You Only Live to Look Once Weapons Object Detection

Sarah Alkahteeb and Camille Porter

***Abstract*—Object detection find multiple objects within an image. We used the YOLOv4 algorithm to implement real-time detections that might help solve some real-world challenges in the security field. Preparing suitable data for this task is essential since we want to reflect on real situations and train our algorithm on different scenarios. We used labeled images from OpenImages as well as images we downloaded and labeled ourselves. We achieved a training loss of $\approx$ 1.1 and validation mAP $\approx$ 83% on 5k images with two classes (guns and knives). Many features are shared between objects of interest (different weapons), making it hard for the algorithm to learn how to distinguish between these small aspects.**

I. Introduction

HUmans are able to easily recognize and locate objects of interest in an image at once. Performing this same skill with computers is more difficult. Computer vision is the field of teaching computers how to process images. Several techniques are available to be used on images in the computer vision field. These techniques help machines understand and identify objects in images. Image classification is one part of computer vision and is used to classify a whole image into one category. Object detection is similar to classification but can detect more than one object in a single image. It locates each object with the image, can detect multiple classes in one image, and can find the number of objects.

Over the years, computer vision techniques are used in many sectors, including healthcare, self-driving vehicles, manufacturing, etc. Here in this project, We are interested in security cameras and object detection in this area. Detecting weapons and perhaps achieving faster responses is useful. Video surveillance is a challenging task because it is difficult to detect weapons in a video. There can be low-resolution images, weapons can be held at different angles, the backgrounds will be different, and the weapons themselves vary quite a bit in size and shape. We are applying the You Only Look Once (YOLOv4) algorithm to weapons detection because YOLO approved to be the fastest so far and can be implemented in real-time detection. Automating weapons detection could be beneficial, especially if it were fast enough to send an alert in real-time. We are focusing on two categories of weapons: guns and knives. There are lots of images of guns and knives available online, and they are probably the most commonly used weapons, so they will be the most useful to detect.

II. Background Theory and Related Work

Generally, what object detection algorithms do is it looks at images and then they try to find different objects in the image. When these objects are found, the algorithm will put them in labeled bounding boxes. The algorithm finds a region of the image (contained within a box) that is likely to contain one of the classes that it is being trained for. Then features are taken from each bounding box and evaluated to see whether they contain the objects of interest. Finally, overlapping bounding boxes are deleted until there is a single most likely box. The number of boxes that are searched has a big effect on the accuracy and speed of the algorithm. If there are more boxes, it is more likely that the algorithm will find the object of interest, but it will also take longer.

Since deciding how many boxes to use is central to the speed and accuracy, it is one of the things that differs most among different algorithms. The first methods used a sliding window approach, where a bounding box was moved over the whole image, shifting a few pixels at a time. Boxes of different aspect ratios would be used as well, and it was slow, as well as less effective. The algorithm treated each box as a separate image and there was no information sharing while training. Moreover, the window would often not happen to be placed right on the center of the object, so some of the object would not be included in the box. Region proposal network was next used to decrease the number of boxes that were searched, so instead of sliding all over the image, the network selects the most useful 2000 boxes that might contain objects of interest. Different algorithms used different methods to find features in an image, such as finding key points and making a histogram of gradient for regions that surround the point. R-CNN is an example of where we use region proposal network. First, it finds the region of interest, next it puts each region in a CNN to extract features from these regions, and then uses a support vector machine to classify each region.

The algorithm that we're going to use is the You Only Look Once (YOLOv4) algorithm. It is faster than the older algorithms because it combines a lot of good ideas from the computer vision field. The most important innovation is that it uses a single convolutional neural network (CNN) for the whole image (hence the look once). One convnet is able to search for the positions of objects as well as the classifications of each group of pixels. The CNN extracts features, predicts bounding boxes, decides which of several closely related boxes is the most likely to contain the image (non-maximal suppression), and perform contextual reasoning all at the same time. [1] It then breaks up the result of the final layer of the CNN with the grid. The size of the grid can vary, with more grid cells leading to a larger number of potential object classifications. Image classification and localization is applied to each resulting grid cell. When the grid is larger, it is possible to find more objects. This is much faster than running a new CNN for each grid cell one at a time and is

why YOLO is more efficient (and can perform in real-time). YOLO outputs the bounding boxes coordinates explicitly, and this allows the neural network to output boxes of any aspect ratio and obtain more precise coordinates.

For each of these grid cells, we specify a label y (a dimensional vector) that contains: p_c to indicate whether there is an object or not in that grid cell, b_x, b_y are the coordinates of the center of the object, b_h, b_w and the height and width of the box, and c_1, c_2.... indicates which of the classes it is. If an object falls within the bounds of more than one grid cell, it is assigned to the cell which contains its center.

If each grid cell can only classify and locate one object and if there were more than one object that had its center in a grid cell, YOLO would be unable to detect both. Anchor boxes are the innovation around this problem. Each grid cell gets a certain number of anchor boxes, each with a different aspect ratio. When there are multiple objects, we define the label y = $[p_c b_x b_y b_h b_w c_1 c_2 p_c b_x b_y b_h b_w c_1 c_2]$ where each $[p_c b_x b_y b_h b_w c_1 c_2]$ denotes a predefined anchor box and y expands according to the number of predefined anchor boxes. Then the grid cell predicts $[p_c b_x b_y b_h b_w c_1 c_2]$ for objects that have a similar aspect ratio to that box. This enables YOLO to predict multiple objects per grid cell. If there are multiple objects close together with the same aspect ratio, YOLO won't detect them very well. Since we have different values inside y, different loss functions used to calculate the loss or evaluate the training. For p_c, since it tells us whether there is an object or not, it has 0 or 1 values and thus a proper loss function is a logistic regression loss. $b_x b_y b_h b_w$ the bounding box coordinates are a regression problem and we use a squared error loss. finally, $c_1 c_2$ classification problem which we probably will use a log likelihood loss to the softmax output or cross-entropy loss.

Since YOLO generates B bounding boxes for each grid, we get multiple predictions and we need to sort these and choose the best one. To get rid of any boxes that overlap, YOLO uses non-maximal suppression so that it only predicts each object once. First, it looks at all the boxes and removes those with a score lower than a specified threshold (like 0.5). Next, non-maximal suppression chooses the boxes with the largest probability (most confident detection), then looks at the remaining boxes with a high overlap with the most probable boxes and removes them, and so we will be left with boxes that have high scores. The overlap is measured by the intersection over union (IoU) score, which is the area of overlap of the two boxes divided by the area of the union of the two boxes. It doesn't discard boxes that don't overlap too much because then it would be unable to detect multiple objects.

For each grid cell, YOLO predicts B bounding boxes and each box has a confidence score. This confidence score is computed by using the box's confidence score multiplied by the conditional class probability. This confidence score measures the confidence of both the location and the class of the object. The box confidence score is the probability that the box contains an object multiplied by the IoU between the true box and the predicted box. The conditional class probability is the probability the object belongs to a class given that there is an object. Thus, the class confidence score is the probability that an object belongs to a class multiplied by IoU.

YOLO also outputs a confidence score that tells us how certain it is that the predicted box actually encloses some object. The score reflects how confident is that the box contains an object and how accurate it thinks the box is that it predicts (It measures the confidence on both the classification and where the object is located). A higher confidence score means it is more likely something is something significant. For each box, the cell also predicts a class. A confidence score for the box and class prediction are combined into one final score that tells us the probability that this box contains a specific type of object. Figure 1 shows the basic stages of YOLO.

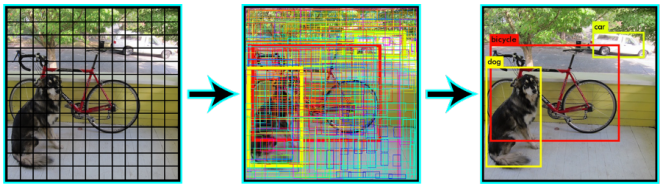

Fig. 1. From the original YOLO paper. [1] The YOLO algorithm breaks an image up with a grid, uses anchor boxes in each grid to make many bounding box proposals, and then uses non-maximal suppression to pick only one box for each object.

When the algorithm detects an object, usually the resulting box is not exactly the same as the human made box. Object detection is evaluated with the mean average precision (mAP) metric. mAP can tell us about the performance of the algorithm, and in object detection we calculate mAP using the precision (the percentage of correct predictions) and recall (what percentage of objects did we find). In object detection we make predictions in both bounding boxes and classes, and so we calculate the precision and recall using the IoU score for a given threshold. To make it simple, let us consider an example. If the IoU threshold is 0.5, then anything above that threshold will be a true positive and anything below will be a false positive. The AP (average precision) is the area under the precision-recall curve and mAP is the average precision. So we calculate the average AP over all classes [11].

The YOLO algorithm was written by Joseph Redmon and it is based on a framework called Darknet, a flexible framework written in low level languages. The original YOLO was the first algorithm that simultaneously predicted bounding boxes and class labels from one convolutional network. YOLOv2 added BatchNorm, higher image resolution, and anchor boxes. YOLOv3 added even more properties like scoring bounding box predictions and improved performance on smaller objects. [8]

The authors of YOLOv4 describe it has having a backbone, neck and head. First the backbone which is CSPDarknet53. The backbone is a pretrained model typically on Imagenet classification. Using a pretrained network means we have access to these network weights that are already trained to identify certain features. We can utilise these weights to implement a new object detection task. CSPDarknet53 is based on Densnet, so it has the ability to reuse features and reduce the number of parameters because Densnet was designed to connect layers in the CNN. After that we have the neck stage

where the features produced from the backbone are combined [8]. YOLOv4 uses Path Aggregation Network (PANet) for its feature aggregator in its neck. PANet is good because it can maintain spatial information exactly which helps in knowing the location of pixels. The image goes through multiple layers in the neural network, and in later layers, the features become more complex and hard to interpret and the spatial information lost accordingly. PANet uses a bottom-up path in addition to the top-down path that was used in YOLOv3. This method allows additional connection between the lower and upper layers and that helps in accessing features from all layers. [9] Finally, we have the head stage where the detection happens. The head stage predicts classes and the bounding boxes. In this stage a dense prediction is preformed that is predicting a label for each pixel in the image. [8]

Data Augmentation in YOLOv4: YOLOv4 comes with what the authors call a bag of freebies–things that improve the performance without causing any latency at inference time. Most of these are data augmentations methods that might help in improving the model performance such as Distortion, Image Occlusion, Cutout, as well as others. One data augmentation method used during training in YOLOv4 is CutMix, which cuts parts from images and then combines these parts together onto the augmented image. This helps the model to learn how to make predictions based on a larger amount of features. Another is mosaic data augmentation, which combines 4 training images into one image. This allows the model to identify objects at a smaller scale rather than a normal scale. One technique not related to image manipulation that YOLOv4 uses is class label smoothing. Instead of using 1 to describe which class is most likely, it uses 0.9. It comes with the assumption that when the model reaches a prediction = 1 is often wrong and indicates an overfitting situation, and so it would be more reasonable to encode values with some uncertainty. Besides data augmentation, YOLOv4 also uses Self-Adversarial Training (SAT). This is where the most useful part of the image is beclouded, which makes the network find new useful features. The image is edited after performing a normal training step [10].

Most algorithms are pre-trained on different sets of images by the authors. This allows for algorithm comparison, and is useful for transfer learning. Transfer learning involves pre-training an algorithm on a large dataset for a long time. Many of the features that are found are generic and can apply to new datasets. We take the weights from the first dataset and use new data and categories which needs to be similar to the pretrained one to make use of the features. This greatly increases the speed of training. We are using the YOLO algorithm that has been pretrained on the COCO images. The COCO dataset contains 200K images in 91 categories with natural backgrounds (not only white behind the object) and that aren't necessarily centered on the object. [3] These images teach the algorithm to perform better under more adverse circumstances which will be good for our use case. Security cameras will have blurry images, with objects partially hidden, and objects that are small and not centered.

III. Method/proposed solution

The problem we seek to solve is that the idea of certain attacks in small shops or maybe areas where it is hard to notice an attack fast. We are interested in experimenting if object detection can help in producing faster alerts and actions. So for example, if an attack happens in a supermarket, while the attacker insider there could be no one else beside the owner of the employee and if the security camera could capture an attack we could then start an alert using maybe sound to make an attention. Another use could be if security guards for some reason are not paying attention to certain area we could help in notifying them to take the proper action fast.

In order to perform object detection, we needed labeled data. If we do not have a large amount of data, the detection algorithms might not work well. We found 951 knife and 1103 gun images that were labeled from Open Images website. [5] In order to make our dataset more robust, we found our own images by searching on Google and annotated them. We used search engines to find images, looking up terms like "knife", "cutting", etc. We found some images that were made by people selling the object. They would be in front of a perfectly white background, with the whole object in the picture, and perfectly centered. That kind of object is easy to classify. We also found images that had a natural background, where the gun or knife was smaller or blurry. We found a variety of kinds of guns and knives–small and large–because we want our classifier to work for all types.

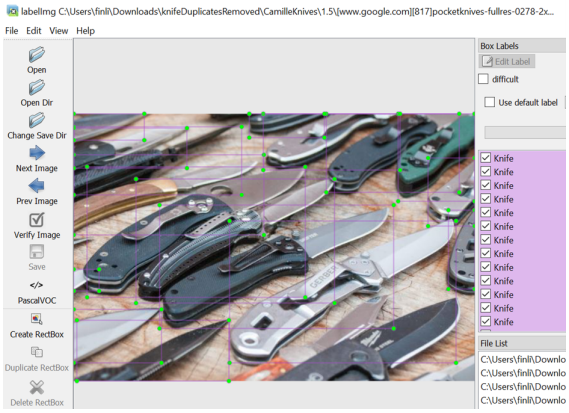

Fig. 2. Using the labelImg tool to label an image with lots of knives.

We had to spend quite a bit of time cleaning the data that we annotated ourselves. A lot of the pictures we downloaded weren't useful–they were blurry or not categorized correctly. After collecting first batch of images, we manually drew boxes around them using labelImg, as pictured in Figure 2. [4] It creates a text file for each image file with the object class and coordinates for each object. Labeling the pictures is time consuming. It took us about 1 hour to label 100 images. Because of lack of attention we accidentally saved some of our annotations as xml files, which YOLO didn't accept (YOLO works with txt files).

After annotating we faced a problem that some names are not acceptable by the YOLO and thus we had to go through all images manually and change anything suspicious.

Coco dataset includes kitchen knives and no guns and we tried running an image that included five knives before performing training on our dataset. Yolo couldn't find any of the knives which was interesting.

For our first attempt at training the YOLO algorithm, we started with the recommended number of images per class (about 2000), so we had 4000 training images overall which we ran for 8,000 epochs. This attempt had really poor performance, so we increased the number of images. The images that we added were in more realistic scenarios–in a natural setting, not centered in the frame, and some were grainy like security footage. We had 4959 training images for our second attempt. Many of them had multiple objects, so we had 3449 knives and 3552 guns labeled within the training set. We used a 85-15 breakdown for the test and training datasets. We trained using this data by following the original steps to code using the original GitHub site [6] by the authors. We also used a tutorial that had instructions for running YOLO on Google Colab. [7] Our parameters in the second training were: 12,000 iterations, batch size = 64, subdivision = 16, image width = 416, image height = 416, steps = 9600, 10800.

IV. Results/Discussion

As mentioned before, The algorithm was trained first using approximately 2000 images per class and after that we decided to increase the number of images to about 2500 per class and see whether adding more images would be useful. The first training results using the 2000 per class were a bit surprising. When we tested on new images, the algorithm captured only the gun class and couldn't identify any knives. The second training attempt (bottom Figure 4) took longer because we did more epochs (12000) and there were more images to analyze. We were not able to calculate the mAP during training during the second attempt because YOLO kept crashing. We calculated it afterwards. This time after evaluating the loss and mAP [12] we noticed that after 4000 iterations there was no improvement so we used this weight for our predictions. The predictions resulted from this round can be shown in Figure 3. The algorithm was able to detect both classes. It sometimes confuses the categories but the results are obviously better.

Fig. 3. Automated weapon detection using YOLOv4

The prediction task took approximately 10 milliseconds per image, which means that the algorithm can process 10,000 images in 2 minutes. The speed can be increased by using a more powerful GPU. We used the GPU available from Google Colab.

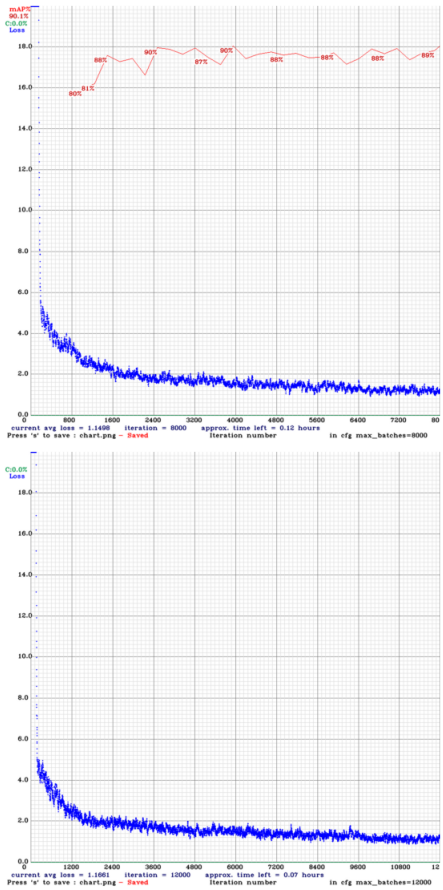

Fig. 4. Loss and mAP results. Top: First training. Bottom: Second training. We didn't calculate mAP during the second training because it caused the GPU to crash. The loss was similar at the end of both trainings, but the second training performed better.

A computer vision algorithm learns to map a set of inputs to a set of outputs from training data, and machines learn by means of a loss function. The loss function reduces all the good and bad aspects of a system down to a single number, which allows the computer to evaluate how well a specific algorithm performs. If predictions/outputs deviate too much from actual results, loss function would be a very large number, in other words, the smaller the loss function the better the performance (the predictions and real classes are pretty similar). [13]

As we can see from Figure 4, the loss function is approximately 1.1, and after 4000 iterations (the X-axis is the iteration number) in both first and second training, it seems that the loss and mAP are not improving. However, 1.1 is a low score indicates that the performance of the algorithm is good, and the differences between predictions and true boxes are very small. Moreover, the mAP ranges between 81-83%, which is a good score.

V. Conclusion

The dataset that we use seems to have a big effect on the results. Anyone who wants to make a weapon detector for the real world should make sure the training images match the

images they will use and that they have enough images to be accurate.

It seems that knife and gun have lots of similar features that made the algorithm bit confused and sometimes mixed these two classes. Since we are interested in general in finding anything harmful and it is common to know that knives and guns are the most popular weapons maybe we can combine these into one class called weapons and make the detection more robust. Moreover, some limitations we thought of is that sometimes people carry weapons around without attacking. One solution or idea is to make the detection more specific by including the landmark detection with the weapons detection. In landmark detection, we can have a neural network that output x,y coordinated of important points in a human body in an image. These landmarks can identify a pose of a body in which we think that in an attack situation, the attackers mostly have similar positions that indicate a dangerous situation, and then we can make the alert more accurate and specific.

Implementing segmentation to this problem could be useful since, in our results, we noticed that the algorithm detected some extra elements that are similar to weapons material and not much cared about the shape of the weapon. Segmentation is an object detection method that allows the algorithm to classify pixels instead of a whole object.

With a very different training dataset made of luggage scans, object detection would be quite useful for airport security. They are already using a camera and looking for weapons, so it is a good fit.

REFERENCES

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, You Only Look Once:Unified, Real-Time Object Detection. https://pjreddie.com/media/files/papers/yolo.pdf

[2] A. Bochkovskiy, C-Y Wang, and H-Y. M. Liao, YOLOv4: Optimal Speed and Accuracy of Object Detection. 2004.

[3] T.Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J.H. Hays, P. Perona, D. Ramanan, C.L. Zitnick, P. Dollár. Microsoft COCO: Common Objects in Context. CoRR, http://arxiv.org/abs/1405.0312, 2014.

[4] D. Tzutali, labelImg [source code]. Available from github https://github.com/tzutalin/labelImg, 2019.

[5] Google Research [source code]. The Open Images dataset [Image urls and labels]. Available from github: https://github.com/openimages/dataset, 2016.

[6] YOLO user manual [source code]. https://github.com/AlexeyAB/darknet

[7] YOLOv4 Training Tutorial. https://colab.research.google.com/drive/1_GdoqCJWXsChrOiY8sZMr_zbr_fH-0Fg?usp=sharing

[8] J. Solawetz, Breaking Down YOLOv4. Roboflow. June 2020.

[9] Miracle-R, PANet: Path Aggregation Network In YOLOv4. Medium. July 2020.

[10] J. Solawetz, Data Augmentation in YOLOv4. Towards Data Science

[11] J. Hui, mAP (mean Average Precision) for Object Detection. Medium. May 2020.

[12] S. Yohanandan, mAP (mean Average Precision) for Object Detection. Towards Data Science. June 2020.

[13] R. Parmar, Common Loss functions in machine learning. Towards Data Science. September 2018.