Machine Learning report - Project II

Bartlomiej Binda, Clement Petit, Yanis Berkani
Department of Computer Science, EPFL, Lausanne, Switzerland
dr inż. Piotr Garbat
Institute of Microelectronics and Optoelectronics, WUT, Warsaw, Poland

***Abstract*—This report focuses on multi-object detection in image sequences. Throughout the paper, several detection algorithms (namely YOLO and SSD) and tracking algorithms (namely MOSSE, KCF and Deep SORT) are compared. Furthermore, the limitations of the algorithms are discussed, along with measures of accuracy and speed. We finally concluded that the best compromise was a model using SSD detection and MOSSE tracking : we managed to achieve 31 fps and 63% of accuracy.**

I. Introduction

The main purpose of this project is to detect many objects on a video. In particular, we will focus on multiple people detection since pedestrians are by far the most studied "objects" in the field of multi-object detection. In the current context of the Covid-19 sanitary crisis, such work could ,for instance, be used to differentiate areas that are too crowded (unsafe) and areas which are under crowded (safe), in places such as public transports, workplaces, and restaurants. Obviously, it could also be used for bad purposes such as mass surveillance, but this is by no means the purpose of this research.

Multi-object tracking is a very challenging task which consists in combining detection and tracking algorithms. First, the purpose of detection is to identify all the objects of interest on the frame and outputting the coordinates of their bounding boxes, which is a costly process as the whole frame needs to be scanned. Afterwards, tracking is used to obtain the respective trajectory of each known object in a sequence of frames, which is much cheaper than proper detection. Nonetheless, this process is especially challenging at a reasonable accuracy and speed, and the performance of the model generally depends on the quality of the detection.

This project aims to train our own detection algorithm, and combine it with a tracking algorithm, in order to obtain a model that achieves a minimum average speed of 20 frames per second on a dataset provided by the MOTChallenge, with moreover a "reasonable" accuracy. The latter means that the model should detect more than half of the people and that visually, each displayed box should indeed surround a person. Hence, the "actual tracking" of people, or in other words the information about "who is who" and "who went where", is not the purpose of this research. Tracking algorithms will still be used, but mostly as a faster alternative to performing detection at every frame (thus greatly improving the speed of the models).

In the following section, we will tackle the different detection and tracking methods that we considered to build different models. Then we will evaluate them to underline the pros and cons of each of them, before analyzing the effects of some particular parameters on a given model. Finally, we will summarize and discuss our results.

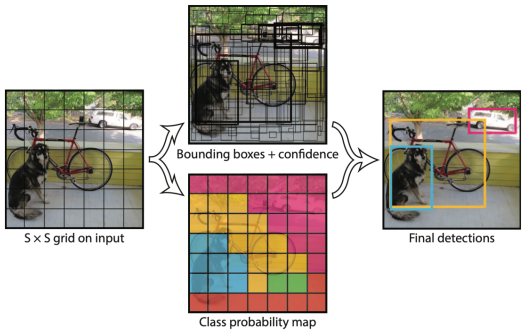

Fig. 1. Visualized idea of YOLO [2]

II. Research, Models and Methods

There exist several algorithms for detecting or tracking objects. Hereby, we opted for some of the most well-known ones and below we present our models and crucial findings about particular methods. Unless stated otherwise, the code for this project was written by us.

A. First glimpse of the problem with YOLO detection

To get more familiar with the subject and obtain some first results, we started with a simple implementation, relying on already trained YOLOv3 (YOLOv3-spp and YOLOv3-tiny). We downloaded some .cfg files from pjreddie [1]. As the name hints, You Only Look Once, the algorithm takes a look at the photo just once and finds all the objects. In particular, YOLO uses a convolutional neural network which divides the image into regions and predicts bounding boxes and probabilities for each region. It can be used in real-time while still achieving great results and thus we decided to start with exploiting this algorithm.

1) **YOLOv3**

In order to utilize the pre-trained neural network, we used the OpenCV library, which reads the deep learning network using some given weights (.weights file) and a configuration (.cfg file). Additional required input was of course the sequence of images in which we wanted to detect the objects. From each provided image we created a blob (Binary Large Object) and set this as

our neural network's input. This neural network is able to recognize 80 different classes of objects, one of which is the class "person". The network first divides the images into multiple sectors and for each tries to find bounding boxes with corresponding probabilities of being an object from each of the classes. This is presented by Figure 1. For every obtained output we find the class id corresponding to highest confidence score (metric of certainty that the prediction is correct), then check if this value is above some threshold - in our case 0.4. If the condition holds we extract the box position from the output and store it with the confidence score and class id. The last step is to simply draw the box on the image and assign to it the correct label using coco names list.

Unfortunately, when we tried to evaluate plenty of images at the same time and see the speed efficiency of our algorithm it performed very poorly. (no more than 2-3 frames per second on a MacBook Pro 2017, with dual-core Intel i5 CPU). This led to the idea of adding a tracking algorithm to the program.

2) **OpenCV Trackers**

As detection alone is too slow, we decided to perform detection once every n frames (we started with n=10) and tracking the rest of the time. Tracking can be done quicker than detection because it exploits the location of the object in the previous frame [3]. At this stage, we decided to implement and compare two tracking algorithms, namely KCF and MOSSE.

The Minimum Output Sum of Squared Error (MOSSE) [4] and Kernelized Correlation Filters (KCF) [5] algorithm are using correlation filters. By exploiting them the algorithms achieve robustness to graphical transformations like rotation or scaling [6]. While KCF is more precise, MOSSE is much faster, which is crucial for our project. The major issue of both algorithms is that if an object becomes hidden by something else, the algorithm will lose the position and mark wrong area [7].

Figure 1:.Erroneous KCF tracking results

Fig. 2. Losing track of object by KCF [7]

3) **YOLOv3 limitations**

While working with the algorithm, we also noticed some flaws that YOLOv3 is prone to. First of all, the algorithm forces heavy spatial constraints on bounding boxes [2]. Thus, it leads to significant problems when detecting many nearby objects. This is especially true for the light YOLOv3 models, which struggles even with little gatherings of people.

Another imposed limitation was the previously mentioned speed. Even with the tracking algorithms, we were still not satisfied with the frames per second that we obtained, by running the code on our hardware. Consequently, we tried to upgrade our model to YOLOv5 to see the difference in terms of speed and accuracy. We used an existing implementation and added tracking. Unfortunately, the newer algorithm did not give us any major improvement in terms of speed efficiency (only 2 more fps achieved). We thus believe that our hardware is partly responsible for the slowness. Furthermore, classifying 80 classes instead of just one is much more computationally expensive, and requires a much deeper and bigger network, which also impacts the speed efficiency (this was especially the case with the YOLOv3-spp model).

Due to the above-mentioned speed problem, partially due to our weak hardware, we transitioned our work to Google Colab for the next research part (implementing SSD detection), in order to utilize their accelerated computing environment.

B. Training an SSD detection with Tensorflow

Single Shot multibox Detector (SSD) [8] is an efficient method for object detection. The method consists of first running a deep neural network (only once) on an input image in order to extract a feature map. Then, small convolution filters are applied to this feature map in order to compute bounding boxes. As this method had previously proven itself in terms of efficiency and speed, we then decided to try and train our own single-class SSD model. For this purpose, we used the Tensorflow object detection API.

1) **Tensorflow object detection API**

First of all, Tensorflow is an open-source library for Machine learning and deep learning, which includes APIs with models and algorithm. The Tensorflow Object Detection API specifically helps in the creation of deep learning networks for object detection purposes. Training of models can be done from different provided pre-trained models, that are available in the Model Zoo. Furthermore, for the training algorithm, we used some code provided on the official tensorflow repo [9].

2) **Selection of data in MOT Dataset**

As mentioned earlier, we used the dataset provided by MOTChallenge. The format of this data consists of two categories, train and test, which group numbers of video folders. These video folders contain all frames for each video, in .jpg format, along with det.txt files which contain bounding box coordinates for every detection of every frame. We selected train and test videos, for which we kept only a subset of the frames so to obtain a (test detection) / (train detection) ratio of around 0.25. As we were working with Tensorflow for training our SSD model, we needed to convert our data into TFRecords, which are Tensorflow's own binary storage format. We wrote a script for this purpose, which combines the images, along with their corresponding detection coor-

dinates, all into TFRecords (two separate TFRecords for train and test data).

As we only need one class in our model, we created a label map with a single entry ”person”, and the corresponding id ”1”.

3) **Training of the model**

To initialize our model, we chose a pre-trained model from the TensorFlow 2 Detection Model Zoo. We went with the SSD MobileNet V2 FPNLite 320x320 model, that is mostly intended for mobile applications, as it is designed to provide a very good speed with still a reasonable accuracy.

The TensorFlow Object Detection API allows to specify some configuration for our model, via a pipeline.config file that comes with the pre-trained model. We then needed to adapt this configuration for our model, through different values, such as the function of our model (detection), the number of classes (1 single class), the paths to our label map and TFRecords, the path to the starting checkpoint and the batch size for the training.

The training checkpoints contain the values of all parameters used by a model. In our case, we started our training from the pre-trained model's checkpoint-0. As the training is performed, some new checkpoints are created, and you can either use your last checkpoint's values to start performing actual predictions or also continue the training of your model, from a specific checkpoint.

Regarding the batch size, putting it to a higher value will allow your model to converge in a smaller number of training steps, provided that your GPU has enough memory. In our case, Google Colab provides Nvidia Tesla T4 GPUs, with 16GB of memory.

We used the training algorithm provided by the Object Detection API. With 5000 training steps, a batch size of 8, and the data we provided, the training took around 30 minutes to complete, and we obtained a total loss of 0.379.

4) **Test of our model with MOSSE tracking**

We tested our detection model with the same ”speed-efficient” approach as we did with Yolo. Thus, we perform a detection every 10 frames (this number is by default and can be changed when calling the demo script) and for the other frames, we rely on OpenCV's MOSSE tracking to keep track of the different detected persons. For a given frame, the detection process gives, for each detected object, the coordinates of the bounding box, and a score. Beforehand, a minimal score is set, and we display on the image only the bounding boxes with a corresponding score superior to this minimum. We found out that despite low scores, the detection is quite accurate for the large majority, so we set the minimum score as low as 0.4 (this minimum is by default, and can be changed when running the demo).

5) **Test of our model with Deep SORT tracking**

The previous approach, although efficient in terms of speed, does not allow to keep track of detection ids (who went where). Therefore we also tried another approach where we performed detection at each frame and used tracking to match each detection to the similar detection at the previous frame. For this approach, we used Deep SORT tracking. Deep SORT relies on Kalman filters, which make predictions on the position of objects, based on their current velocity. Thus we were able to keep track of our detection through entire videos (this is proper tracking), but the fact of performing detection at each frame has of course a large impact on speed. We used some of the code from official implementation of Deep SORT [10][11], along with a slight modification from pythonlessons that enables in particular compatibility with Tensorflow v2, and the use of their trained Deep SORT model.

Fig. 3. Comparison of the algorithms. SSD MOSSE : the fastest; YOLOv3-spp : the most accurate; SSD Deep SORT : the only one to ”track” people.

III. Results

Let us now compare the different models before analyzing the influence of the minimum score threshold parameter on one of them: SSD MOSSE. The following results were obtained this time on a MacBook Pro 2019, with hexa-core Intel i7 CPU.

A. Different algorithms

Figure 3. clearly depicts that there is a speed/accuracy trade-off and that the different models vary a lot in those terms. On one hand, we have YOLOv3-tiny (with KCF) and SSD (with MOSSE), that are fast but not highly accurate. On the other, hand we have YOLOv3-spp (with KCF) and SSD (with Deep SORT), that are more accurate but much slower.

In particular, we obtained that YOLOv3-tiny with KCF gave a speed of 18.9 FPS on average but that it could clearly not spot enough people, especially under tough conditions, like for example a crowded or dark picture. Then, SSD with MOSSE appears as the fastest of the models with a speed of 72 FPS on average, while still achieving a good accuracy with close to 40% of people detected, but once again struggling a bit under

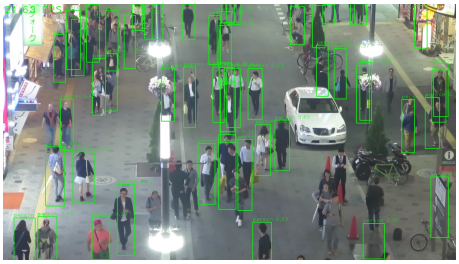

Fig. 4. SSD Mosse with threshold=0.25

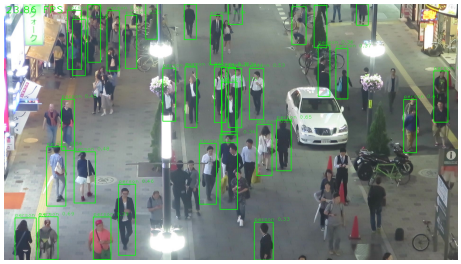

Fig. 5. SSD Mosse with threshold=0.325

Fig. 6. SSD Mosse with threshold=0.40

bad conditions (for example, it sometimes detects two close persons and consider them as a single person). Concerning YOLOv3 using the YOLOv3-spp weights with KCF tracking, we achieved the best detection accuracy by spotting almost 80% of the people on the picture and even being able to detect other objects such as cars, bicycles and motorcycles. It is also less prone to errors under bad conditions. However, it comes with a cost of running at only 2.9 FPS despite the fact that we use detection once every 10 frames (and thus are not able to "track" people). Finally, SSD with Deep SORT is also very slow (4.1 FPS on average), and quite accurate (50% of detected people), but especially, it is the only one that uses detection at every frame and that is thus able to make use of the tracking algorithm to actually identify people all along the video.

To conclude, all those methods have their own advantages and inconveniences and they all underline the fact that there is an accuracy/speed trade-off. One should choose between those methods based on their needs and preferences. In our case, the one that meets the most our needs seems to be our model of SSD with MOSSE, as it is very fast and has a reasonable accuracy. Thus, we will now see the impact of the confidence score threshold on this particular model (III-B).

B. Different parameters

As mentioned, we will focus on the SSD MOSSE model to see the influence of the confidence score threshold parameter on the speed and accuracy of a model. We will also analyze the impact of the "frequency of detection" parameter.

Overall, the results are very intuitive. Indeed, as we can see on the figures 4, 5 and 6, the higher is the threshold for the probability of detecting a human being, the quicker is the model (but hence it detects less people so the accuracy is lower). In particular, we observe that with a very small threshold (0.25) we have the most accurate results in the sense that we detect in average more than 81% of the people, but at the cost of being the slowest (19.9 FPS in average). It is also important to notice that with such a small threshold, there is a high rate of false positive (detection of a person at a random place where there is no human being) : 9% of the detection are false positives. Then, with a medium threshold of 0.325, the accuracy (amount of detected people) goes down to 63% but the false positive rate decreases as well, to 2.5% and in parallel the speed increases to 31 FPS. Finally with a threshold of 0.4, the accuracy goes down to 39%, with a false positive rate of less than 1%, while the speed goes up to 72 FPS.

When it comes to the frequency of detection, once again we find coherent results: the less frequent is the detection, the faster is the model, but also the less accurate it is. For example, with one detection every 5 frames instead of 10, we found that the boxes were more precise with a slight increase of accuracy (42% instead of 39%) but that the speed of the model decreased (61 FPS instead of 72 FPS).

To conclude, we found that the parameters that best fits our requirements are a confidence score threshold of approximately 0.325 and using detection once every 10 frames. Using those parameters on our model that uses SSD and MOSSE we have an accuracy of 63% and a speed of 31 FPS in average.

IV. Summary and Discussion

In this report, we provided our experiences and findings about different detection and tracking algorithms. We can conclude that detection, for most of the algorithms, takes indeed much more time than tracking. The model that we found to be the most accurate is YOLOv3-spp with KCF, but because it did not satisfy the project's speed requirements we trained ourselves a much faster model using SSD and MOSSE. All the research that we have conducted was done for the Institute of Microelectronics and Optoelectronics, Faculty of Electronics and Information Technology at the Warsaw University of Technology. Based on our work, the Laboratory plans to continue the project during the next two years, expand to more objects and enable receiving inputs from multiple cameras at the same time.

References

[1] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016.

[3] H. Luo, W. Xie, X. Wang, and W. Zeng, "Detect or track: Towards cost-effective video object detection/tracking," 2018.

[4] D. Bolme, J. Beveridge, B. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters," *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 2544–2550, 2010.

[5] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 3, pp. 583–596, 2015.

[6] T.-W. Mi and M.-T. Yang, "Comparison of tracking techniques on 360-degree videos," *Applied Sciences*, vol. 9, no. 16, p. 3336, Aug 2019. [Online]. Available: http://dx.doi.org/10.3390/app9163336

[7] J. wan Park, S. Kim, Y. Lee, and I. Joe, "Improvement of the kcf tracking algorithm through object detection," *International Journal of Engineering Technology*, vol. 7, no. 4.4, pp. 11–12, 2018.

[8] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," *Lecture Notes in Computer Science*, p. 21–37, 2016. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46448-0_2

[9] C. Chen, X. Du, L. Hou, J. Kim, P. Jin, J. Li, Y. Li, A. Rashwan, and H. Yu, "Tensorflow official model garden," 2020. [Online]. Available: https://github.com/tensorflow/models

[10] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2017, pp. 3645–3649.

[11] N. Wojke and A. Bewley, "Deep cosine metric learning for person re-identification," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018, pp. 748–756.