Augmented Reality labels for mobile devices

ELG5900 Project Report

Name: Lingyun Huang
Student Number: 300019917

***Abstract*—We present an Augmented Reality application on mobile devices. The application can detect objects of different categories, track their movements, and display AR labels attached to the objects. The application supports tracking multiple instances from any class simultaneously, also, users can filter categories or select a single instance to track. Rather than using any AR engine, we researched on several object detection and tracking methods, and selected Single-Shot MultiBox detection method and Lucas-Kanade-based object tracking method. In order to work on the mobile device, Tensorflow Android API is used for performing inference, and mobile orientation sensor assists rendering labels in appropriate angle. The performance of the application was evaluated with various instances, under different light condition, and attention has been paid to object scaling, transformation and rotation, camera movement, and partial occlusion.**

***Index Terms*—Augmented Reality, Object Detection, Object Tracking**

I. Introduction

Augmented Reality (AR) is an emerging technology which integrates computer-generated perceptual information with real-world environment. It can be widely adopted to multiple areas, such as education, military, medical treatment and entertainment. Augmented Reality is quite different from Virtual Reality (VR). In Virtual Reality, users are fully immersed in artificial digital world. Special headsets are needed, like HTC Vive, Oculus Rift, Sony PlayStation or Google Cardboard, in order to shut out the real world and transport users into virtual environment. However, in Augmented Reality, users can interact with the real world with virtual object overlaid on it. Although there are AR glasses and other display devices, a simple smart phone with camera is enough to provide AR experience.

In order to integrate artificial digital information with physical world, AR system generally follows the pipeline of detection, tracking, and rendering. The first stage aims to decide whether there are predefined target objects that need to be augmented and detect their location. The second stage is to assign an unique id to each initial detected instance, and track their real-time locations as they move around meanwhile maintain the id. The third stage estimates the position and orientation to display the digital information.

Various AR SDKs are available nowadays, for example, Vuforia and EasyAR, which makes it easier for developers to create AR applications. They support native Android and iOS development, as well as Unity3D for cross-plateform development. They both offer APIs for planar image tracking, 3D object tracking, multi-target, cloud recognition and SLAM. Vuforia is one of the most popular AR platform, some of its special features are: first, it supports cylinder and conical target so that users can track images on a cylinder or conical surface; second, users can use an app, called Vuforia Object Scanner, to scan and create 3D object targets; third, it allows to attach content to planes such as floors and tables; fourth, it's especially good at occlusion handling, and can maintain a reference for object even when the object is no longer visible. The disadvantage of Vuforia is that it can not track more than 5 image targets or 2 3D targets simultaneously. EasyAR is a cheaper substitution of Vuforia. EasyAR provides demo for 2D object tracking, it's easy to follow the demo to create several trackers, define the tracking target by putting image and json explanation in assets folder, assign target to trackers, load virtual models and display it with regard to tracked image. Furthermore, EasyAR can store up to 1000 offline targets.

AR SDKs are easy to use and tend to shorten development time. However, not all functionality they provided are free. Besides, developers are unable to modify and improve the underneath algorithms for detection and tracking. To address the problem, we can develop AR applications using deep learning methods directly, without the help of any AR SDK. In this way, the detection and tracking stages can be customized for any specific situation. Besides, the model can be trained with suitable dataset to detect any object of certain categories, while most SDKs only support detecting the specific predefined instances. Therefore, this project uses deep learning methods.

This paper proposes the implementation of an Augmented Reality application on android device with Tensorflow Android API. The application aims to detect and track objects in space captured by camera, and displays Augmented Reality labels on top of them. The text labels are fixed to the centroid of tracked objects as camera moves around. For object detection, four categories of objects are supported, including laptop, bottle, person and book. The application enables tracking multiple categories and multiple instances simultaneously. User can filter the categories of objects to be tracked, as well as click the screen to select a certain instance to track. The application is efficient enough to give real time response on mobile devices, meanwhile robust to cope with camera movements and partial object occlusions.

II. Related Work

A. Object detection

Object detection aims to simultaneously classify and localize object instance in images or videos. The output of object detection usually includes the class label prediction, the bounding box position and size. The up-to-date deep learning-based object detection methods can be divided into one-stage methods and two-stage methods.

Two-stage methods propose region of interest before refining the bounding box position. The most popular ones are R-CNN family. R-CNN [1] first uses Selective Search [2] to generate Region of Interest, then affines and crops the proposed region into the same size as required by the AlexNet, next uses AlexNet to extract feature, finally applies SVM for classification and a linear regression model for bounding box regression. Fast R-CNN [3] uses VGG-16 instead of AlexNet for feature extraction, and swaps convolution layers and region proposal step, in order to share the convolution computation for overlapping proposals and speed up the network. Also, it discards SVM classifier and uses a single end-to-end network for multi-task learning. Besides, it introduces RoI Pool to generate fixed-length feature vectors. Faster R-CNN [4] replaces Selective Search by Region Proposal Network, which creates jaw-dropping speed improvements. The two-stage methods are usually accurate, but slow due to the intensive computation in region proposal stage.

While for one-stage methods, instead of proposing region of interest, they use default boxes to densely sample across the whole images, and perform classification and localization directly. The best-known one-stage methods are You Only Look Once (YOLO) [5] and Single-Shot MultiBox Detection (SSD) [6]. YOLO uses a simple network architecture containing only convolution, pooling and dense layers, however, the design of mapping relationship between input and output is fairly delicate. First, it splits the input image into S x S girds. Each grid is responsible for identifying the object whose ground truth box center located in this grid. Each grid cell predicts B bounding boxes, each bounding box corresponds to 4 location and size predictions, (x, y, w, h), and one confidence scores, Besides, each grid predicts P object class probability. The output size is S * S * ((4 + 1) * B + P). At test time, we multiple the confidence score and the highest class probability, to get a class-specific confidence score. SSD is very similar to YOLO, since both of them predict bounding box offsets and object scores based on default boxes. While SSD has 3 main improvements: first, it uses multi-scale feature maps to improve prediction accuracy; second, it uses pre-defined prior boxes of different size and aspect ratio, so that one grid can detect multiple objects; third, it changes fully connected layers into convolution layers.

The speed and accuracy are compared in [6]. As is clearly shown in Fig. 1, Fast R-CNN is 20 times faster than R-CNN, and Faster R-CNN is incredibly 350 times faster than R-CNN. Although accurate, faster R-CNN is still much slower than YOLO. SSD outcomes all other methods, and SSD300 is the fastest of all. In a more recent research [7], it's shown that the mAP of Faster R-CNN with inception backbone surpasses all other methods, while SSD with MobileNet backbone is the fastest.

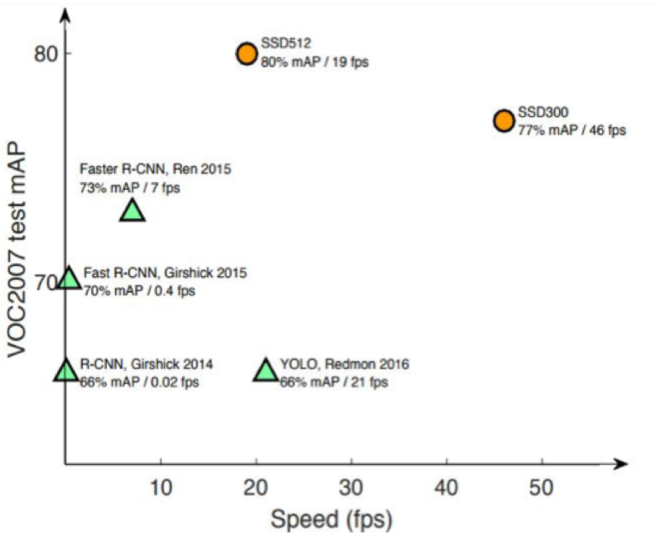

Fig. 1. Compare different methods in terms of speed and accuracy

Since we need to perform inference on mobile devices which have restricted computation capacity, the speed is the most critical determinant. Thus, we will use SSD model with MobileNet backbone for detection.

B. Object tracking

Object tracking takes the state of an instance at starting frame as input, and aims to analyze its states in subsequent frames. Object representation is the foundation of object tracking. Usually, features like points, articulated models, contours, or optical flow are considered in practice [8]. As for model update methods, there are template-based trackers and discriminative trackers.

In template-based methods, the instance to be tracked is represented by an image patch or the color histogram. The matching algorithm takes the bounding box location which has minimum distortion or maximum correlation regarding to the template as the output. Some popular methods for calculating the degree of similarity between the template and image are Sum of Absolute Differences (SAD), Normalized Sum of Squared Difference (NSSD), Cross Correlation (CC), Normalized Cross Correlation (NCC) [9]. Template-based methods are straightforward, while they can only model the object from certain viewpoint. Adaptive template-based methods gains improvement, since instead of extracting template representation from merely the initial frame, they use the tracking results of recent frames as template representation. There are also methods which combines both static methods and adaptive methods.

An alternative approach is discriminative tracking, which is suitable to handle several appearance changes. Discriminative methods classify the image frame into positive object class and negative background class. Some well known previous works of discriminative tracking are Multiple Instance Learning [10], Kernelized Structured SVM [11] and Tracking-learning-detection [8]. These methods have advantages in dealing with intances appearance changes and partial/full occlusion. However, they tend to cause drifting.

III. System

The Augmented Reality system enables the mobile device to capture frames in the field of view of its camera, detect and track four categories of objects in real-time, then attach the AR labels to object instances displaying their names. In general, the system can be roughly divided into 3 parts: detection methods, tracking methods and Graphical User Interface(GUI).

Detection and tracking are performed on the frames simultaneously which promote each others. The object detection model localizes the objects appear in the frame, then the tracker follows each detected instance from frame to frame. The tracker can estimate the motion much faster, while the error will accumulate. The detector can correct the tracking error and avoid drifting since it works on each frame independently. What's more, when an instance disappear in the frame, the tracker tends to fail, while the detector will help when the instance reappear, and trigger re-initializing a tracker.

The GUI can be further split into the view for previewing the frames captured by camera, the view for rendering the result, and the interactive panel for filtering categories or instance to track.

A. Development environment

a) Hardware

The detection model was trained on PC, and the android app utilized the model for detection inference and tracking on mobile phone. Thus, the following devices were used:

- PC (Asustek g58v, Intel i7-6700 CPU, NVIDIA 960m GPU, 16 GB RAM)
- Android phone (Huawei KIW-AL10, Octa-core 1.5 GHz CPU, CPU type of armeabi-v7a, 3.0 GB RAM, Android system version 6.0.1)

b) Software

For training the detection model, tensorflow object detection API was applied, and we used the following tools or dependencies:

- python 3.5
- tensorflow-gpu: 1.9.0
- cudnn: 7
- cuda: 9
- Protobuf: 3.0.0
- COCO API

For android development, Android Studio 3.2.1 was used. Here lists android gradle configuration:

- compile sdk version: 23
- build tools version: 26.0.2
- gradle-wrapper: gradle-4.1-all

Detection on android requires Tensorflow Android API, the version we used were tensorflow-android-1.12.0-rc2, and cmake was needed to compile the library. Android-ndk 15.2 was installed for reusing the tracking code library.

B. Implementation of detection

A Single-Shot Multibox detection model was trained with TensorFlow Object Detection API on COCO dataset. Then, the android application loaded the offline trained model for inference.

a) Training with tensorflow object detection API

TensorFlow Object Detection API is an open source framework, which not only contains a collection of various sample scripts and pre-trained models on different datasets, but also lists the comparison of model speed and accuracy.

After installing the required dependencies, we should prepare the training data. We took COCO 2017 validation dataset for training and validation, which contains images of 80 categories. Then, we modified 'create_coco_tf_record.py' to filter 'object_annotations' and skip categories not needed, and ran the command to generate data of '.TFRecord' format. Finally, we modified the 'mscoco_label_map.pbtxt' file which associates the category id and name.

For training, we downloaded pretrained SSD model with mobilenet_v1 backbone from [12], set it as the fine-tune checkpoint, and revised 'pipeline.config' file for our task. Next, we run the 'train.py' to fine-tunning the model. Finally, 'export_inference_graph.py' was used for exporting the model to '.pb' format, which recorded the frozen graph for inference.

b) Inference with tensorflow android API

On android devices, tensorflow android API offers a wrapper for using frozen detection models in android. We created a detector with the API by passing the '.pb' model, the '.txt' label name, and the input image size (300 * 300). The captured frames were pre-processed before previewing, then they are resized to 300 * 300 and feed into the detector. The detector first extracts R, G, B information from data bytes in 0x00RRGGBB form, then feeds the input into frozen inference model and runs the inference cell, next finds the best detection in tensorflow output, and finally returns the detection results. The detection is processed in background thread, and a boolean value marks whether the system is computing detection currently. If the system is computing detection on a previous frame currently, the newly coming frames will only be used for tracking and then discard. When the system finishes computing the detection, it is ready to compute both tracking and detection for a new frame. The detector returns a list of instances, each instance is represented by the category id, the category name, the confidence score and the location in resized image.

C. Object tracking with Lucas-Kanade tracker

The obeject tacker is initialized with the bounding box detection results. "FAST" feature keypoint detector [13] is used to extract a set of keypoints. Lucas-Kanade optical flow tracker estimates the displacement of the keypoints between the successive two frames. Then, the reliability of the estimated position of each keypoints is calculated. Only the points with reliability higher than 0.5 are used to estimate the displacement of the whole bounding box. For estimation, we calculate the median value of the displacement of all reliable keypoints in x and y direction independently.

The reliability is measured by Forward-Backward method [14], which assumes that tracker should work whether in chronological order or opposite to chronological order. We keep record k image frames $(I_t, I_{t+1} \ldots, I_{t+k})$. For forward tracking, the tracker predicts the moving trajectory of point x from time t to time t + k. For backward tracking, the tracker uses the predicted result at time t + k to estimate the object position $\hat{x}$ at time t. The larger the difference between the forward and backward trajectory, the less reliable the tracking result is. The reliability is calculated by Normalized Cross Correlation (NCC).

An object tracking library is reused which implements the above method and returns the instance position in next frame. We initialize a queue of colors, for each object detection result, a unique color is assigned, which can be regarded as id. So the number of predefined colors is exactly the max number of instances which can be tracked simultaneously. If the Normalized Cross Correlation value is smaller than a threshold, the object is regarded as not worth tracking, so object is removed from tracking list and the color is released for upcoming objects.

D. Graphical User Interface

a) Interface Design

The graphical user interface can be simplified as the following figure. The AutoFitTextureView and OverlayView are overlapped and previewing the captured frames and displaying the bounding box and labels respectively. For category filter, there are four checkboxes related to laptop, person, bottle, and book, all of which are listened to in real time. The ClickView applies to capture touch event, record the position where user clicks and draw a point at the position. When clicking 'confirm' the button at the bottom right of the screen, the system will find the instance whose center point is closest to the clicked point, and in the following frames, the system will track the clicked instance only. When clicking 'reset' button, the system will remove the restriction of the selected instance, thereby detect and track all objects.

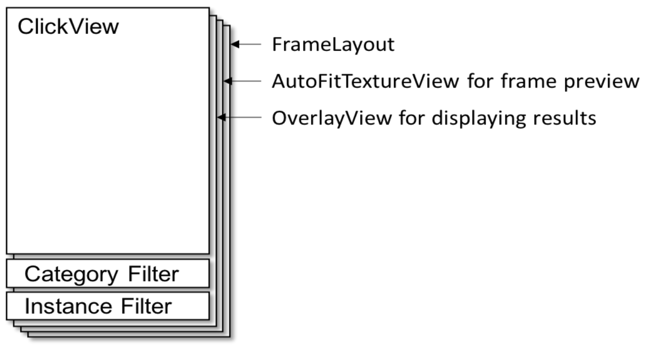
ClickView
FrameLayout
AutoFitTextureView for frame preview
OverlayView for displaying results
Category Filter
Instance Filter

Fig. 2. Interface design

b) Frame previewing

The application first requests permission to access to camera, then finds a suitable preview aspect ratio which is 640 * 480 on test phone, and sets preview orientation. Next, 'startPreview()' function is called to start capturing and drawing preview frames to the screen. 'onPreviewFrame()' function will be called when preview frame becomes available, which reprocesses the frame by converting YUV data into RGB data, which is required by the detection model.

c) Result rendering

The results are rendered on OverlayView. The bounding box is drawn according to detection and tracking results, and each tracked instance is assigned a unique color. At a single frame, 15 boxes can be displayed at most. Besides, an AR label is attached to the center point of the bounding box. Whats more, the sensor of mobile phone is utilized to keep the label parallel to the ground in the screen plane. The orientation sensor offers data of current orientation of phone, representing by yaw, pitch and roll angles respectively. The following formulas can be used to infer the angle that the text needs to rotate.

$$\frac{y}{x} = -\frac{\text{pitch angle}}{\text{roll angle}}$$

$$\beta = \operatorname{atan}\frac{y}{x}$$

$$\alpha = \beta - 90^\circ$$

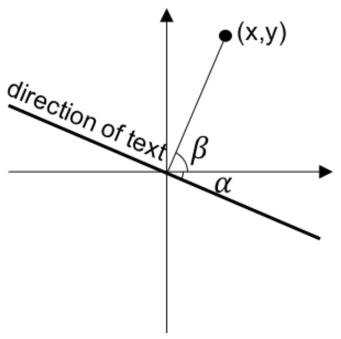

Fig. 3. Direction of text

Since multiple labels are drawn on one canvas, to avoid interference, when drawing each label, we need to rotate the canvas around the box center for $-\beta$ degrees, then draw the text, next rotate canvas in opposite direction.

IV. Experiment

A. Experiment Results

We tested multi-object tracking and the experiment results are shown in Fig.4. Then, we test single instance tracking in the same scene, while click the white bottle to track it individually, the result is shown in Fig.5. It can be seen from the figures that the tracker can deal with camera moving, object scaling and partial object disappearing.

TABLE I
AVERAGE RECALL IN BRIGHT ENVIRONMENT

	instance 1	instance 2	instance 3	instance 4	instance 5	average
bottle	1306/2103=62.1%	1870/2119=88.2%	893/2157=41.4%	1563/2110=74.0%	742/2157=34.4%	54.5%
person	1537/2203=69.8%	946/2139=44.2%	1366/2159=63.2%	1048/2171=48.2%	879/2107=41.7%	53.4%
book	956/2190=43.7%	581/2092=27.8%	281/2025=13.9%	1148/2061=55.7%	1659/2158=76.9%	43.6%
laptop	1408/2144=65.7%	-	-	-	-	65.7%

TABLE II
AVERAGE RECALL IN DIM ENVIRONMENT

	instance 1	instance 2	instance 3	instance 4	instance 5	average
bottle	1787/2412=74.0%	1329/2307=57.6%	682/2401=28.4%	1428/2097=68.1%	579/2403=24.1%	37.1%
person	1602/2097=76.3%	631/2094=44.2%	824/2129=38.7%	851/2118=40.2%	782/2103=37.2%	47.3%
book	1831/2326=78.7%	886/2210=40.0%	710/2098=33.8%	1814/2184=83.1%	1336/2368=56.4%	58.4%
laptop	1901/2338=81.3%	-	-	-	-	81.3%

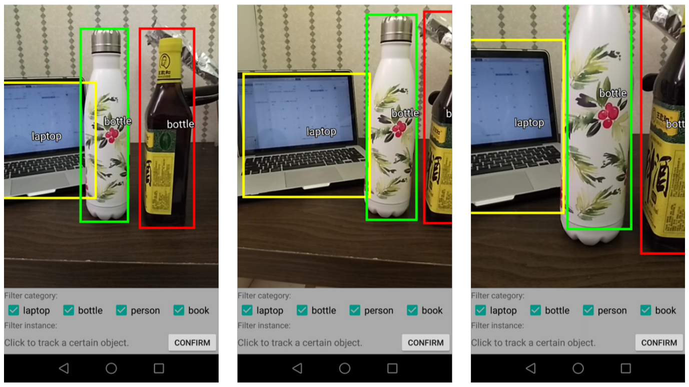

Fig. 4. Multi-object tracking result

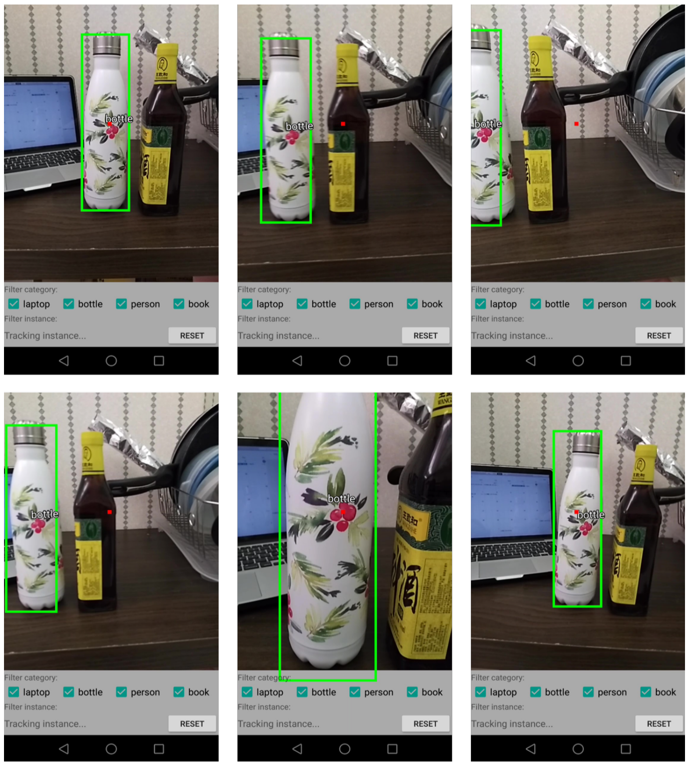

Fig. 5. Single instance tracking result

B. Performance evaluation

The performance of the system can be evaluated in terms of speed and accuracy. The detection costs about 1.3 seconds per frame on a low-end android phone, while the tracking is rather fast and can generate result at real time.

As for recall, we chose 5 instances of each categories, and used the application to detect and track each instance for 100 seconds. During the 100 seconds, we held the mobile phone by hand and conducted different movement to test the robustness. The movement was roughly organized as follows. For the first 10 seconds, we tested scaling by moving close to the instance and then moving backward. For the next 20 seconds, we gradually changed the angle of view and looked down from above, then restored to original states. For the next 30 seconds, we slowly walked around the object. For the next 10 seconds, we moved the mobile phone to test the translation while keeping the distance between phone and object approximately unchanged. For the next 20 seconds, we rotated the phone in the plane vertical to the ground. And for the last 10 seconds, we tested object translation in a more complex background. For calculating average recall, the number of frames corresponding to true positive bounding box is divided by the total number of frames, the following table presents the results.

Considering the influences of light, it's clearly shown that the application performs better in bright environment when copying with bottles and people. Unusually, books and laptop can be detected and tracked better in dim environment. Some possible reasons are: first, the mobile camera enhance the images automatically when the whole view is dark. Second, the covers of books reflect light in bright environment which affect the tracker. And the screen of laptop is more distinct in dim light. Third, we only considered whether the tracker can traker the object, while ignore the bounding box localization accuracy which is hard to improve in dim light.

As is shown in the table, the performance differs for different instances. Take the bottles as an example, the first two instances are in solid color and easy to detect. While instance 3 and 5 are much harder, since instance 3 is a semitransparent beer bottle, and instance 5 is a transparent water bottle.

Observing the recorded results of different camera movements, the most challenged ones are rotating the mobile phone and walking around the object. Object translation is the easiest task, and object scaling can also be coped with unless the camera is extremely close to the object. What's more, the application is robust to complex background.

V. Conclusion

For this project, we successfully developed an AR labelling application on mobile devices. We implemented the detection and tracking methods which support each other. On the one hand, detection is time-consuming on mobile devices with limited computation capability, while tracking can estimate the position pretty fast. On the other hand, detection results are used to initialize trackers, and detection helps revise accumulated errors in tracking. Instead of using the AR API, we can customize and further improve the model used, we can detect then track unknown objects, and we are able to upgrade the max number of trackers with no limitation. An experiment has been conducted, which shows the system is fast enough to achieve real time tracking, and can effectively tackle with problems like camera/object movement, light changing, partial occlusion.

References

[1] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[2] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[3] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[4] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[5] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[6] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[7] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *IEEE CVPR*, volume 4, 2017.

[8] Zdenek Kalal, Krystian Mikolajczyk, Jiri Matas, et al. Tracking-learning-detection. *IEEE transactions on pattern analysis and machine intelligence*, 34(7):1409, 2012.

[9] Wisarut Chantara, Ji-Hun Mun, Dong-Won Shin, and Yo-Sung Ho. Object tracking using adaptive template matching. *IEIE Transactions on Smart Processing & Computing*, 4(1):1–9, 2015.

[10] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. Visual tracking with online multiple instance learning. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 983–990. IEEE, 2009.

[11] Sam Hare, Stuart Golodetz, Amir Saffari, Vibhav Vineet, Ming-Ming Cheng, Stephen L Hicks, and Philip HS Torr. Struck: Structured output tracking with kernels. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):2096–2109, 2016.

[12] Huang J. ssd mobilenet model. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md.

[13] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[14] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Forward-backward error: Automatic detection of tracking failures. In *Pattern recognition (ICPR), 2010 20th international conference on*, pages 2756–2759. IEEE, 2010.