DeepDish: Multi-Object Tracking with an Off-the-Shelf Raspberry Pi

Matthew Danish
Cambridge University
mrd45@cam.ac.uk

Justas Brazauskas
Cambridge University
jb2328@cam.ac.uk

Rob Bricheno
Cambridge University
rwhb2@cam.ac.uk

Ian Lewis
Cambridge University
ijl20@cam.ac.uk

Richard Mortier
Cambridge University
rmm1002@cam.ac.uk

ABSTRACT

When looking at in-building or urban settings, information about the number of people present and the way they move through the space is useful for helping designers to understand what they have created, fire marshals to identify potential safety hazards, planners to speculate about what is needed in the future, and the public to have real data on which to base opinions about communal choices. We propose a network of edge devices based on Raspberry Pi and TensorFlow, which will ultimately push data via LoRaWAN to a real-time data server. This network is being integrated into a Digital Twin of a local site which includes several dozen buildings spread over approximately 500,000 square metres. We share and discuss issues regarding privacy, accuracy and performance.

CCS CONCEPTS

• **Computer systems organization → Sensor networks**; • **Hardware → Sensor applications and deployments**; **Sensor devices and platforms**.

KEYWORDS

object detection, object tracking, edge computing

ACM Reference Format:
Matthew Danish, Justas Brazauskas, Rob Bricheno, Ian Lewis, and Richard Mortier. 2020. DeepDish: Multi-Object Tracking with an Off-the-Shelf Raspberry Pi . In *3rd International Workshop on Edge Systems, Analytics and Networking (EdgeSys '20), April 27, 2020, Heraklion, Greece.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3378679.3394535

1 INTRODUCTION

We propose a system of edge devices based on a low-power computing board, the Raspberry Pi, that analyses the movement of people using a standard camera peripheral and can publish real-time events and anonymous statistics to a low-bandwidth and secure network. The system runs several machine-learning-based algorithms to perform multi-object tracking (MOT) on sequential image data and distils the information down to a few numbers that can be transmitted over a LoRaWAN network from sensors in the field.

The MOT problem takes as input an image sequence and a set of objects of interest. Solutions must trace their movement throughout the image sequence while maintaining the distinct identity of each object. We identify objects of interest by category (e.g., 'person') and then apply category-specific object-detection methods to automatically draw bounding boxes around all objects of interest within each frame. Solving the MOT problem then requires finding the corresponding bounding boxes in each successive frame, as well as determining if a bounding box is being drawn around a newly introduced object, and if a previously-known object has disappeared. This is known as the *tracking-by-detection* approach [4].

The object-detection problem has been studied intensively and solutions have advanced rapidly in recent years thanks to the success of supervised deep learning methods [5]. We focus in particular on the MobileNet model architecture [7] because it is optimised for use in the mobile processor context, and we use a model trained on the COCO data-set [11] that offers 91 possible object categories.

For tracking-by-detection, we employ DeepSORT [17], which extends SORT [2] with a 'feature encoder' that extracts a vector using a pre-trained convolutional neural network (CNN) on the image data within each bounding box. SORT is designed to be fast because it relies only on simple techniques of association based on scores computed using Kalman filtering and the Mahalanobis distance metric and then solved using the Hungarian method, but SORT can easily be fooled by object occlusion. By adding the feature encoder vector to the mix, DeepSORT helps avoid accidental identity switches between overlapping objects, while maintaining much of the performance required for online tracking of real-time video.

We show that this can be achieved with a low cost, stock Raspberry Pi 4B thanks to some unexpected findings we discuss in §7.

2 RELATED WORK

Past work by Ren et al. [14] relied on a network of edge-based servers with high-performance GPUs that could be placed close enough to gather image data from local network of cameras. Cartas et al. [3] performed object-detection by sending video frames from mobile devices to nearby small servers backed up by high-performance but more distant servers; they were only able to achieve 150ms inference time by equipping the small servers with GPUs. EdgeEye [12] similarly depends on having a GPU. Hochstetler et al. [6] bench-marked a Raspberry Pi 3B processor both with and

EdgeSys '20, April 27, 2020, Heraklion, Greece

ACM ISBN 978-1-4503-7132-2/20/04…$15.00
https://doi.org/10.1145/3378679.3394535

without an Intel Movidius™ Neural Compute Stick on data from a visual recognition challenge. When setting the model input image size to 224x224, they found that the CPU alone performed at about 2.1 frames-per-second (FPS) on the object detection task, while adding the Neural Compute Stick boosted that to 17.2 FPS. The DeepX project [9] looked at ways of distributing inference tasks across heterogeneous resources such as GPUs and lower-power processing units present on certain mobile platforms, with which they were able to achieve sub-500ms object-detection inference times at significant energy savings.

DeepSORT is a popular tracking algorithm; one recent and notable work using it is by Zhang et al. [18] who considered the case of fixed-view cameras: they computed a differential filter to isolate only the portions of the view that were changing, then used the YOLOv2 [13] object-detection system with DeepSORT to perform online MOT on a high-performance server and GPU. Al-Tarawneh et al. [1] used a different style of feature vectors computed on high-performance servers to re-identify customers over the course of a day as they browsed a shop, in order to produce retail analytics of their behaviour.

3 EXPERIMENTAL SETUP

Our pilot project intended[1] to count the number of people entering and exiting a particular building for fire safety purposes. While person-counting is a venerable field, this is only intended as a test case for a much larger experiment that will study the movement of people in public space as part of a project to create a Digital Twin [8] of a local site, which includes several dozen buildings and covers approximately 500,000 square metres of land.

With a large enough field-of-view, we hypothesise that the counting task can be performed even if the software can only process a low number of FPS, because people would enter the view for at least several seconds. Careful placement of the camera is critical to ensure that people are within view for sufficient time on either side of the pre-determined 'counting line', while not being too far away to compromise the object detector.

3.1 Hardware

Our edge node is a Raspberry Pi 4B, which is a Broadcom BCM2711 system-on-chip with a quad-core Cortex-A72 (ARM v8) 64-bit processor running at 1.5GHz, with 4GB of RAM. We used a night-vision camera module recording video at 640×480 in full colour, however we also tested with the standard Pi camera, which provides a similar quality of video albeit without the infrared lighting support. We added the Fan SHIM for temperature control, and a PiJuice battery 'HAT' mounted on the GPIO interface to provide a power supply backup. The device is packaged in a 3D-printed case that allows the camera to be rotated into the desired position, as shown in Figure 1. Future work will include a LoRaWAN HAT as well.

3.2 Operating software

We use TensorFlow 1.15.0, including the TensorFlow Lite engine, and the controlling software is written in Python 3.7 running under Hypriot, a Debian-based operating system customised for Raspberry Pi.

[1]Late note: the pilot project has been cancelled due to the global COVID-19 pandemic.

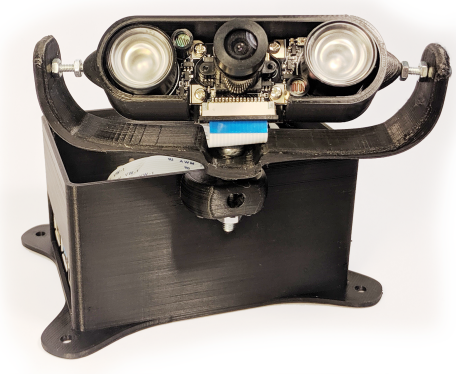

Figure 1: The counting device.

3.3 Algorithm

Following the tracking-by-detection concept used by DeepSORT, we break down the algorithm into three steps shown in Figure 2: object-detection, feature-extraction and tracking/association.

The object-detection model is a pre-trained quantised version of SSD MobileNet v1 [7] compiled for TensorFlow Lite, made available by Google. We feed it images of size 300×300, and from the output use only the detections that are labelled with the category 'person'. The feature-extraction step uses the DeepFLOW CNN model trained on the Motion Analysis and Re-identification Set (MARS) data-set [15] using the *cosine metric learning* technique [16]. Feature-extraction must be run separately on each person detected, therefore the algorithm scales linearly by the number of people that need tracking. Association of tracks with known history of objects is performed by the DeepSORT method of combining Mahalanobis distance computed on Kalman Filter distributions and cosine metric distance computed on extracted feature vectors. People who are new to the tracking history (based on a feature threshold) are assigned a fresh identification number, and people that fail to be found for over 30 frames are considered to have left the scene.

At each step the most recent track vector of each known person is compared against the pre-determined counting line by solving for line intersection and cross-product (to determine direction of movement). When an intersection is found, a 'count' event is generated, and the intersecting segment is highlighted in red, as shown in Figure 3. Internally, we maintain a running total of counts going in each direction, as well as the number of tracks that have been deleted after the person was not identified for 30 consecutive frames. When an HDMI monitor is hooked up to the device we display this information as an overlay on the current camera view, for debugging purposes, as well as drawing the count line, rectangles around detected people and their tracked vectors. You can also enable a web interface that shows the same debugging information.

4 PERFORMANCE

We used SSD MobileNet v1 because in our testing, we found it to give the best response times by far, as can be seen in Figure 4. The accuracy scores are also quite good, under the circumstances, as

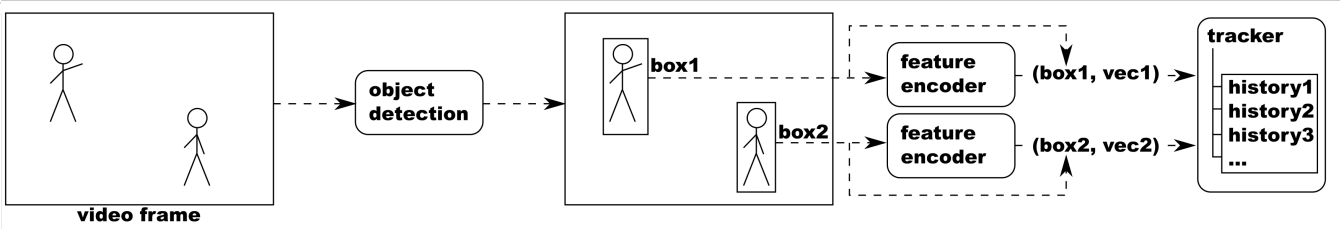

Figure 2: Tracking-by-detection pipeline.

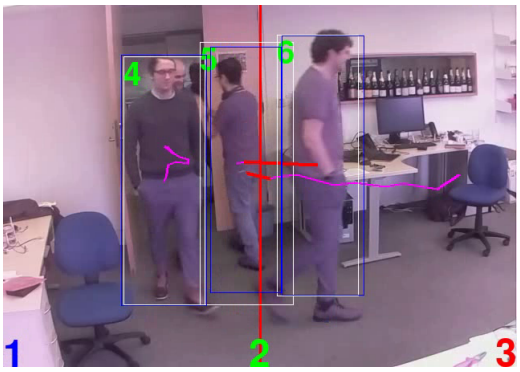

Figure 3: Screenshot from the 'Office' test video, showing two intersection events. The numbers at the bottom corners of the screen record the counts of people who have crossed the red counting line in either direction. The central number is the difference. The boxes around people are results from object-detection, and the numbers in the corners of the boxes are tracking identities. The purple lines that follow people are the tracking vectors; the intersecting segments are coloured red momentarily when an intersection event is detected.

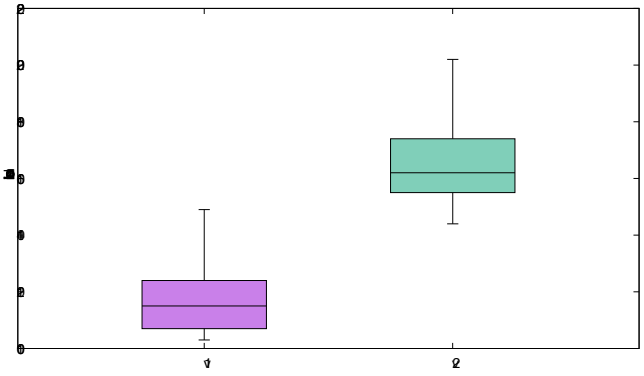

Figure 4: Inference time for object detection using different versions of SSD MobileNet.

discussed in §6. A further speed-up can be obtained by overclocking the Raspberry Pi, as seen in Figure 5. We found that the platform remained stable up to 1,900MHz, speeding up inference by about

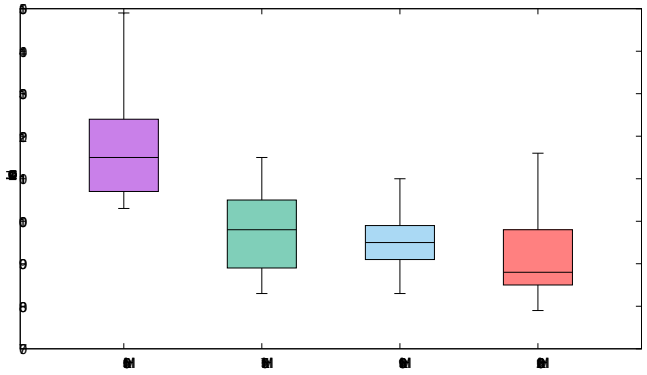

Figure 5: Object detection inference times for SSD MobileNet v1 with overclocking.

16.5%, with core temperatures fluctuating somewhat under 60°C with active cooling from the Fan SHIM. However, we learned that the PiJuice did not work reliably when the Pi was over-volted, and the HAT had to be removed before conducting these tests.

The DeepSORT feature-encoder CNN model comes trained on MARS data with an image input size of 128×64. However, we found that running it took 95ms per person per frame: too slow for real-time processing of scenes with multiple people. Therefore, we resized the image input to 64×32 and retrained the CNN model. After doing that we achieved a 35ms running time per person per frame, 63% faster than before. With overclocking the typical time for feature-encoding was reduced further to 32ms per person per frame. This is considerably more practical in our expected use case.

When overclocking to 1,900MHz, the running time can be estimated using the following numbers:

- Fixed costs per frame: approximately 130ms, composed of,
  - Object-detection: about 96ms
  - Processing (e.g. input, resizing and output): about 34ms
- Cost per person tracked: approximately 36ms, composed of,
  - Feature-encoding: about 32ms
  - Association: about 4ms

Therefore, the overall inference time per frame when tracking n people at a time averages approximately $T(n) = 130 + 36n$ milliseconds. We can track up to ten people at a time while still maintaining two FPS, five people at a time at three FPS and two people at about five FPS.

Table 1: Power draw and CPU temperature in different CPU frequencies and modes.

CPU Clock (MHz)	Mode	Power (W)	CPU Temp. (°C)
1,500	*Run*	6.0	49
1,500	*Sleep*	3.6	35
1,500	Idle Pi	3.4	33
1,750	*Run*	7.0	51
1,750	*Sleep*	3.9	35
1,750	Idle Pi	3.6	34
1,900	*Run*	8.5	57
1,900	*Sleep*	4.2	36
1,900	Idle Pi	3.7	35

4.1 Power usage

Table 1 shows the power draw and CPU temperature of the Raspberry Pi under different loads. *Run* mode is the normal operation, seeking maximum performance. In *Sleep* mode the program does not invoke inference but instead periodically checks if anything has changed in the input, waking back up if necessary. Both are compared against a baseline of an idle Raspberry Pi. There are significant opportunities for power-saving during quiet periods when there is no motion in front of the camera and it is looking at a fixed scene.

5 PRIVACY

Putting up cameras raises privacy concerns, even in public spaces. One assurance we can offer is that since our data transmission will ultimately be carried by LoRaWAN, there is simply not enough bandwidth for it to be practical to transmit images at all. The only data transmitted are counting events and the current state of the counters. An attacker could learn the number of people within the public part of the building, but this is not considered sensitive information. For debugging purposes, the WiFi device in the Raspberry Pi has been configured with a private network protected by a pre-shared key. It is not connected to the Internet and can only be accessed by a person with the password and in close proximity to the device.

As the algorithm reads and processes each camera frame, it finds the coordinates of boxes around each object it detects, then distils the contents of each box down to a short vector of 128 numbers, and finally discards the frame. The vectors may stay in memory as long as the object is within sight but they are a very sparse encoding of pixel colours and basic shapes, with no personally identifying information associated with them.

6 ACCURACY

We set up a test-bed to accept pre-recorded video in place of the live camera feed into the same algorithm used by the live tracker. The test video named 'Office' comprises two minutes of filming using the Raspberry Pi camera in an office environment similar in character to the pilot project location, with seven volunteers instructed to enter and exit the office repeatedly and without any particular patterns. The test video named 'Plaza' comprises 50 seconds of video from

Figure 6: Screenshot from the 'Plaza' test video.

the MOT17 challenge [10], taken from a fixed camera overlooking a pedestrian plaza with a fairly intense flow of people moving about, as seen in Figure 6.

The code was instrumented so that it wrote the current value of several counter variables into a file every ten seconds: persons crossing the counting line in each direction (as determined by the sign of the cross-product), overall number of crossings, and number of tracking identities that have expired.

Ground truth for test videos was established using the following method: in each ten-second interval we counted the number of people crossing the counting line in each direction.

Therefore, for each ten second interval of test video, we know the number of people walking to the 'negative' side of the line (*negcount*) and the number of people walking to the 'positive' side of the line (*poscount*). These values were subtracted and then compiled into a vector covering the whole video. This allowed us to see how the algorithm varied from the ground truth over time, and penalised offsetting mistakes to some extent.

Vectors drawn from the ground truth and from test runs of the algorithm are compared using cosine-distance, subtracted from 1. The ideal score is 1, the worst score is 0. We ran the tests with a number of configurations of the test-bed, tweaking parameters to DeepSORT, and trying much slower (but more accurate) YOLO-based object detectors in addition to MobileNet. The test-bed also was able to simulate different frame-rates, for example, by dropping five frames out of every six to simulate a 200ms inference latency.

Parameters adjustable for testing include the resolution of the MARS-trained feature encoder (64×32, 128×64 or 256×128), the simulated FPS value (from 5–30), the maximum cosine distance (*max-cos-distance*) threshold for two feature vectors to be considered part of the same 'track', and the 'non-maximum suppression' (*nms-max-overlap*) threshold that eliminates spurious overlapping object detection boxes (at 1.0 the boxes must overlap completely for one to be pruned, and at 0 it would eliminate even non-overlapping boxes). Over 450 configurations were tested. A selection of scores is shown in Table 2.

Table 2: Accuracy of counting: a selection of test results.

Score	Test name	Object-detector	Feature encoder	FPS	*max-cos-distance*	*nms-max-overlap*
0.968	Office	YOLO v3	128x64	5	0.3	0.6
0.948	Office	MobileNet v1	64x32	5	0.6	0.6
0.938	Office	MobileNet v2	64x32	30	0.9	0.6
0.906	Office	MobileNet v1	128x64	5	0.3	0.3
0.878	Office	MobileNet v2	64x32	5	0.9	1.0
0.794	Office	MobileNet v1	128x64	30	0.9	0.6
0.656	Office	MobileNet v2	256x128	30	0.6	0.6
0.424	Office	MobileNet v2	256x128	15	0.01	1.0
0.986	Plaza	YOLO v3	64x32	5	0.6	0.6
0.903	Plaza	MobileNet v1	128x64	15	0.9	0.6
0.880	Plaza	MobileNet v2	128x64	5	0.3	0.6
0.843	Plaza	MobileNet v1	64x32	5	0.6	0.8
0.839	Plaza	MobileNet v2	256x128	5	0.9	0.6
0.815	Plaza	MobileNet v2	64x32	5	0.6	0.3
0.713	Plaza	MobileNet v1	64x32	15	0.01	1.0
0.596	Plaza	MobileNet v2	128x64	30	0.9	1.0

7 DISCUSSION

The most surprising finding from our experiments is that increased frame-rate was not helpful and in fact could make things worse. As seen in Table 2, using a configuration with conditions similar to our live video counter, we achieved a score of 0.948 on the 'Office' test with MobileNet v1 running at 5 FPS. This rivalled the top test score of 0.968 that was obtained using a powerful GPU running YOLO v3 (tests at both 5 and 30 FPS achieved this mark). Figure 7 shows the overall average scores at each frame-rate: the worst cases are considerably worse with 15 and 30 FPS compared to 5 FPS.

The cosine distances were often not that large between vectors encoded from the views of different people, especially if they were wearing similar colour clothing. This could result in unwanted identity swaps, but that would only affect the score in a small way if the overall count was correct. Generally, setting *max-cos-distance* to very low values tended to reduce the performance of the tracker, effectively removing any assistance from the feature encoder. This effect can be seen in Figure 8.

Lower 'non-maximum suppression' was slightly more important with MobileNet than YOLO because the latter tends to generate higher-quality object-detection boxes and get less confused when multiple people are standing together in a group. With MobileNet, it helped to suppress some spurious boxes that could be generated by clusters of people, and lower values of *nms-max-overlap* led to slightly improved performance, as can be seen in Figure 9.

Another unexpected finding is that lowering the resolution of the input to the feature encoder in order to gain performance did not affect scores overall. In Figure 10, it shows that the 64×32 feature encoder gave approximately the same scores as the slower default resolution of 128×64. We also tried experiments with an even higher resolution feature encoder, at 256×128, and that backfired, producing worse results.

Finally we compare models in Figure 11. This chart focuses on the generally good configurations: with feature encoder resolution 64×32, processing 5 FPS, having *max-cos-distance* ≥ 0.3, and

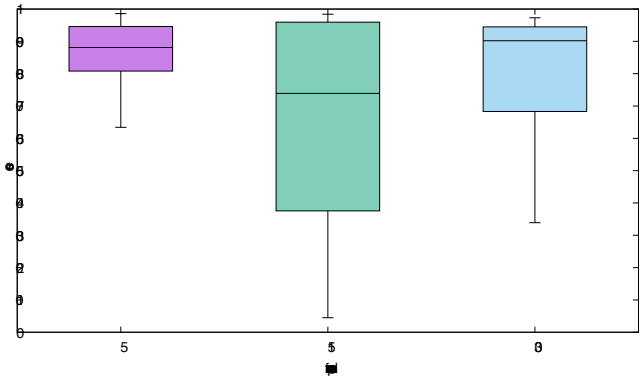

Figure 7: The effect of frame-rate on score.

nms-max-overlap < 1.0. Both MobileNet versions score about the same (just a slight advantage for v2) but in our judgement that is outweighed by the worse running time.

8 CONCLUSION

We show that practical performance can be achieved on a Raspberry Pi 4B in this application without requiring special hardware acceleration. Although our pilot project was meant to take place inside a building, we anticipate future deployments in places where network access is limited, space is at a premium and power supply may be more circumscribed. We are continuing to refine the models and parameters so this system may be used in more extensive experiments for our larger Digital Twin project.

ACKNOWLEDGMENTS

This research forms part of Centre for Digital Built Britain's work within the Construction Innovation Hub. The funding was provided through the Government's modern industrial strategy by Innovate UK, part of UK Research and Innovation.

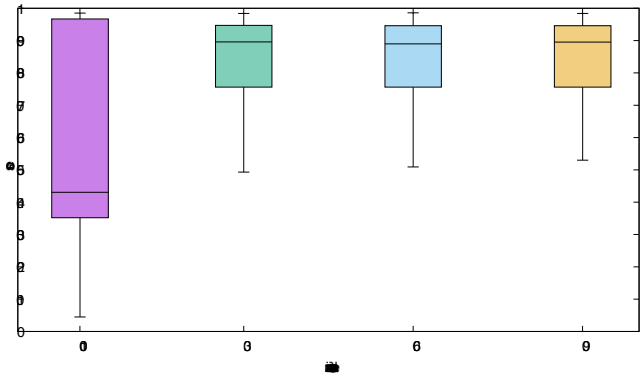

Figure 8: The effect of maximum cosine distance on score.

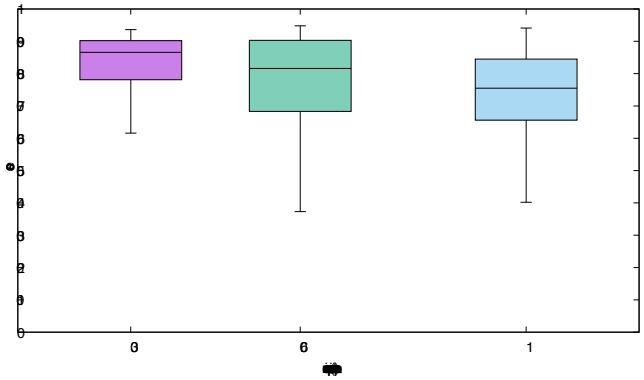

Figure 9: The effect of non-maximum suppression on score for MobileNet.

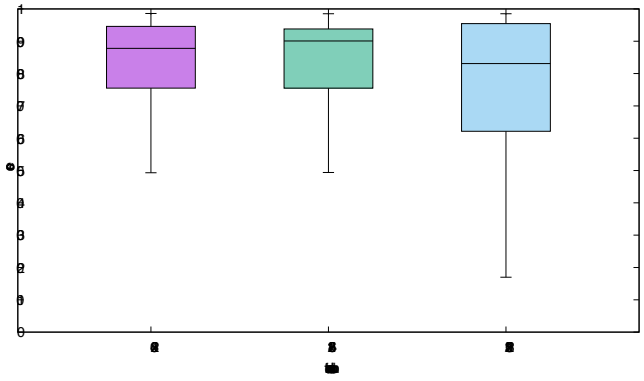

Figure 10: The effect of feature encoder resolution on score.

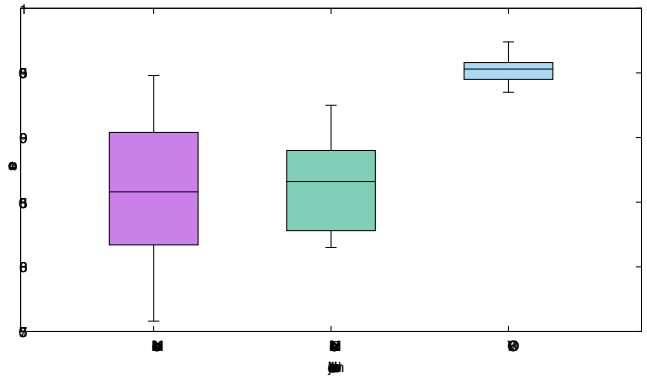

Figure 11: The effect of model type on score.

REFERENCES

[1] Ragaad Al-Tarawneh, Christina Strong, Luis Remis, Pablo Munoz, Addicam Sanjay, and Srikanth Kambhatla. 2019. Navigating the visual fog: analyzing and managing visual data from edge to cloud. In *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*. USENIX Association, Renton, WA. https://www.usenix.org/conference/hotedge19/presentation/altarawneh

[2] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. 2016. Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, 3464–3468.

[3] Alejandro Cartas, Martin Kocour, Aravindh Raman, Ilias Leontiadis, Jordi Luque, Nishanth Sastry, Jose Nuñez-Martinez, Diego Perino, and Carlos Segura. 2019. A reality check on inference at mobile networks edge. In *Proceedings of the 2nd International Workshop on Edge Systems, Analytics and Networking*. 54–59.

[4] Gioele Ciaparrone, Francisco Luque Sánchez, Siham Tabik, Luigi Troiano, Roberto Tagliaferri, and Francisco Herrera. 2019. Deep learning in video multi-object tracking: a survey. *Neurocomputing* (2019).

[5] Junwei Han, Dingwen Zhang, Gong Cheng, Nian Liu, and Dong Xu. 2018. Advanced deep-learning techniques for salient and category-specific object detection: a survey. *IEEE Signal Processing Magazine* 35, 1 (2018), 84–100.

[6] Jacob Hochstetler, Rahul Padidela, Qi Chen, Qing Yang, and Song Fu. 2018. Embedded deep learning for vehicular edge computing. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 341–343.

[7] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: efficient convolutional neural networks for mobile vision applications. *CoRR* abs/1704.04861 (2017). arXiv:1704.04861 http://arxiv.org/abs/1704.04861

[8] Kirsten Lamb. 2019. Principle-based digital twins: a scoping review. (2019).

[9] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. 2016. DeepX: A software accelerator for low-power deep learning inference on mobile devices. In *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 1–12.

[10] L. Leal-Taixé, A. Milan, I. Reid, S. Roth, and K. Schindler. 2015. MOTChallenge 2015: towards a benchmark for multi-target tracking. *arXiv:1504.01942 [cs]* (April 2015). http://arxiv.org/abs/1504.01942 arXiv: 1504.01942.

[11] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. 2014. Microsoft COCO: common objects in context. arXiv:1405.0312 [cs.CV]

[12] Peng Liu, Bozhao Qi, and Suman Banerjee. 2018. EdgeEye: an edge service framework for real-time intelligent video analytics. In *Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking*. 1–6.

[13] Joseph Redmon and Ali Farhadi. 2016. YOLO9000: Better, Faster, Stronger. *CoRR* abs/1612.08242 (2016). arXiv:1612.08242 http://arxiv.org/abs/1612.08242

[14] Ju Ren, Yundi Guo, Deyu Zhang, Qingqing Liu, and Yaoxue Zhang. 2018. Distributed and efficient object detection in edge computing: challenges and solutions. *IEEE Network* 32, 6 (2018), 137–143.

[15] Springer 2016. *MARS: a video benchmark for large-scale person re-identification*. Springer.

[16] Nicolai Wojke and Alex Bewley. 2018. Deep cosine metric learning for person re-identification. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 748–756. https://doi.org/10.1109/WACV.2018.00087

[17] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. 2017. Simple online and realtime tracking with a deep association metric. In *2017 IEEE international conference on image processing (ICIP)*. IEEE, 3645–3649.

[18] Xu Zhang, Xiangyang Hao, Songlin Liu, Junqiang Wang, Jiwei Xu, and Jun Hu. 2019. Multi-target tracking of surveillance video with differential YOLO and DeepSORT. In *Eleventh International Conference on Digital Image Processing (ICDIP 2019)*, Jenq-Neng Hwang and Xudong Jiang (Eds.), Vol. 11179. International Society for Optics and Photonics, SPIE, 701 – 710. https://doi.org/10.1117/12.2540269