# Object Tracking with Darknet and INT8 Inference

Yuval Levental
Rochester Institute of Technology
Rochester, NY 14623 USA
yhl3051@rit.edu

## Abstract

*Object tracking between video frames is a significant topic in computer vision. Most of the underlying principles of object tracking are well-established. One promising method for video frame tracking is the use of deep convolutional neural networks. These networks are used to track and classify objects between frames with the Tracking by Detection method. Additionally, one major challenge is increasing tracking speed, since a significant number of mathematical operations are required for video inference. This challenge is important for small devices such as drones, as they are limited in size, power, and weight. Darknet, an open source neural network in C, is the chosen framework for this project due to fast calculation speed and a wide range of usability. The main technique to significantly improve speed is INT8 quantization using YOLOv2. Ideally, both inputs and weights will be quantized to integers that range from 0 to 255.*

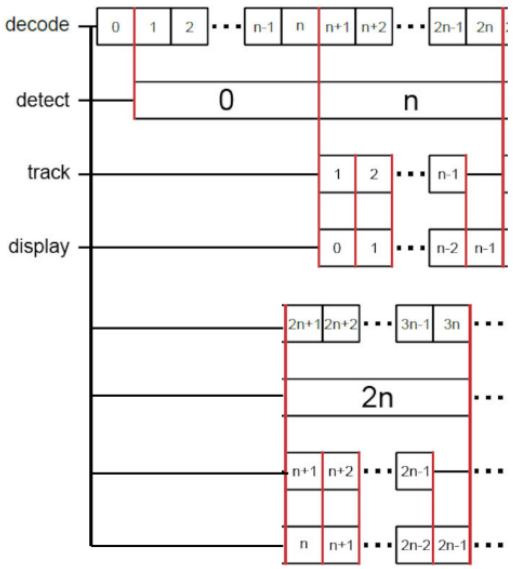


Figure 1. A visual depiction of the object tracking method. There are four attributes: decode, detect, track, and display. Decode reads in the frames, and detect searches for the object in every set of n frames. Track and display restore the original timeline for the object after detections have been completed.

# 1. Introduction

Computer vision and deep learning are important for technologies such as assisted driving, virtual reality, and collision detection. The main focus of this project is object detection and tracking from video frames. Object detection can be performed on a single frame because only the x and y coordinates matter. Object tracking, on the other hand, involves attempting to predict the motion of an object over several video frames.

The most common method for object tracking is Tracking by Detection [TBD], which performs object tracking by detection on every single frame. There is lots of resulting data available, but the implementation is inefficient. Additionally, TBD is not suited for effective movement

predictions because it only tracks from the past to the future.

The most promising solution to this problem will involve a combination of traditional and artificial intelligence [AI] based algorithms. Traditional algorithms are very efficient, but they do not work for multiple situations, which well-trained CNNs can handle. One strategy is to analyze an sampled fraction of frames, taken from the video played at a relatively high FPS.

The proposed algorithm for this project is a lightweight tracker which will quantize the input weights to INT8 values. The algorithm will be based on a CNN object detector, and will incorporate optical flow, or apparent motion between an observer and a scene. Additionally, the algorithm will have a mechanism to "look back" at previous frames to determine an overall prediction of motion. The algorithm will be designed to work on a wide variety of hardware.

# 2. Related Work

For this algorithm, the object classifier should be run at a significantly slower rate than the object tracker. This is because objects typically move much faster than change class in a video. Additionally, the tracker algorithm should be relatively simple, though would also have deep learning key frames. As a result, the traditional tracker can be loaded on every key frame.

One objective of quantization is to reduce the latency of the tracker without significant accuracy degradation. This is highly possible through efficient utilization of memory resources. There is already lots of research that involves interpolated edge-based object detection, but the CNN inference causes significant latency. Motion-based interpolation to reduce latency has already been used to modify 3D point cloud data.

In general, the quantization of neural networks has been widely adopted. For instance, the ARM Compute Library supports quantized neural networks [16]. One existing example of object detection using quantization is the FINN-R Framework, developed in 2018. FINN-R is end-to-end, and automatically quantizes for platforms and targets based on the given precision This tool enables design space creation and automates the synthesis of customized FPGA engines [1].

The closest implementation of quantized look-back tracking is the Glimpse algorithm, developed by Chen et al. [4]. Due to device constraints, relatively little data was collected but there were several compensatory mechanisms in place. These include mostly tracking edge points of objects, and caching the motion information in a remote server [12].

Regarding key frames, Cintas et al. proposed vision-based tracking by use of YOLOv3 and kernelized correlation filters [KCF] [5] [7]. Their algorithm used deep learning key frames to generate correlation templates that can be used in between the frames. While this method has proven to be immune to translation, is is not immune to rotation or scaling [2].

# 3. Proposed Methods

The main proposed method will similar to the Glimpse algorithm, but an ROI-to-centroid algorithm and motion pre-calculation routines are going to be added to reduce technical requirements [6]. Object detection and interpolation will be done with the YOLO algorithm and optical flow methods [11].

## 3.1. Object Detection

The detectors that will be used for this project include YOLOv2, YOLOv3, and YOLOv4. YOLO algorithms are the best choice for edge inference. Object detectors with more layers, like YOLOv3 and YOLOv4, will perform better on smaller objects. Accuracy is more important than speed in this case, since the detection of small objects is important.

YOLOv3 is a good baseline algorithm, since the resulting detections are accurate and data

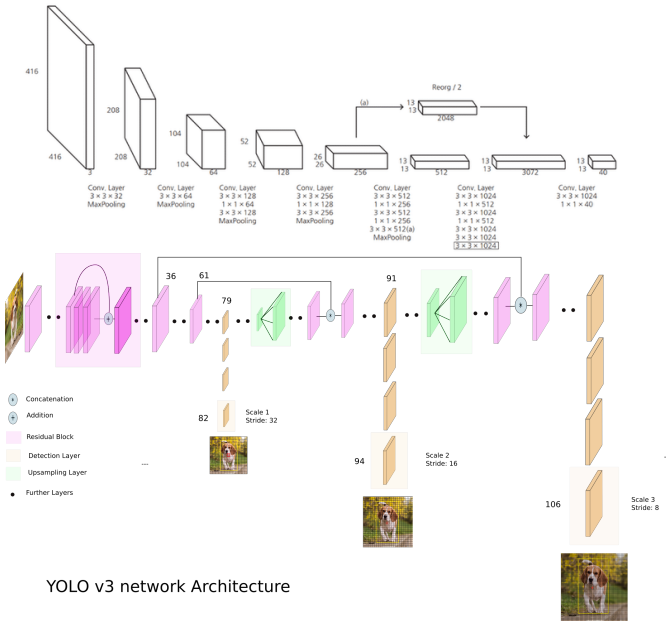


Figure 2. Diagrams of YOLOv2 and v3 networks. YOLO is a CNN architecture for object detection. This architecture only uses a single CNN, which makes the network very fast. The entire image is processed at once, which puts objects into a wider background context. YOLO can also be easily applied to new domains and inputs.

skipping can be observed. The algorithm takes a relatively long time to run, but the code is excellent at small object detection. YOLOv2 has decent accuracy and has the smallest weights, so it is good for testing low-memory applications. YOLOv4, by contrast, has the highest FPS and can still work with low-memory applications.

## 3.2. Motion Estimation

Lucas-Kanade Optical flow motion estimation is the best choice for detection, as the algorithm has a fast running time and works well with OpenCV. Optical flow block sizes of 8 × 8 pixels will be utilized for this purpose. Harris Corner Detection will also be used for ROI points. Overall, various full frame, ROI, and ROI-to-centroid tracking schemes will be tested.

$$I_x(q_1)V_x + I_y(q_1)V_y = -I_t(q_1)$$

$$I_x(q_2)V_x + I_y(q_2)V_y = -I_t(q_2)$$

$$\vdots$$

$$I_x(q_n)V_x + I_y(q_n)V_y = -I_t(q_n)$$

The method is based on motion interpolation of video object detection. Latency will be reduced by adding logged motion to previous detection information. The predicted tracking result is displayed to a frame before the CNN detections on that frame are finished. Overall, the best strategy will be to perform inference on various key frames, then interpolate object motion using intermediate frames.

$$\begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix} \begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix}$$

## 3.3. Main Features

The main method will be motion interpolation in a video sequence. Tracking latency will be reduced by frequently adjusting the detection information with recorded motion vectors. This algorithm is designed to output a tracking result for a frame before detection on the frame is finished.

To speed up the frame rate of object detection, inference on key frames is performed, and motion is interpolated. The skip frames are where data is interpolated from one frame to the next. The overall inference will be in units of three frames.

Inference starts when the object enters the first frame, which is denoted as Frame 0. The inference is then delayed by three frames. Up to 15 key points are used to calculate the actual location of the object. A search is used to determine which points were in the original box, and which points were in the box of Frame 3.

After reducing the number of points, the motion vectors are averaged, then moved three frames ahead to Frame 6. The predicted box will be slightly different from the actual bounding box, returning the error. At this stage, ROI-to-centroid tracking can now be used based on the

Figure 3. Example interpolated results from Lucas-Kanade Optical Flow. The lines represent the main keypoints that were tracked, which were retrieved from an edge detection algorithm. The lines are not smooth because they were averaged over three frames each.

other previous frames. This makes tracking faster and more efficient. Every three frames, there is new inference information to mitigate the motion tracking error.

The next method, the accumulated motion method, starts at Frame 9. The original ROI is moved forward three frames using the stored motion vector that links frames 6 and 9. This is more efficient than the original ROI method, since the points in the bounding box do not need to be determined, which results in less latency. If the tracker does not have motion marching capability, then the frame track is always off by one. Using the proposed look back method, this latency is eliminated.

Finally, this method uses a "keeping" mechanism that will keep the object's box alive for a certain number of frames just in case that the object is not detected. In this case, the algorithm will generate tracking data for three frames, and the data is linked using Intersection-over-Union [IoU] matching. The keeping feature pulls in the last detection result from previous tracking data and re-uses it in case the CNN does not predict a bounding box with enough confidence for an object.

# 4. Frameworks and Datasets

For this project, Darknet, developed by J. Redmon, will be used as the object detection framework. Darknet is a good choice, as it is fast, easy to work with, and does not require a GPU. The YOLO models that were used are accessible from the current version of Darknet on GitHub. The selected models will be trained on the datasets and will be left unmodified.

**Algorithm 1:** Entropy Calibration for INT8

**Input:** FP32 Histogram H
  **for** i in range (128,2048) **do**:
    ref_dist = [bin[0],...,bin[i-1]]
    outliers_count = sum(bin[i],bin[i+1],...)
    ref_dist[i-1] += outliers_count
    P /= sum(P)
    cand_dist_Q = quantize([bin[0],...,bin[i-1]])
    divergence[i] = KL_divergence(ref_dist,
    cand_dist_Q)
  **end for**
  `// Find minimal divergence`
  threshold = ( m + 0.5 ) * ( width of a bin )

As mentioned earlier, the YOLO models used are YOLOv2-Tiny, YOLOv3, YOLOv4, and YOLOv4-Tiny. All of the YOLO models were trained using the MS-COCO dataset and originally were ImageNet pre-trained weights. Additionally, the models come from the current version of Darknet. Training for special circumstances is not required for this project [14].

One important metric that will be recorded for the dataset is the bounding box overlap with the IoU ground truth, which will be indicated as a percentage. Additionally, the normalized centroid drift [NCD] metric will be measured as a distance from the ground truth to the predicted result. The only sequences that will be analyzed are ones that contain a tracked object.

Finally, the Optical Training Benchmark [OTB] by Wu *et al.* will be used as a method of assessment [17]. Singular object tracking was designated for the testing section. The sequences had

to track all of the entire object. A large portion of the OTB is for the testing set. All results will be compared to the best possible scenario, where motion tracking is not used, and the detector can be run on each image frame.

# 5. Experiments

## 5.1. Implementation

The first step is to create the INT8 look-back algorithm, and test on typical GPUs and processors. This is so that baseline measurements can be successfully conducted. The main programming languages used were C and MATLAB. C was chosen because the code is good for maximizing performance on low-power computers. MATLAB was chosen because of its Deep Learning Toolbox, which is useful for designing Deep Neural Networks [9]. MATLAB also provides greater flexibility and control for the user.

For the main algorithm, a single shot detection method was chosen over a region proposal CNN. This is because single shot detection requires much less resources than region proposal. Baseline inference was performed on a desktop computer, using Darknet for the main code and OpenCV for various image analysis and display purposes. Optical flow was included in the algorithm's inference loop so that interpolation between frames could be performed.

The Lucas-Kanade optical flow algorithm was quantized, and modified so that motion information could be cached with look back to reduce frame latency. The algorithm was further modified so that the detections would happen on a separate device, and processing would still occur on the main computer. This is classified as a multi-threading algorithm because the computations happen on multiple locations, and are combined in a later sequence.

Quantization was performed differently in C and MATLAB. In the C program, the weights for each convolutional layer were multiplied by a number that was calculated based on the range

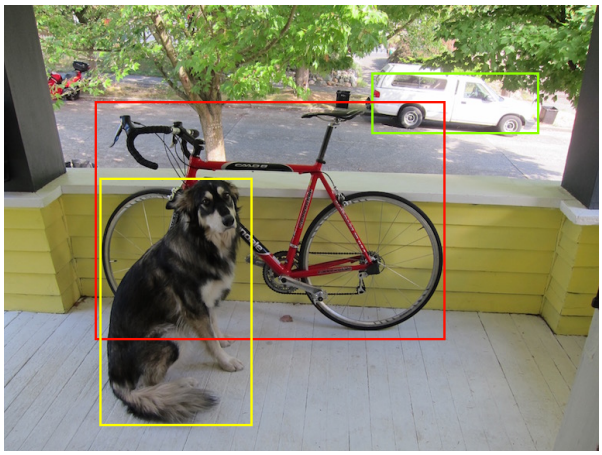

Figure 4. Object detection with regular YOLOv3 weights in C. All objects are accurately detected, including the bicycle.

and standard deviation of the weights. The number is a power of two, and the results are rounded. In the MATLAB program, the weights are not initially quantized, but the code uses fixed point arithmetic to calculate the motion.

The accuracy metric is obtained by finding the percentage of positive frames of the base CNN inference. The focus in this case is on the IoU metric. However, the main measurement in this case will be the normalized centroid drift metric. This is defined as the direct distance from the predicted object centroid to the baseline bounding centroid. Additionally, all the components are normalized in the horizontal and vertical directions [13].

## 5.2. Results in C

| Weights | Time (s) | Size (MB) | Accuracy |
|---|---|---|---|
| YOLOv3 | 6.533 | 248.013 | 97.333% |
| Tiny | 0.523 | 35.443 | 63.333% |
| INT8 | 0.394 | 18.297 | 37.333% |

Table 1. Comparison of detection performance with respect to various weights. Factors include detection time, weight file size, and classification accuracy.

All three weight files were tested on a sample image from the official Darknet website: dog.jpg, which contains three objects: A bicycle, a dog, and a pickup truck. The dog and bicycle closer to

| Epoch | Iteration | Time Elapsed (hh:mm:ss) | Mini-batch RMSE | Mini-batch Loss | Base Learning Rate |
|---|---|---|---|---|---|
| 1 | 1 | 00:00:01 | 11.54 | 133.3 | 0.0100 |
| 1 | 50 | 00:00:05 | 4.51 | 20.4 | 0.0100 |
| 2 | 100 | 00:00:08 | 3.48 | 12.1 | 0.0100 |
| 2 | 150 | 00:00:11 | 3.27 | 10.7 | 0.0100 |
| 3 | 200 | 00:00:14 | 2.97 | 8.8 | 0.0100 |
| 3 | 250 | 00:00:17 | 2.78 | 7.7 | 0.0100 |
| 4 | 300 | 00:00:20 | 2.65 | 7.0 | 0.0100 |
| 4 | 350 | 00:00:24 | 2.17 | 4.7 | 0.0100 |
| 5 | 400 | 00:00:27 | 1.96 | 3.9 | 0.0100 |
| 6 | 450 | 00:00:30 | 2.56 | 6.6 | 0.0100 |
| 6 | 500 | 00:00:33 | 2.29 | 5.2 | 0.0100 |
| 7 | 550 | 00:00:36 | 2.09 | 4.4 | 0.0100 |

Figure 5. Command line output for the network training script in MATLAB. The base learning rate is constant, and the RMSE and loss quickly decrease in the first few epochs.

the front of the image, and the pickup truck is part of the background.

Due to time constraints, the only data that was obtained for the C program was a comparison of different weight file detections for the created algorithm. Obviously, the YOLOv3 weight file from the official Darknet website is the most accurate. However, the file size is definitely the largest and the computation time is the longest. All three objects were easily detected.

The custom YOLO algorithm in C was based on a freely available algorithm titled yolo2_light by Alexey Bochkovskiy [3]. Yolo2_light can process images faster, and works with YOLO v2 and v3 weights. This algorithm also supports INT8-inference and BIT1-XNOR-inference. The latter inference was only available for custom models, though it was unused.

Tiny YOLOv3, also from the official Darknet website, managed to detect all three objects in a much shorter time with a smaller file size. However, accuracy was significantly lower, and the bounding boxes for the dog and the bicycle overlapped more compared to YOLOv3. Additionally, the program had difficulty determining whether the pickup truck was a car or a truck, which are relatively similar for a truck of this size [15].

The generated YOLOv3 INT8 weight file was slightly faster, but the major difference was that its file size was about half of the file size of Tiny

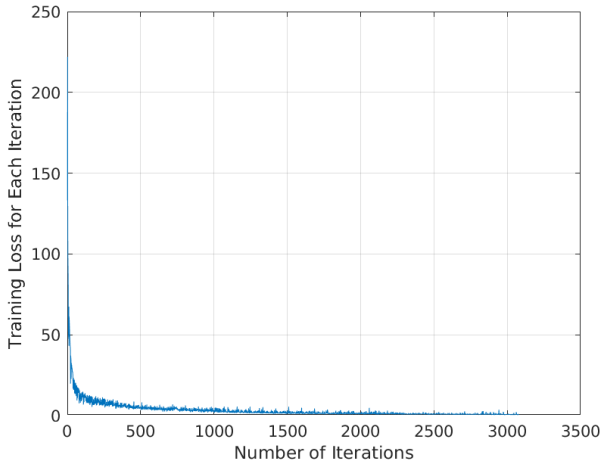


Figure 6. Training loss of the look-back model vs number of iterations in MATLAB. Training loss significantly decreases with more iterations.

YOLOv3. This is because INT8 numbers take up much less space than FP32 numbers, which is the format of regular YOLOv3. The overall accuracy is much lower because while the dog and truck are successfully detected, the bike is not detected at all. This is because the bike is less opaque than the dog and truck, due to the bike frame being relatively thin.

## 5.3. Results in MATLAB

Most of the work which was performed in MATLAB involved converting object detection code from double to fixed point precision. The name of the algorithm used is Ravven YOLO,

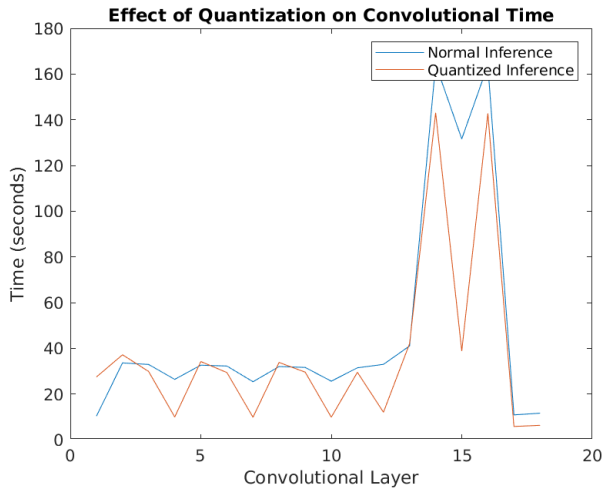


Figure 7. Comparison of quantization time for each layer between normal and quantized code. The quantized code has significantly lower times.

which is written in MATLAB and taken from Ravven Labs at RIT. The convolutional layers use a for loop, as fixed-point conversion is not compatible with MATLAB's convolution functions.

Ravven YOLO is similar to Darknet-19, but two convolutional layers were removed resulting in a total of 59 layers. The resulting model has 17 convolutional layers, including 5 max-pooling layers. The initial network was trained on 295 images, and is used for both the regular and quantized Ravven YOLO code [10]. About 60% of the images were used in the training dataset, and 40% of images were used in the testing dataset.

For this YOLO model, Leaky ReLU layers are used. The scale is chosen to be 0.125, since this constant fits the image parameters. For maximum accuracy, the stride is simply [1 1]. The convolutional padding values are either [0 0 0 0] or [1 1 1 1]. The largest convolutional layer contains 512 3x3 convolutions.

A network training script was developed to successfully create a training model based on the images. First, the data is preprocessed so that the image resolution is significantly reduced. The original size is 448x448x3, and the new size is 224x224x3. There were approximately 3000 training iterations. The mini-batch RMSE and Loss exponentially decreased, and the base learning rate was set to a constant of 0.01. For the RMSE, the value decreased from 11.54 to 0.88, and for the loss, the value decreased from 133.3 to 0.8. The total training time was only 3:23.

The Fixed-Point Designer app was used to convert the convolutional code [8]. Conversion proved to be difficult, as many components of Ravven YOLO were not compatible with the fixed-point conversion app. Initially, the chosen word length for the converted code was 16, but detection results did not show up due to highly limited accuracy. The word length was increased to 64, and while the new program is faster, the new detection results are almost exactly like the original results.


Figure 8. Sample bounding box for the MATLAB car dataset. This was generated using the conv2 algorithm, and the confidence is very high, because the number is very close to 1.0.

After quantization, the model is run on the test data, which evaluates the model's precision and recall. The detection results are then visualized by drawing the predicted box around the detected object. One possible outcome of this project is that the algorithm can target generic FPGA hardware.

# 6. Conclusion

Overall, there was less progress on this project than was anticipated. This was due to difficulties with coding various aspects of this project, especially given that the convolutional code in MATLAB could take a very long time to run. The main result that was proven to be true is that weight quantization only causes a relatively small accuracy loss for a single frame, with significant savings in computation time and/or file size.

For the C code, the computational speed was only slightly faster, but the weight size was much smaller. The opposite was true for the MATLAB code, where the exact same weights were used but the computational speed was significantly faster. The C code is much simpler than the MATLAB code, so it would be harder to simplify the C code.

Only the YOLOv3 weights were really tested. It would have been interesting to compare YOLOv3 to different weight detectors to see what the differences would have been. At least the research into different optical flow methods, especially Lucas-Kanade optical flow, could be useful for understanding the differences between continuous frames.

# 7. Future Work

Clearly, the next objective would be to actually compare motion detection using the standard and quantized weights using the Lucas-Kanade algorithm. However, based on the current results, there might be a relatively small loss in object tracking accuracy. However, the effect of the loss could be more significant when multiple frames are taken into account.

Additionally, quantized algorithms have been shown to be more effective with clearly visible, opaque objects as opposed to objects with a thinner frame like the bicycle, which has more open spaces inside the frame. Possibly, the quantization could somehow be modified to take into account these open spaces in relation to the entire object.

# References


[1] Michaela Blott, Thomas Preusser, Nicholas Fraser, Giulio Gambardella, Kenneth O'Brien, and Yaman Umuroglu. Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks, 2018. 2

[2] Erik Bochinski, Tobias Senst, and Thomas Sikora. Extending iou based multi-object tracking by visual information. *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, 2018. 2

[3] Alexey Bochkovskiy. yolo2_light. GitHub, 2019. 6

[4] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. Glimpse. *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, 2015. 2

[5] Emre Cintas, Baris Ozyer, and Emrah Simsek. Vision-based moving uav tracking by another uav on low-cost hardware and a new ground control station. *IEEE Access*, 8:194601–194611, 2020. 2

[6] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Detect to track and track to detect. *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017. 2

[7] Joao F. Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):583–596, 2015. 2

[8] The MathWorks Inc. *Fixed Point Designer*. Natick, Massachusetts, United States, 2019. 7

[9] The MathWorks Inc. *Deep Learning Toolbox*. Natick, Massachusetts, United States, 2021. 5

[10] Daniel Kaputa. Ravven yolo. GitHub, 2019. 7

[11] Luyue Lin, Bo Liu, and Yanshan Xiao. An object tracking method based on cnn and optical flow. *2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, 2017. 2

[12] Luyang Liu, Hongyu Li, and Marco Gruteser. Edge assisted real-time object detection for mo-

bile augmented reality. *The 25th Annual International Conference on Mobile Computing and Networking*, 2019. 2

[13] Binh X. Nguyen, Binh D. Nguyen, Gustavo Carneiro, Erman Tjiputra, Quang D. Tran, and Thanh-Toan Do. Deep metric learning meets deep clustering: An novel unsupervised approach for feature embedding, 2020. 5

[14] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 4

[15] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018. cite arxiv:1804.02767Comment: Tech Report. 6

[16] Dawei Sun, Shaoshan Liu, and Jean-Luc Gaudiot. Enabling embedded inference engine with arm compute library: A case study, 2017. 2

[17] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Online object tracking: A benchmark. *2013 IEEE Conference on Computer Vision and Pattern Recognition*, 2013. 4