Optical Flow Enhanced Multiple Objects Tracking

Guo Jixin
Huang Shan
Qian Kui

1 Introduction

1.1 Abstract

Multiple objects tracking refers to the process of detecting and identifying multiple moving or stationary objects in a video, which can be used in various industries like surveillance, medical imaging and augmented reality. The objective of multiple objects tracking is to associate target objects in consecutive video frames. However, this process faces one major challenge – Since it is a detection based identification process, when the detection of objects fails, the tracking of this object will also fail.
Optical flow is the pattern of apparent motion of objects, surfaces and edges in a visual scene caused by the relative motion between observer and a scene. It can be utilized to predict the position of certain object using the information in the previous frames of a video, which has the potential to solve the problem of multiple objects tracking. In this project, we implement an optical flow enhanced multiple objects tracking algorithm focused on human tracking. And we test the reliability and robustness of our tracking algorithm on multiple human intensive video clips, and analyse the results.

2 Description of Method

2.1 Algorithm

This project is about multi-object-tracking for persons. The algorithms involved consist of object detection with YOLO v3[1], Simple Online and Realtime Tracking (SORT) with a deep association metric[3] and a optical-flow feature tracker based on Kanade–Lucas–Tomasi (KLT)[2].

The flow of the algorithm goes as the followings. We process the frames of videos one by one. YOLO v3 is used to detect all persons within this frame with indexes, bounding boxes and scores. The detection results will be fed into deep SORT and the tracker will be updated. Given previous frame, current frame and current detection result, the Kanade–Lucas–Tomasi based optical flow algorithm will predict the possible positions of all persons. Then the predicted positions will be fed into deep SORT and the tracker will be updated again. So, in all, the tracker will be updated 2xN number of times, which N is the number of frames of the video.

2.2 Architecture

YOLO v3 network and Deep SORT CNN network

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 1: Darknet-53

Name	Patch Size/Stride	Output Size
Conv 1	$3 \times 3/1$	$32 \times 128 \times 64$
Conv 2	$3 \times 3/1$	$32 \times 128 \times 64$
Max Pool 3	$3 \times 3/2$	$32 \times 64 \times 32$
Residual 4	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 5	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 6	$3 \times 3/2$	$64 \times 32 \times 16$
Residual 7	$3 \times 3/1$	$64 \times 32 \times 16$
Residual 8	$3 \times 3/2$	$128 \times 16 \times 8$
Residual 9	$3 \times 3/1$	$128 \times 16 \times 8$
Dense 10		128
Batch and ℓ_2 normalization		128

Figure 2: deep sort CNN architecture

2.3 Equations

YOLO

Bounding boxes prediction. The network predicts 4 coordinates t_x, t_y, t_w, t_h. If the cell is offset from the top left corner of the image by (c_x, c_y) and the bounding box prior has width and height p_w, p_h, then we get the following bounding boxes prediction.

$$b_x = \sigma(t_x) + c_x \tag{1}$$
$$b_y = \sigma(t_y) + c_y \tag{2}$$
$$b_w = p_w e^{t_w} \tag{3}$$
$$b_h = p_h e^{t_h} \tag{4}$$

Deep SORT

Assignment Problem. Motion information has been captured using Mahalanobis distance between predicted Kalman states and new observations.

$$d^{(1)}(i,j) = (d_j - y_i)^{\mathrm{T}} S_i^{-1} (d_j - y_i) \tag{5}$$

Second metric is the smallest cosine distance between i-th and j-th detection in appearance space.

$$d^{(2)}(i,j) = \min\left\{1 - r_j^{\mathrm{T}} r_k^{(i)} | r_k^{(i)} \in \mathcal{R}_i\right\} \tag{6}$$

Build the association problem by combining these two metrics.

$$c_{i,j} = \lambda d^{(1)}(i,j) + (1-\lambda) d^{(2)}(i,j) \tag{7}$$

Optical Flow

Dominant feature detection, Harris feature points. Compare the error of shifting windows and find the largest and smallest values by finding the eigenvectors.

$$M = \sum_{(x,y)\in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \sum_{(x,y)\in W} I_x^2 & \sum_{(x,y)\in W} I_x I_y \\ \sum_{(x,y)\in W} I_x I_y & \sum_{(x,y)\in W} I_y^2 \end{bmatrix} \tag{8}$$

Kanade–Lucas–Tomasi tracker. Find residual pixel displacement vector $\mathbf{d}^{\mathrm{L}} = \left[\mathrm{d}_{\mathrm{x}}^{\mathrm{L}} \mathrm{d}_{\mathrm{y}}^{\mathrm{L}}\right]^{\mathrm{T}}$ that minimizes the matching error function between previous frame and new image frame.

$$\epsilon^L(\mathbf{d}^L) = \epsilon^L(d_x^L, d_y^L) = \sum_{x=u_x^L-\omega_x}^{u_x^L+\omega_x} \sum_{y=u_y}^{u_y^L+\omega_y} \left(I^L(x,y) - J^L\left(x + g_x^L + d_x^L, y + g_y^L + d_y^L\right)\right)^2 \tag{9}$$

3 Implementation Details

We have trained a YOLO model on a new dataset and details will be covered at section 4. This project is the environment of python 3.5, ubuntu 16.04. The dependencies of this project are tensorflow, keras, numpy, sklearn, scipy, scikit-image and opencv.

First, every frame of the video is captured. Yolo is used to detect all persons within every frame, which gives indexes, scores and bounding boxes. We use non-maxima suppression (NMS) to get dominant bounding boxes. These detections are used to update the SORT object tracker with a Deep Association Metric. During update, we will perform a matching cascade and the nearest feature association is based on a pre-trained CNN network as described above.

Listing 1 Matching Cascade

Input: Track indices $\mathcal{T} = \{1, \ldots, N\}$, Detection indices $\mathcal{D} = \{1, \ldots, M\}$, Maximum age $A_{\max}$

Compute cost matrix $\boldsymbol{C} = [c_{i,j}]$ using Eq. 5
Compute gate matrix $\boldsymbol{B} = [b_{i,j}]$ using Eq. 6
Initialize set of matches $\mathcal{M} \leftarrow \emptyset$
Initialize set of unmatched detections $\mathcal{U} \leftarrow \mathcal{D}$
for $n \in \{1, \ldots, A_{\max}\}$ **do**
  Select tracks by age $\mathcal{T}_n \leftarrow \{i \in \mathcal{T} \mid a_i = n\}$
  $[x_{i,j}] \leftarrow \text{min\_cost\_matching}(\boldsymbol{C}, \mathcal{T}_n, \mathcal{U})$
  $\mathcal{M} \leftarrow \mathcal{M} \cup \{(i,j) \mid b_{i,j} \cdot x_{i,j} > 0\}$
  $\mathcal{U} \leftarrow \mathcal{U} \setminus \{j \mid \sum_i b_{i,j} \cdot x_{i,j} > 0\}$
end for
return $\mathcal{M}$,$\mathcal{U}$

Figure 3: Matching Cascade

Given previous frame, current frame and current detection result, the Kanade–Lucas–Tomasi based optical flow algorithm will predict the positions of all persons. Then the predicted positions will be fed into deep SORT and the tracker will be updated again. So, in all, the tracker will be updated 2xN number of times, which N is the number of frames of the video. During the optical flow, we use Harris and Shi-Tomasi feature points within every bounding boxes. By tracking these feature points, we estimate a translation of objects between previous and current frame.

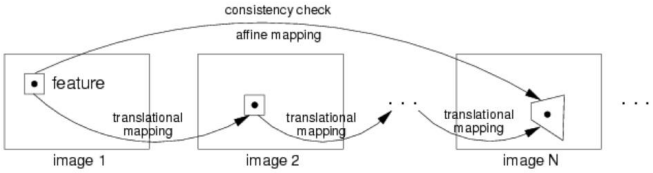


Figure 4: Kanade–Lucas–Tomasi algorithm

For illustration, we visualize detection and tracking bounding boxes. The detection and tracking results are written into txt files and a output video will be generated for visualization purpose.

4 Experimental Setting

4.1 Dataset

The origin dataset in our proposal, TrackingNet, whose size is more than 1TB is too large. We then turn to Pascal Visual Object Classes (VOC) data sets. Our own training work was based on VOC2007. There are twenty object classes selected in this set:

- Person: person
- Animal: bird, cat cow, dog, horse, sheep
- Vehicle: aeroplane, bicycle, boat, bus, car, motorbike, train

• Indoor: bottle, chair, dining table, potted plant, sofa, tvmonitor.

The training data provided consist of a set of images; each image has an annotation file giving a bounding box and object class label for each object in one of the twenty classes present in the image. Note that multiple objects from multiple classes may be present in the same image. The data has been split into 50% for training/validation and 50% for testing. The distributions of images and objects by class are approximately equal across the training/validation and test sets. In total there are 9,963 images, containing 24,640 annotated objects.

4.2 Training Parameters

Load pretrained weights from YOLO website and freeze layers in the first stage of training to get a stable loss. Freeze the first 249 layers of total 252 layers. Then unfreeze and continue training, to fine-tune. Parameters are shown in the Table 1.

Table 1: Training Parameters

Parameters	First Stage	Second Stage
Batch Size	32	8
Epochs	40	40
Early Stop	No	Yes
Learning Rate	0.001	0.0001
	Constant	Reduce (factor=0.1, patience=3)

4.3 Hardware Used

The training work was carried out on a Paperspace machine with Quadro M4000, which has 8 GB memory. The code has been tested in python 3.6, ubuntu 16.04. The test environment is python 3.6.7, keras-gpu 2.2.4 and tensorflow-gpu 1.12.0.

5 Results

In all the figures shown in this section, the blue bounding bounding box is the detection result while the white bounding box is the tracking result. The unique number on the top left corner of the white bounding box is the object ID. For the complete tracking videos, please refer to these links:
Video 1 Video 2

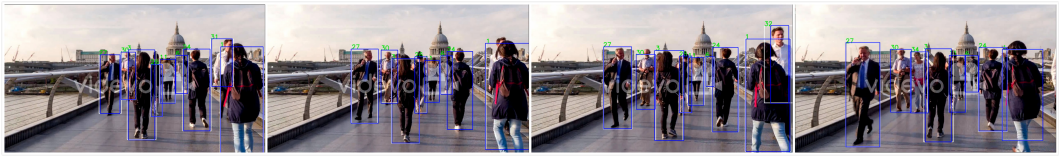


Figure 5: General Tracking Results

The above figure shows the general tracking results extracted from consecutive frames from a video clip. It clearly shows that how each object is detected and identified, and how the algorithm reacts when a new object shows up in the frame.

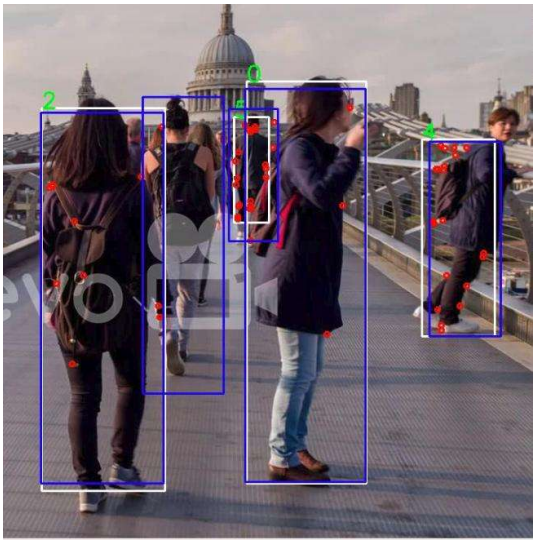


Figure 6: Feature Points

The above figure shows how optical flow extract feature points to perform further prediction.

Figure 7: with optical flow(Left),without optical flow(Right)

Figure 8: with optical flow(Left),without optical flow(Right)

The above two figures shows four frames that the detection fails since there is no blue bounding boxes. The two frames on the left are generated with algorithms equipped with optical flow, the white bounding box on the pedestrian shows that the tracking persist even if detection fails. However, the right two figures are generated without optical flow enhancement, where tracking also fails.

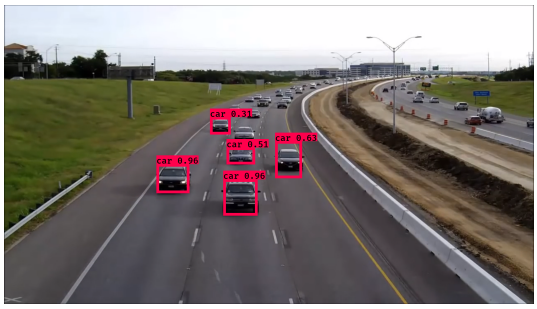
Figure 9: Wrong Tracking

In left graph, notice the man in black on the right of the figure who has been identified as object 5, in the next few frames he is occluded by the man in white, then in the right figure when the man in black shows up again, he is wrongly identified as object 41. This indicates that when the object has been occluded for too many frames and has too much posture variation, even with optical flow prediction, the algorithm still has high probability of failure.

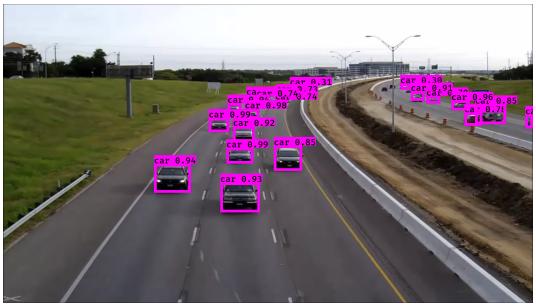
6 Discussion

6.1 Training results

We completed the training work of YOLO v3 on VOC2007 and assessed the performance of the trained weights on daily photos. The training process was stopped due to triggering conditions of early stopping. However, the loss was always up and down around 20, and our trained weights did not perform as well as the pretrained weights based on COCO data set. As shown in Figure 10, our trained YOLO v3 model recognizes many fewer objects and has less confidence on its judgment. It seems that the properties of data sets may have a decisive impact on the results of models.



(a) VOC



(b) COCO

Figure 10: Comparison of weights based on different data sets

6.2 Conclusion

Combining the methods of yolov3, deepsort and KLT based optical flow, this project realize a satisfaction level of multiple object tracking for human class. Traditional detection based tracking algorithm suffers from the problem that the failure of detection will result in the failure of tracking, with the help of optical flow algorithm, it is possible to predict the object next position using the information from previous and current frames, therefore, we have extra measurement of object information which can help to ease the pain of detection failure. Experiment shows that this method provides robust tracking result of multiple objects.

6.3 Challenge

Multiple object tracking, involving classification, localization and even prediction, is really a hard work. Thus, training a neural network for multiple object tracking takes much more time, even running with GPU. Although the results of our training work are not satisfying for complex scenes, our trained model can work well for VOC challenges and scenes with few objects.

Meanwhile, for object tracking, some classical methods are still competitive. With the pretrained weights of neural networks, these methods can solve problems that once they have no ways to deal with. We choose to combine these methods with deep learning and achieve better results.

6.4 Future Work

However, it is not negligible that when an object has been occluded for multiple frames and shows up again, it is of high probability to be misidentified due to the change in object's posture, this suggests that the information provided by deepsort and optical flow is not capable of handling long time occlusion, which is the main challenge in the future work of multiple objects tracking.

7 Apendix

Git Link

Multi-Object-Tracking

General Idea
A multi-object-tracking algorithm uses YOLO v3, deep-sort and optical flow based on Kanade–Lucas–Tomasi (KLT).

Methodology
1. YOLO v3 detection
2. deep-sort tracker update
3. optical flow tracker update

Dependences
The code has been tested in python 3.5, ubuntu 16.04.
1. tensorflow
2. keras
3. numpy
4. sklearn
5. scipy
6. scikit-image
7. opencv

How to run
1. Download yolov3 model from [YOLO website](http://pjreddie.com/darknet/yolo/). Convert this model to a Keras model. For this project, we train a new yolov3 model and use Keras.save-model.
2. Run script: python3.5 tracking.py

Results

1. test result video 1: https://youtu.be/SKX-EcQnens
2. test result video 2: https://youtu.be/56RKbOaInYI

Git Link for Training

Yolo3-voc-train

Dataset
VOC2007

Dependences
The code has been tested in python 3.6, ubuntu 16.04.
1. keras-gpu
2. tensorflow-gpu
3. matplotlib
4. Pillow
5. opencv

Reference work
1. keras YOLO v3: https://github.com/qqwweee/keras-yolo3
2. deep-sort: https://github.com/nwojke/deep-sort
3. YOLO v3 deep-sort integration: https://github.com/Qidian213/deep-sort-yolov3
4. optical flow: https://github.com/ZheyuanXie/OpticalFlow

References

[1] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". In: *arXiv* (2018).

[2] Carlo Tomasi and Takeo Kanade. *Detection and Tracking of Point Features*. Tech. rep. International Journal of Computer Vision, 1991.

[3] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. "Simple online and realtime tracking with a deep association metric". In: *2017 IEEE International Conference on Image Processing (ICIP)* (2017), pp. 3645–3649.