ReLU as the generic activation function. Multiple attempts were made to use other activation functions such as sigmoid and tanh. Unfortunately, close observation of changes at different layers shows no obvious increment or decrement of performance. Moreover, the training time got extremely more slow by some of the activation functions, comparing to the fast leaky ReLU calculation. Therefore, we abandoned the modification of the default leaky ReLU activation function.

Batch This parameter indicates the size of batches used during the training process. The default batch size of darknet is 64. However, our single-class detector used a dataset much smaller than the entire original dataset used to train darknet. Therefore, the intuition is to reduce the batch size in according to emulate similar training behavior. After a couple set of training iterations, we determined the batch size to be 32 in our final version.

Decay Decay allows the network to penalize large weight parameters to prevent model overfitting. Even though almost all machine learning algorithms need to avoid overfitting, backpacks, unlike other classes of objects, varies relatively little. Therefore, the encouragement of tightly fitting the model over the backpack data in the training set actually brings more benefits than drawbacks. In the end, we decreased the default decay effect.

Angle Angle parameter allows data augmentation by duplicating the training data and give it a random rotation. The utilization of angle is critical in our case, due to the fact that plenty of real-life photos contain backpacks not being carried straight up, but being carried sideways on shoulders,

or horizontally placed on a flat surface. Thus, we assigned value 45 to angle so that we will get a random rotation by 45 degrees, so that our model can learn more about different possible rotations.

Learning Rate Learning Rate controls the rate of learning for each batch iteration. Since we decreased the batch size, the value of each iteration should also be decreased. Therefore, we adjusted the learning rate as well.

Burn In Burn In is for setting a period at the beginning of training in which the learning rate slowly increases to the maximum learning rate. This parameter prepares the network so that it learns faster later on. However, due to the fact that our simple-class detector is a relatively simpler task compared to comprehensive detection, it is reasonable to have a shorter warming up session. Thus, we decreased burn in rate from the default as well.

Confidence Threshold Confidence Threshold determines how confident we allow "correct detections" to be. This is typically used to filter out un-confident findings by the algorithm, so that a region will only indicate to contain backpack if the model is more confident. In our backpack detecting problem, similar to any other security problems, the system can tolerate false positive, while false negative are intolerable as they can cause mission failing danger to grow unnoticed. We should try our best to detect all backpacks, even if some of our detections are mistakes. Also, due to the complexity of our real-life based problem, lots of complicated situations make instances harder to detect and lead to lower confidence values. Therefore, we slightly lowered the confidence threshold.

5 Performance and Comparison

5.1 Testing Dataset

As a narrow focus of Computer Vision, detecting baggage in high security spaces requires performance evaluation specifically in settings of danger-prone or gated areas. In other words, the test photos need to look like they are taken from security cameras, instead of a close-up or a portrait; and the baggage in those photos tend to be occluded and low-quality. Since most of the organized and labeled image data online are general purpose and very diverse in its environments, our specific test requirements has forced us to filter out many unlikely security footage. In addition, because most of the mature open-source implementations of Yolo and Faster R-CNN do not have accuracy estimation (mAP) for a single class, we tentatively created our own accuracy measurement metrics.

Our current handpick data comes with 50 test images with backpacks and 50 without, and their respective count of backpacks from Google Images. We took the effort to meticulously find data from Google Images, outside of our training data from the Open Images dataset and Coco dataset. The process took about 4 man-hours.

5.2 Effects of Tuning

By reducing the batch size, an immediate observation is the decrease of the training time for each batch of proportion by around 2 - 4 times. This allows training speed of about 1500 batches per an hour. When combined with reducing the learning rate, we observed that the confidence of correct predictions generally increases, and the accuracy of bounding boxes increases as well.

Decrease in burn-in rate aims to make model learn faster overall. Even though it produced no direct observable results, we have observed and utilized it to expedite the training process.

Adjusting the decay and angle proved to have little impact over the accuracy. We observed such tuning enables some new detections to be made yet failed to detect some of the original positive results. There is not an obvious pattern or connection between the new ones and the failed ones; thus, we will require larger dataset to effectively find the variance introduced by decay and angle towards right detections.

However, the number of false positives increased due to the decay and angle. With two groups of controlled tunings, we found that decrease in decay could increase false positives; in the other independent tuning, activating the angle could increase false positives by a huge amount. For the decrease in decay, this is largely because of overfitting. For the effect of angle, randomly-generated

Table 1: Performance Results and Comparison

Category Name	Our Detector	Faster R-CNN
Total Correct Detection	24.49%	32.65%
All Instances Captured	10.2%	8.2%
False Positives	12.24%	6.12%

Figure 3: Example output from our detector.

rotated backpack images could have made the network to learn weird shapes, resulting in the model to incorrectly detect image region that looks like backpack in some angle.

Decrease in the Confidence Threshold enabled the detector to predict more regions with lower confidence to be backpacks. More predictions will necessarily yield both more false positives and true positives. Given more resources and time, we will be able to find the optimal Confidence Threshold beyond our default 0.5 threshold.

5.3 Final Overall Performance

After hyper-parameter tuning, we finalized the configuration and weights of our detector, and ran it, as well as Faster R-CNN, over our testing dataset. Below are the results as well as few selected detection example.

As we can see (from Table 1), despite the lower accuracy, our detector performs relatively well over our test set compared to Faster R-CNN, given the complicated nature of the problem at hand. Also, comparing to Faster R-CNN, our detector predicted completely correct more often (the metrics found all backpacks within an image with no false positive). However, we do have noticeably more false positives over Faster R-CNN, yet this is largely due to the tuning of decay and angle. More fine tuning and training should be able to get rid of the problem. Note that the state-of-the-art Faster R-CNN is more accurate than YOLOv3-based object detectors, yet it trades off such accuracy with time. However, the faster nature of our detector is still the more preferred traits within the industry.

6 Conclusion

Security requires constant monitoring which is critical and repetitive. This inherent requirement is currently solved with tedious security jobs, which introduces human bias (e.g. bribes) and errors (tiredness and lack of inattention), and urges a more automated system. The proper utilization of newly advanced object detection techniques will dramatically change the field of security in terms of efficiency and accuracy.

Our results yields a visible improvement in accuracy and on par speed comparing to the baseline. Though small and tentative, this incremental step shows the possibility to bring solution to a long-lasting problem.

Since the current effort focus on the object class “backpack”, future plan is to include all possible portable object containers such as “briefcase”, “suitcase”, “backpack”, “handbag” as the result improves. This way we can deal with all possible large malicious attempts by perpetrators.

Our handpick images are limited both in number and in measurements, as we only counted the number of backpacks with no coordinates or bounding boxes. Future plans will include adding more data in order to more accurately measure the performances of the controlled and tuned models.

In the future, we expect more from the model in order to be production ready. Our algorithm should be able to find a sweet spot between speed and accuracy with high recall, should be robust enough to deal with different types of b, and should be able to maintain properties above under vastly different conditions such as bad-illuminated, partially or and intentionally occluded, or blurred. Given the wide potential application and the current advances in technology, we this is as something worth working towards.

Acknowledgments

Special thanks to Dr. Sunita Nayak from Big Vision LLC. for his explanatory post over single-class object detectors (https://www.learnopencv.com/training-yolov3-deep-learning-based-custom-object-detector/) as well as the codebase provided, which became the basis of our project.

Special thanks to Google Images Search, for our custom testing dataset is collected from there. All images' copyright belongs to original owners/creators.

Special thanks to Rohith Gandhi, for his short, but thorough and clear explanation on the R-CNNs and Yolo (https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e).

References

[1] J. Yang, J. Lu, D. Batra, and D. Parikh. A Faster Pytorch Implementation of Faster R-CNN. 2017. *https://github.com/jwyang/faster-rcnn.pytorch*

[2] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. 2015.

[3] J. Redmon, and A. Farhadi. An Incremental Improvement, ArXiv eprints, 2018.

[4] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollar. Microsoft COCO: Common Objects in Context. ArXiv eprints, May 2014.

[5] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, T. Duerig, and V. Ferrari. The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale. arXiv preprint arXiv:1811.00982, 2018.