DeepDish on a diet: low-latency, energy-efficient object-detection and tracking at the edge

Matthew Danish
Cambridge University
United Kingdom
mrd45@cam.ac.uk

Rohit Verma
Cambridge University
United Kingdom
rv355@cam.ac.uk

Justas Brazauskas
Cambridge University
United Kingdom
jb2328@cam.ac.uk

Ian Lewis
Cambridge University
United Kingdom
ijl20@cam.ac.uk

Richard Mortier
Cambridge University
United Kingdom
rmm1002@cam.ac.uk

ABSTRACT

Intelligent sensors using deep learning to comprehend video streams have become commonly used to track and analyse the movement of people and vehicles in public spaces. The models and hardware become more powerful at regular and frequent intervals. However, this computational marvel has come at the expense of heavy energy usage. If intelligent sensors are to become ubiquitous, such as being installed at every junction and frequently along every street in a city, then their power draw will become non-trivial, posing a severe downside to their usage. We explore Multi-Object Tracking (MOT) solutions based on our custom system that use less power while still maintaining reasonable accuracy.

CCS CONCEPTS

• **Computer systems organization → Sensor networks**; • **Computing methodologies → Computer vision**.

KEYWORDS

object detection, object tracking, edge computing

ACM Reference Format:
Matthew Danish, Rohit Verma, Justas Brazauskas, Ian Lewis, and Richard Mortier. 2022. DeepDish on a diet: low-latency, energy-efficient object-detection and tracking at the edge. In *5th International Workshop on Edge Systems, Analytics and Networking (EdgeSys '22), April 5–8, 2022, RENNES, France.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3517206.3526273

1 INTRODUCTION

Dynamic digital twins [9] and other embedded sensor networks [7] are becoming popular ways to gather data about the built environment in real-time for immediate analysis or reaction. This requires ubiquitous installation of sensors, some of which may be naturally low-power (such as temperature sensors), but others invoke computationally intensive workloads. We have been exploring a sensor design that can solve the Multi-Object Tracking (MOT) problem using video camera input, using machine learning techniques to distil the high-bandwidth image stream down to relatively few numbers that can be easily transmitted without significant bandwidth or privacy concerns.

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 approach this problem by finding bounding boxes around objects in each frame and using tracking and data association techniques to find correspondences between bounding boxes in successive frames. It is important to determine if a bounding box is being drawn around a newly introduced object, of if a previously-known object has disappeared. Altogether, this is known as the 'tracking-by-detection' approach [3].

Most approaches to solving the MOT problem are computationally and power-intensive. Feeding the video stream into software running on a high-performance workstation with a top-end GPU will produce a highly accurate result but with intensive power requirements: VoVNet [11] significantly reduces power requirements for object detection but still required 63.9W when tested on a workstation with a NVIDIA Titan X GPU. There are deployments [13] with such powerful equipment, but we believe they are using much more power than is necessary for the task, and at a large scale this raises environmental concerns. We focus on low-power solutions, ideally in the single-digit watt range.

2 RELATED WORK

Lujic, et al. [12] proposed InTraSafEd5G (Increasing Traffic Safety with Edge and 5G) as an application of object detection to help prevent crashes between turning drivers and people who are walking or cycling at junctions. They deployed a Raspberry Pi 4 with Coral EdgeTPU running the SSD MobileNet v2 model to detect people at a real-life installation with a camera pointing at a junction in Vienna. The presence of people was then transmitted using MQTT over 5G to drivers running a special mobile phone app that then would display a warning message in the driver's view. The average information delivery latency was about 108ms in their experiment.

Fernández-Sanjuro, et al. [6] developed a real-time multi-object tracking-by-detection system on the NVIDIA Jetson TX2 embedded

EdgeSys '22, April 5–8, 2022, RENNES, France

ACM ISBN 978-1-4503-9253-2/22/04…$15.00
https://doi.org/10.1145/3517206.3526273

computing board with Pascal GPU. The object detector is based on customised YOLOv3 architecture and the tracker is a combination of an appearance-based correlation filter tracker (KCF) and a motion-based Kalman Filter tracker. To reach real-time speeds the object detector is only used on every eleventh frame. The Jetson is a more powerful computer than the Raspberry Pi and consumes approximately 15W under load, about twice as much as the Pi. SkyNet [19] is also a Jetson-based project with an object detection model developed from the bottom-up to capture hardware limitations.

The previous iteration of DeepDish [4] explored object detection and tracking on the Raspberry Pi. It demonstrated feasible CPU-only tracking-by-detection and showed power usage could be kept to within 7W when not overclocking the CPU. One surprising result was that low framerate did not necessarily affect the accuracy of the results — in terms of a people-counting metric. In some cases the result was even improved by lower framerates. DeepDish is the basis of the current work, however the detection and tracking engine has been completely rewritten for high-performance and flexibility on various platforms, and works with either GPUs or Coral EdgeTPUs when available.

MARLIN [2] is an object detection and tracking system for augmented reality with an explicit goal of reducing energy drain on mobile phones (it was tested on the LG G6 and Google Pixel 2). They used the Tiny YOLO object detector but focused on finding ways to avoid invoking it because the detection latency is 1,200ms and it consumes an additional 1.7-1.9W when running. Object tracking is maintained using optical flow and they trained and evaluated a 'change detector' based on random forest classifiers to determine if the object detector needed to be run again. They compared this against methods that simply skipped frames and a baseline method that ran detection on every frame. They found that MARLIN reduces power consumption by 34% on average with a typical accuracy loss of less than 10%.

3 EXPERIMENTAL SETUP

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 have connected a Google Coral EdgeTPU device to one of the USB 3.0 ports to provide hardware acceleration for specially compiled models. EdgeTPUs work well for convolutional neural networks and other simple feed-forward networks.

3.2 Software

We use the TensorFlow-Lite (TFLite) engine for evaluation on the Raspberry Pi, and the controlling software is written in Python 3.7 running under Hypriot, a Debian-based operating system customised for Raspberry Pi and specialised to run Docker. Our software runs within a Docker container based on the BalenaLib framework, with the necessary libraries and modules all ready to go.

3.3 Test harness

In order to simulate the capture of live video, we have written the program with a queue feeder thread to read video files at a constant framerate (such as 25 FPS) and offer the video frame to the main thread for a limited time period until the next one is ready. If the input queue of the main thread is not ready to accept the video frame when the time elapses, then that video frame is dropped and the feeder moves onto the next one. This allows us to simulate the effects of live video while being able to make comparisons against existing well-known video sets with ground-truth data and metric-analysis scripts, such as those from MOT15 [10] including its multi-object tracking accuracy (MOTA) metric.

Power consumption is measured by a Tasmota smart plug. Whenever the power usage changes by more than 0.1W, this sensor transmits the new numbers using MQTT over Wi-Fi to our experiment-monitoring workstation. DeepDish also transmits its telemetry using MQTT to the same computer, so its data can be correlated with the power readings, and average power usage measured over the running time of a test.

4 ALGORITHM

We follow the basic architectural concept of DeepSORT [18], as shown in Figure 1, but with substantial modifications to object detection, feature encoding and tracking/association.

4.1 Object detection

We use a pluggable object detection architecture designed to support numerous models and new ones to come. At the time of this writing, here are some notable examples we support that generally fall into the following categories:

- TFLite models, and their quantised EdgeTPU versions:
  - EfficientDet Lite [1]
  - SSD MobileNet v1 and v2 [8]
- YOLOv5 [16] family of models, converted into TFLite and EdgeTPU-supporting formats:
  - In particular, the smaller and faster architectures, namely: YOLOv5n, YOLOv5s and their recent updates, YOLOv5n6 and YOLOv5s6
- Other models that fit in the TensorFlow object detection family, but are less suitable for edge computing:
  - EfficientDet family of models (D0…D7) [15]
  - CenterNet ResNet101 and CenterNet HG104 [5]

The output of the object detector is mapped into a common interface composed of bounding boxes, labels and confidence scores for each. If the background-subtraction option is enabled then we prune any bounding boxes that do not contain regions of motion. If there are no objects detected in the scene, and powersaving is enabled, then we begin an artificial slowing-down process that intentionally bottlenecks the pipeline so that the object detector runs less frequently while there are no objects in the scene. Once any object is detected, the pipeline returns to full speed.

4.2 Feature encoding and tracking

The DeepSORT-based feature-extraction step uses the DeepFLOW CNN model trained on the Motion Analysis and Re-identification Set (MARS) data-set [14] using the cosine metric learning technique [17]. Compared to the original work, we use smaller input sizes of 16x32 and 32x64 to reduce latency. In order to speed up processing times even further, we have also retrained a modified

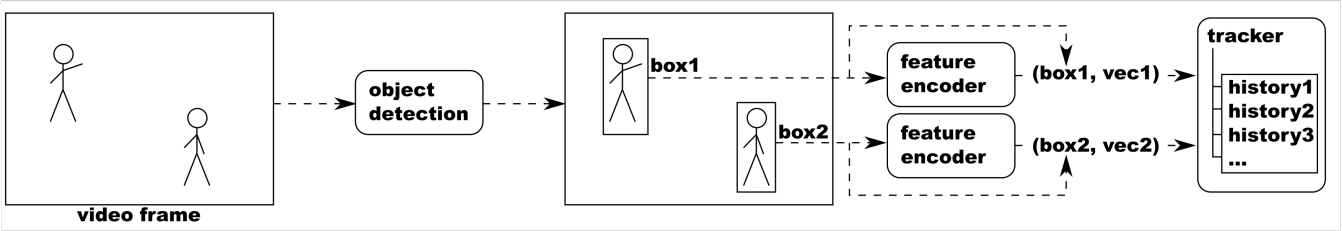

Figure 1: Tracking-by-detection pipeline.

Feature Encoder	MOTA
constant	40.8%
8x16-mod	41.4%
16x32	41.4%
32x64	41.2%

Table 1: Comparing the accuracy performance of four feature encoders, with object detections in all cases provided by the high-quality EfficientDet D7 model (see Section 4.2).

Model	**e2e**	**f2f**	**obj**	**enc**	**cpu**	**fif**
EfficientDet-Lite0	459	175	162	78	231%	2.2
SSDMobileNetV1	409	104	65	99	296%	3.9
SSDMobileNetV2	384	99	64	92	296%	3.8
YOLOv5n6-320x320	537	188	175	104	227%	2.4
YOLOv5s6-320x320	455	194	183	50	183%	2.0

Table 2: Latencies (in ms) for various pipeline components, comparing various object detector models (see Section 4.3).

version of this model ('8x16-mod') with several layers removed, on a smaller input size of 8x16 pixels, and found it to run much faster with basically no loss of tracking accuracy. See Table 1 for a comparison of encoders and MOTA on the 'TUD-Stadtmitte' example from MOT15, with object-detections in this case provided by the high-quality EfficientDet D7 model running on an NVIDIA GeForce RTX 2080Ti GPU.

In performing these tests, we converted the encoder model to the TFLite format and modified it to accept a fixed batch-size so that the deployed device need not run the full TensorFlow stack, but only the much smaller tflite-runtime package, while still efficiently processing batches (TFLite cannot handle variable input sizes in the same manner as the full-blown TensorFlow). As a control measure we also implemented a 'constant' image-encoder that always returns the same unit-vector for any input, which shows that a significant amount of the value of the tracking module comes from the data association portion discussed below.

Although running in batches reduces fixed overhead, feature extraction must be run separately on each person detected, therefore this portion of 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 as well as the cosine metric distance computed on extracted feature vectors as discussed above. Objects that are new to the tracking history are assigned a fresh identification number, and objects that fail to be found for over 60 frames are considered to have left the scene.

4.3 Pipeline

One of the goals with the complete rewrite of DeepDish was to architect it in a scalable way. This has been achieved by structuring the pipeline as a series of asynchronous Python tasks connected by queues, based on the asyncio module. The frame capture loop, object detector and feature encoder all run in separate threads in order to ensure that they do not block the cooperatively-scheduled asynchronous pipeline. The result has been a success, with the pipeline capable of processing multiple 'frames-in-flight' at a time. We instrumented the code to measure performance in several ways; all of the below are averages over the course of a single run on the MOT15 'TUD-Stadtmitte' sample with the object detector running on every frame:

- **e2e** (ms) The time it takes from when a frame enters the pipeline to when the pipeline finishes with it.
- **f2f** (ms) The time between invocations of the same pipeline stage by consecutive frames, a.k.a. 1/framerate.
- **obj** (ms) The latency of the object detector per frame.
- **enc** (ms) The latency of the feature encoder per frame.
- **cpu** Average CPU utilisation.
- **fif** Average number of frames-in-flight in the pipeline.

Table 2 shows instrumentation output for several sample runs. Note that the **f2f** timings are significantly smaller than the **e2e**; this means frames are hitting the end of the pipeline much more quickly than it takes for a single frame to be processed. Some of the **e2e** latency comes from waiting in queues, however both **obj** and **enc** measure the latency of some of the significant computations that take place in the pipeline and their combination is always longer than the **f2f** time; this indicates that the pipeline is at the very least performing object detection and feature encoding in parallel on consecutive frames.

The previous generation DeepDish had frame-to-frame performance (on an overclocked Raspberry Pi) modelled as $T(n) = 130 + 36n$ milliseconds where n is the number of people detected in the scene. All of the runs shown in Table 2 have comparable performance to $T(2)$, some significantly better than $T(1)$, on a recorded scene with 4-5 people on average. The differences are: use of the EdgeTPU, pipelining to take advantage of the multiple CPU cores, and a faster and lighter-weight feature-encoder.

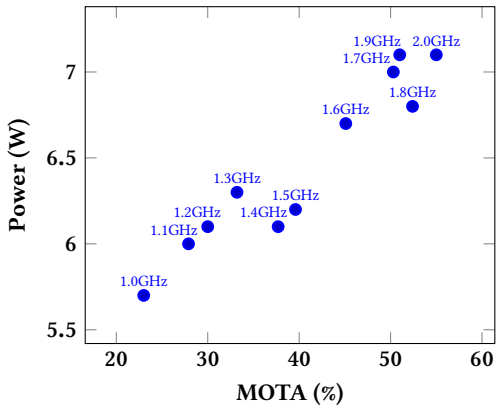

Figure 2: Power and accuracy at different CPU clock-speeds (discussed in Sections 5.1 and 6.3).

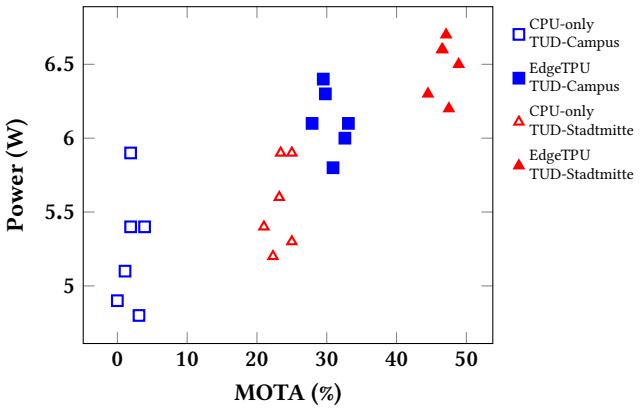

Figure 3: Two sample videos from MOT15 are tested for power usage and accuracy, first running on CPU, and then boosted with the EdgeTPU (see Section 5.2).

5 REDUCING POWER USAGE

5.1 Underclocking

We experimented with underclocking the Raspberry Pi 4B CPU from the normal 1.5GHz to see how it would affect power consumption and performance. We tested different CPU clock-speeds under the object detector model SSDMobileNetV1 and the constant feature encoder, skipping two out of every three frames. The MOT15 video 'TUD-Stadtmitte' was fed into the test harness at a steady rate of 25 FPS, whether or not the system could keep up. The scatter plot in Figure 2 shows the average MOTA and power readings from six runs at eleven different CPU frequency settings, of which the 1.5GHz is the factory setting and those below it constitute underclocking.

It should be noted there is significant variability as frame-dropping affected the results. In general, though, we can see that underclocking causes an immediate reduction in accuracy without necessarily reducing power usage, at least not until the clock-speed is drastically lowered, severely affecting accuracy.

5.2 EdgeTPU effect

We experimented with holding all settings constant except for swapping between the EdgeTPU and CPU-only versions of SSDMobileNetV1. These experiments were conducted on the 'TUD-Campus' and 'TUD-Stadtmitte' videos from MOT15. Unsurprisingly, connecting and using the Coral EdgeTPU increases power consumption of the whole device, on average from 5.4W to 6.3W in these tests. However, the acceleration of the EdgeTPU also greatly increased the MOTA scores, as can be seen in Figure 3.

5.3 Choice of model

We looked at whether the object detector model affected the power usage of the Raspberry Pi. We tested six models, all of which use the EdgeTPU to some extent, running them six times each on five different videos (the ones listed in Table 3). The results are shown in Figure 4. The corresponding latencies are shown in Figure 5. Power consumption was effectively the same for all the models tested. Latencies varied substantially, with the MobileNet models winning handily; see Section 6.2 for further discussion.

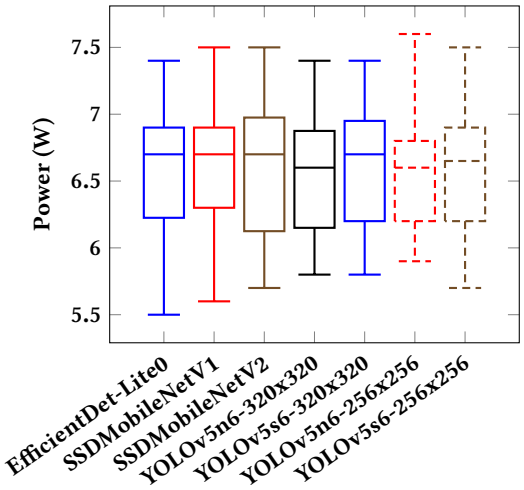

Figure 4: Comparison of power consumption by different object detector models (see Section 5.3).

6 ACCURACY

6.1 Comparison to GPU

The DeepDish software is also able to run on workstations with GPUs. We collected accuracy results of such high-power runs for comparison purposes. With a GPU available, we chose to use the high-quality EfficientDet D7 object detection model and to run it alongside a 32x64 feature encoder generated from the original DeepSORT neural architecture. These tests were run with no time constraints on an Intel Core i9-based workstation equipped with one NVIDIA GeForce RTX 2080Ti GPU using videos from MOT15 to produce the MOTA scores in Table 3. We selected these particular videos, from the available benchmark set, because these ones largely fit the target use-case of DeepDish, which is motion-tracking people or vehicles in urban or interior scenes.

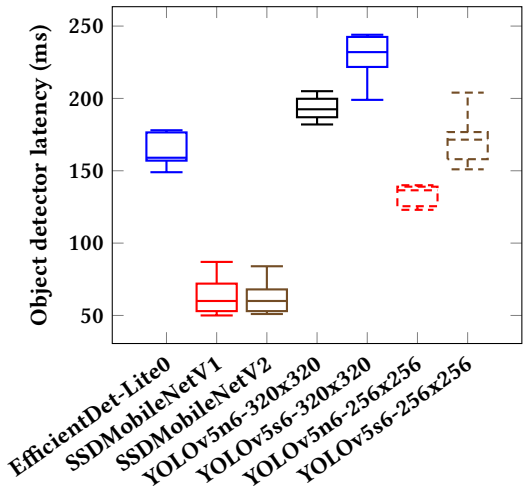

Figure 5: Comparison of latency of different object detector models (see Section 5.3).

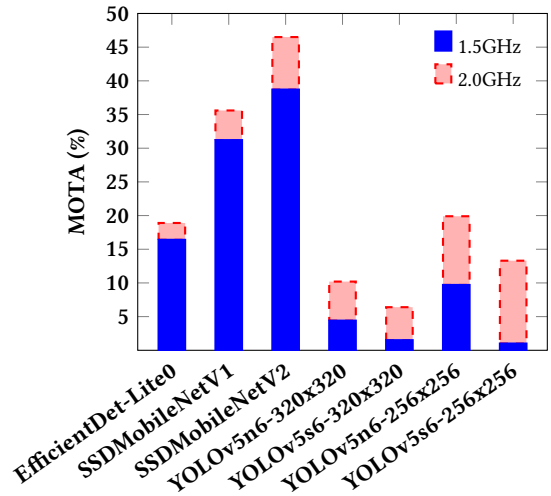

Figure 6: Video 'TUD-Campus': accuracy vs choice of model (see Section 6.2).

Video	MOTA
PETS09-S2L1	78.7%
TUD-Campus	63.2%
ETH-Pedcross2	50.6%
ADL-Rundle-6	55.6%
TUD-Stadtmitte	41.2%

Table 3: Accuracy of DeepDish running on a high-powered workstation with GPU (see Section 6.1).

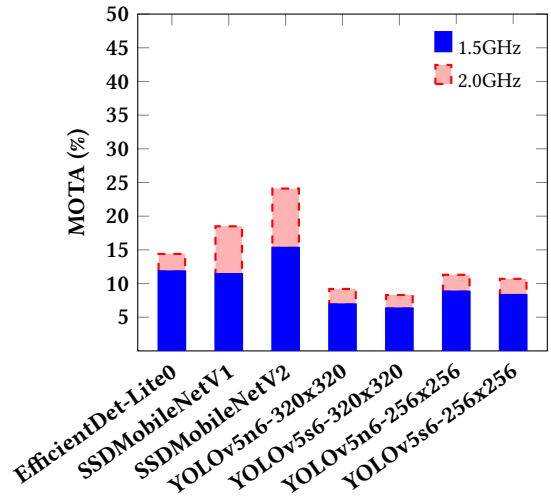

Figure 7: Video 'ETH-Pedcross2': accuracy vs choice of model (see Section 6.2).

6.2 Accuracy and choice of model on the Pi

We examined accuracy on a model-by-model basis more closely for two of the MOT15 videos, 'TUD-Campus' (Figure 6) and 'ETH-Pedcross2' (Figure 7), first without overclocking and then at 2.0GHz and higher voltage. Both videos have a fair amount of pedestrian activity, the latter being a longer and more complex video with a moving camera. We ran them on the test harness at 25 FPS, dropping frames whenever the pipeline was no longer able to accept them in time. We note that the performance of the SSDMobileNet family continues to hold up, with its low latency (Figure 5) seeming to make a big difference in outcome. EfficientDet-Lite0 comes in third, dragged down by its higher running time, meaning that more frames must be dropped. Surprisingly, the YOLOv5 family does poorly, even using the newer 'n6' and 's6' variants; it only manages to achieve somewhat better latencies when using the 256x256 input size, but then their MOTA scores come out worse than the ones from EfficientDet-Lite0 (with its 320x320 input).

6.3 Overclocking

While overclocking goes against the grain of this work, because it increases power consumption, we looked at it to see whether the accuracy / power trade-off might be worthwhile sometimes. Figure 2 shows MOTA and power consumption at a range of frequencies, all the way up to 2.0GHz, in the manner described in Section 5.1. The chart shows that overclocking does increase power consumption sigificantly, but also accuracy. This is most likely because fewer frames are dropped. In this test it was possible to keep average power usage under 7W even when overclocked to 1.8GHz, while gaining most of the accuracy advantage of higher speeds.

7 CONCLUSION

The previous generation DeepDish relied on overclocking at least to 1.9GHz in order to bring the frame processing latency to within tolerable margins. That setup consumed 8.5W under load and brought the CPU temperature to 57C, ultimately requiring active cooling and ventilation.

In comparison, the complete redesign of the pipeline and utilisation of the EdgeTPU has resulted in a lower average power consumption with quicker frame-processing latency. The MOTA metric is affected by lower framerates in much different way that the aforementioned people-counting metric was in the older work (see Section 2 discussion on [4]). However, these lower scores can be ameliorated to some extent through overclocking at the cost of additional power consumption. In practice, the parameters and choice of model may require some tuning for each particular scene that DeepDish is monitoring.

We have shown results for a viable object detector and tracking system able to run on hardware rated for wattage in the single-digits. Furthermore, we should be able to take advantage of faster and more accurate EdgeTPU-based object detection models as they are developed. In the future, our work could be improved by potential advancements such as a combined object-detecting and feature-encoding model, or the addition of support for LSTM object detection models to the EdgeTPU.

7.1 Source code

DeepDish is open-source and may be found on the web at: https://github.com/AdaptiveCity/deepdish.

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. This work is funded in part by EPSRC grant EP/T022493/1.

REFERENCES

[1] 2020. EfficientNet-Lite. https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet/lite

[2] Kittipat Apicharttrisorn, Xukan Ran, Jiasi Chen, Srikanth V Krishnamurthy, and Amit K Roy-Chowdhury. 2019. Frugal following: power thrifty object detection and tracking for mobile augmented reality. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*. 96–109.

[3] 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).

[4] Matthew Danish, Justas Brazauskas, Rob Bricheno, Ian Lewis, and Richard Mortier. 2020. DeepDish: multi-object tracking with an off-the-shelf Raspberry Pi. In *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*. 37–42.

[5] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. 2019. Centernet: Keypoint triplets for object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*. 6569–6578.

[6] Mauro Fernández-Sanjurjo, Manuel Mucientes, and Victor Manuel Brea. 2021. Real-time multiple object visual tracking for embedded GPU systems. *IEEE Internet of Things Journal* 8, 11 (2021), 9177–9188.

[7] John Heidemann and Ramesh Govindan. 2004. An overview of embedded sensor networks. *Handbook of Networked and Embedded Control Systems* (2004), 1–20.

[8] 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

[9] Kirsten Lamb. 2019. Principle-based digital twins. (2019).

[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). arXiv: 1504.01942.

[11] Youngwan Lee, Joong-won Hwang, Sangrok Lee, Yuseok Bae, and Jongyoul Park. 2019. An Energy and GPU-Computation Efficient Backbone Network for Real-Time Object Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.

[12] Ivan Lujic, Vincenzo De Maio, Klaus Pollhammer, Ivan Bodrozic, Josip Lasic, and Ivona Brandic. 2021. Increasing traffic safety with real-time edge analytics and 5G. In *Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking*. 19–24.

[13] Keely Portway. 2021. Smarter cities in sight: Keely Portway finds out how imaging is helping keep cyclists and pedestrians safe. *Imaging and Machine Vision Europe* 103 (2021), 16–20.

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

[15] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*. PMLR, 6105–6114.

[16] Ultralytics. 2021. YOLOv5. https://github.com/ultralytics/yolov5

[17] 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

[18] 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.

[19] Xiaofan Zhang, Haoming Lu, Cong Hao, Jiachen Li, Bowen Cheng, Yuhong Li, Kyle Rupnow, Jinjun Xiong, Thomas Huang, Honghui Shi, et al. 2020. SkyNet: a hardware-efficient method for object detection and tracking on embedded systems. *Proceedings of Machine Learning and Systems* 2 (2020), 216–229.