A Fast Video Image Detection using TensorFlow Mobile Networks for Racing Cars

Selahattin Akkas*, Sahaj Singh Maini*, Judy Qiu*
Luddy School of Informatics, Computing and Engineering
Indiana University
Bloomington, IN, USA
{sakkas, sahmaini, xqiu}@indiana.edu

***Abstract*—With the growth of the Internet of Things, we see an increase in the importance of analysis of data from the edge, often with the results needed in real-time. Indy Car series is one of the well-known racing series in North America. All cars are equipped with multiple cameras. The video streams captured by these cameras can be used for detection and predictive tasks to increase race safety and develop better strategies to win the race. Moreover, the data can be used together with the telemetry data to provide better analysis and predictions for the drivers and the teams. In a lot of video analytics tasks, the tasks begin with object detection as its foundation. The existing pre-trained object detection models are inadequate to detect IndyCar race cars. Therefore, we have created a new dataset and have compared three different Single Shot Multibox Detector models from TensorFlow Detection Model Zoo. We run experiments on CPU and GPU. Since transferring the data from edge devices to a server, running inference, and sending the result back is time and resource consuming, we also test mobile detection models on an Edge TPU, which is a Google Coral Dev Board. Our initial results show that the Edge TPU gives the best inference time, and it is more suitable for a real-time machine learning task.**

***Index Terms*—Race car detection, IndyCar series, Real-time object detection, Edge TPU**

I. Introduction

The IndyCar Series, or the NTT IndyCar Series, is North America's top-level open-wheel racing [1]. Cars have twin-turbocharged 2.2-liter direct-injected V6 engines, which run at 12000 RPM. Engines are developed by Honda and Chevrolet, and estimated horsepower varies between 500 and 700 depending on the turbo settings [5].

Anomalies are detected in [39] by using the telemetry data collected by more than 150 sensors placed on each car. Other than the telemetry data, each track, as well as each car, have multiple cameras. The visual data collected from these cameras can be used in detection and prediction to help drivers, teams, and race organizers. For instance, anomalies can be detected, accidents can be prevented by warning the drivers or the detections can provide better analysis for the following races. However, the prediction tasks require to detect the cars first, as we can see in [19], [43]. On the other hand, the models trained on well-known datasets do not perform well on the IndyCar race cars, as can be seen in Fig. 1. Therefore, a specialized labeled dataset is needed for IndyCar race cars.

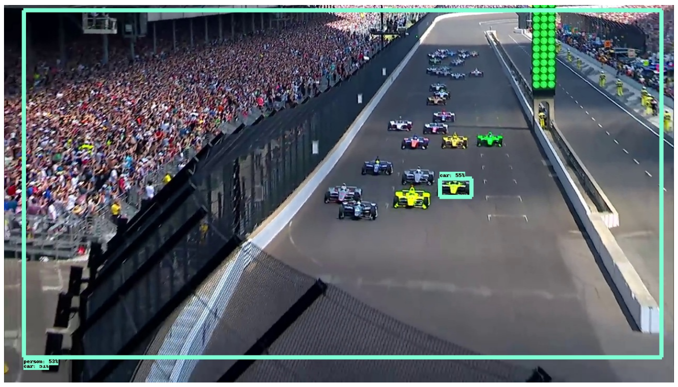


Fig. 1: Indycar detection results with pre-trained SSD Resnet-50 FPN on COCO dataset [2], [24]

The detection tasks for IndyCar can be highly time-sensitive similar to self-driving cars. However, IndyCar race cars are significantly faster and can go up to 240 miles per hour. In a high-speed environment, detection and prediction times become notably more crucial. To reduce the detection and prediction time and to be able to meet the real-time constraints, suitable models and infrastructure are required. The average data arrival time from the car to a server is 80-90 ms for the telemetry data [39]. Currently, we are unable to measure the arrival time for video streaming; however, the arrival time will be much slower in video streaming compared to the telemetry data. We aim to meet the inference rate of 30 images per second, which is equal to the frames per second (FPS) of the video files used for the construction of the dataset.

The contributions of this paper include: 1) to create new labeled image dataset to detect race cars accurately, 2) to re-train three widely used Single Shot Multibox Detector [25] models (which are SSD MobileNetV1 Quantized COCO, SSD MobileNetV2 Quantized COCO, and SSD ResNet-50 FPN COCO) on the new dataset, and 3) to benchmark the performance of the models on CPU, GPU, and Edge TPU respectively. We further include interaction with the MLPerf [6] community, where we will contribute our measured performance results and a new dataset based on this work.

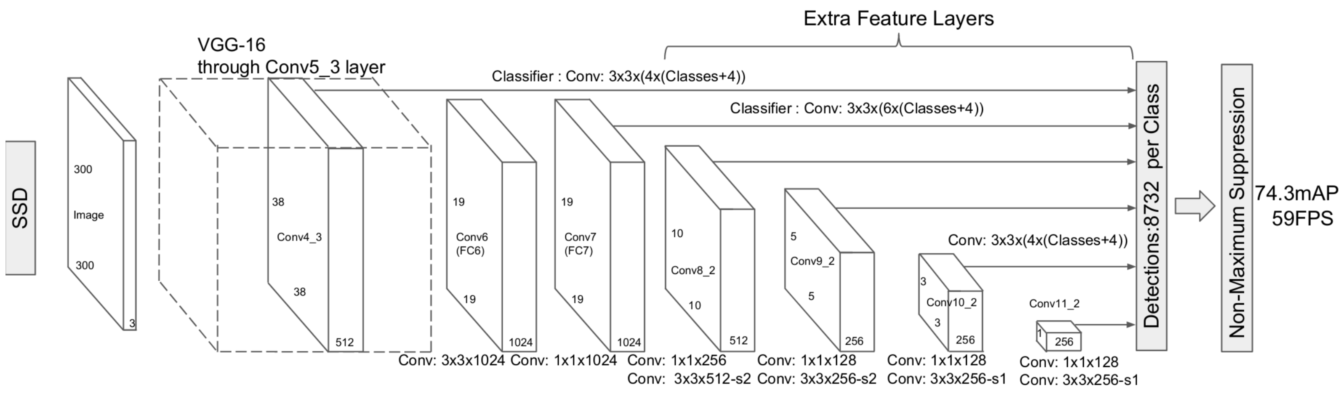


Fig. 2: Single Shot Multibox Detector (SSD) architecture [25]

II. Motivation and Background

Due to the development of self-driving car application systems as well as IoT in the past years, there has been a lot of attention being given towards offloading from Cloud to Edge. According to [33], the previous task assignment works can be split into two categories based on their objectives: (1) Energy-aware or latency-aware task assignment [9], [13], [21], [26], [28], [32], [38], and (2) Bandwidth-aware [8], [14], [27], [37], [41]. There is a notable amount of computation that is involved in the calculation of the task assignment schedule that makes the energy-aware or latency aware task assignment approaches that try to improve energy efficiency or reduce task latency inappropriate for real-time machine learning applications with energy efficiency requirements; however, none of the preceding approaches for bandwidth-aware task assignment take into consideration the limited energy of the edge devices and deadline-awareness of task [33].

In order to efficiently run machine learning models on edge devices in terms of execution time and memory, researchers have introduced techniques for optimization of the neural networks in the machine learning models [16], [22], [42]. For the cloud servers, many job schedulers for ML jobs [15], [20], [29], [40] have been proposed. Some are for CPU-based clusters [29], [44] while others are for GPU-based clusters [15], [40]. Our proposed work can evaluate CPU and GPU servers, and Edge TPU based approaches to clarify the ML real-time performance further.

[12] detects cones in Formula Student Driver-less competition. They modify the Tiny YOLO for ASIC and FPGA, then they compare GPU, ASIC, and FPGA. While GPU that is placed in the car used in [12] consumes 75W, ASIC consumes ~1W and FPGA consumes ~12W. On the other hand, Google Coral Dev Board consumes 4W while it is idle, and 8.5W while operating at high-performance [11]. They obtain their data from a camera fixed on the car, and they crop the image during inference to remove the portion of the image majorly containing the sky. However, this approach does not apply to our work as the video stream contains data from multiple different cameras both on and off the car, and we do not use one fixed camera to collect the data.

[23] provides a self-driving car simulator to improve the acceptance of self-driving cars. They detect the cars in front of the car and calculate the distance, then they estimate the danger based on the distance. They use YOLOv3 [30] to detect the objects.

III. Methodology

Video/image object detection using streaming data is an extension to traditional approaches of using neural networks for object detection, which is well suited for identifying objects in a high-speed racing environment. This work is to answer the following research question:

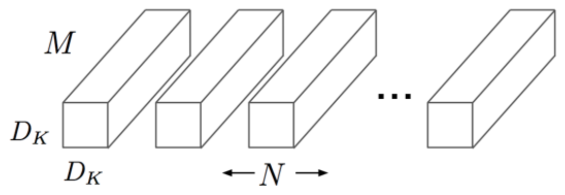
How does the choice of neural network models for object detection and hardware (servers and edge devices) impact accuracy and latency?

A. Object Detection Models

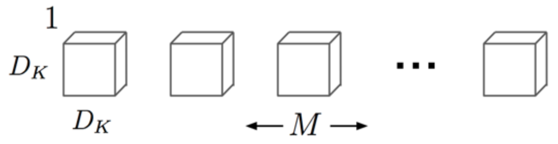
Single Shot Multibox Detector (SSD) [25] generates a certain number of bounding boxes and confidence scores using a feed-forward CNN. It then applies non-maximum suppression to reduce the number of boxes for the prediction. The architecture of the SSD model uses VGG-16 [34] without a fully connected network as its base network traditionally; however, multiple variations of SSD exist. The feature layers are added following the base network to decrease the size of the input to the successive layer and give predictions at different scales. Fig. 2 shows the architecture of the SSD.

Although SSD with a large architecture gives better detection results, it is not applicable to edge devices due to its large memory footprint and computational complexity. Therefore, networks with fewer layers and simpler operations are needed for edge devices. We use three different SSD based networks in which two of them are SSD mobile networks that are compatible with Edge TPU, and one is an SSD Resnet-50 FPN to compare the SSD mobile networks with a full-sized network.

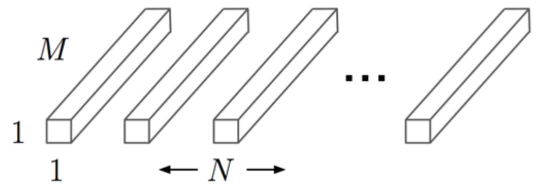
1) MobileNetV1 [18]: Depthwise separable convolution is used instead of a standard convolution in MobileNetV1. In a depthwise separable convolution, different filters are used for each channel. Then, a 1 x 1 point based convolution is applied to merge the outputs of the separable convolution. When 3x3



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Fig. 3: Depthwise separable convolution and 1 x 1 point based convolution [18]

kernels are used, the total computation cost drops by 8 or 9 times compared to the standard convolution with an acceptable reduction of the accuracy [18]. Fig. 3 shows the filter shapes for standard, depthwise, and pointwise convolution.

2) MobileNetV2 [31]: It is an improved version of MobileNetV1 where the performance has been improved by introduction of linear bottlenecks in between layers and shortcut connections between respective bottlenecks. Further improvements can be seen in [31]. These changes reduce the number of parameters and increase the accuracy compared to the MobileNetV1.

3) Resnet-50 [17]: Resnet-50 is a neural network model that contains 50 layers with skip connections in its architecture. We include this model to compare the object detection accuracy with MobileNets.

Table I shows a total number of parameters and total multiply-add counts (MACs) for the above mentioned neural network models. These numbers do not include SSD feature layers. We can see that Resnet-50 is quite large compared to mobile nets.

TABLE I: TOTAL MULTIPLY ADD COUNTS (MACs) AND PARAMETERS FOR A 224 x 224 INPUT IMAGE [3], [4], [35]

Network	MACs (millions)	Parameters (millions)
MobileNetV1	569	4.24
MobileNetV2	300	3.47
Resnet-50	3900	25.5

IV. Data Collection

Data preparation is an essential part of this work. The data life cycle has four different stages, which are - data collection, data cleaning, data storage, and the data processing stage. In this paper, we use video streams from multiple Indianapolis 500 race events to extract and prepare the data required to train the deep neural network models to detect the relevant information needed. The video streams used in this paper to prepare the data contain different camera angles. The video streams contain multiple frames from different positions on the track-side looking over the race cars, and they also contain multiple frames from cameras positioned on each car that gives the audience the driver's view of the race at different times. We extract the frames using OpenCV [7]. The extracted images from the video stream are manually annotated using rectangular bounding boxes around the race cars via LabelImg [36] labeling tool, as can be seen in Fig. 4 to be used for training and evaluating the object detection models for the required tasks. We label 3096 images using the LabelImg labeling tool, which are split into a training set that contains 2632 images with a total of 10637 labeled race cars and test set, which contain 464 images with a total of 1821 labeled race-cars. The LabelImg tool can be used to produce two different formats of annotations that are PASCAL-VOC format and YOLO format. In both formats, there exists a corresponding annotation file for each image file that contains object class as well as bounding box information for all objects in the image file.

V. Experimental Setup

A. Hardware

We use a node that has 2x12-core Intel(R) Xeon(R) E5-2670 v3 (Haswell) processors with 128 GB main memory; and Nvidia Tesla K80 GPU with 12GB of memory. We also use an Edge TPU device, which is a Google Coral Dev Board with quad Cortex-A53, Cortex-M4F CPU, Integrated GC7000 Lite Graphics, Google Edge TPU coprocessor as an ML accelerator, and 1 GB LPDDR4 main memory. While the server node runs Red Hat Enterprise Linux Server 7.7 (Maipo), the Edge TPU device runs Mendel GNU/Linux 3 (Chef).

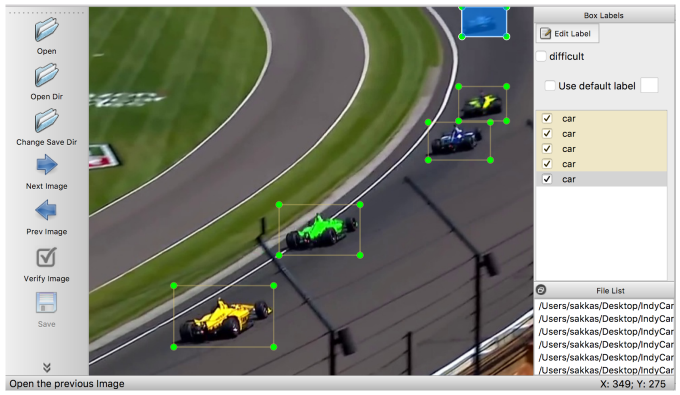

Open
Open Dir
Change Save Dir
Next Image
Prev Image
Verify Image
Save
Box Labels
Edit Label
difficult
Use default label
car
car
car
car
car
File List
/Users/sakkas/Desktop/IndyCar
/Users/sakkas/Desktop/IndyCar
/Users/sakkas/Desktop/IndyCar
/Users/sakkas/Desktop/IndyCar
/Users/sakkas/Desktop/IndyCar
/Users/sakkas/Desktop/IndyCar
Open the previous Image
X: 349; Y: 275

Fig. 4: Image labeling

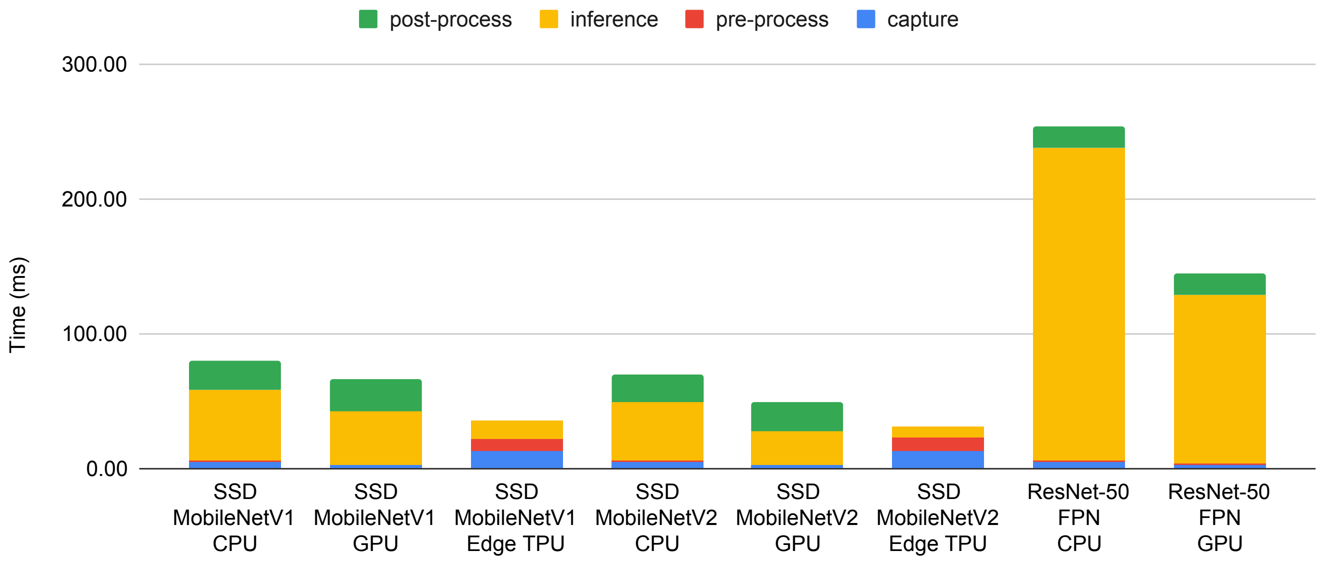
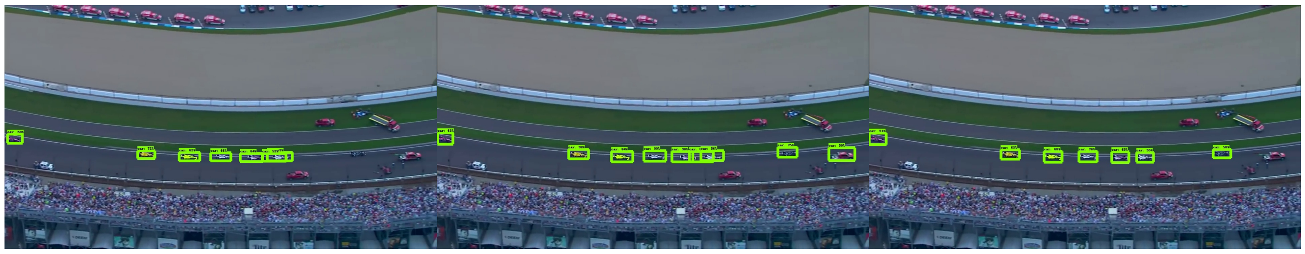


Fig. 5: Inference Time (ms) breakdown on CPU, GPU and Edge TPU



(a) SSD MobileNetV1 Quantized (b) SSD MobileNetV2 Quantized (c) SSD Resnet-50 FPN

Fig. 6: Detection results after re-training on our dataset

B. *Software*

We use Cuda 10, Cudnn 7.6.2, TensorFlow 1.14.0 for the GPU experiments, and TensorFlow MKL 1.14.0 which is optimized for Intel CPU's for the CPU experiments. The Edge TPU only supports TensorFlow Lite; therefore, we use TensorFlow Lite 1.14.0 for the Edge TPU experiments.

VI. Video/Image Detection Results and Analysis

We use SSD MobileNetV1 Quantized, SSD MobileNetV2 Quantized, and SSD ResNet-50 FPN models from the TensorFlow Detection Model Zoo [2] that are pre-trained with the COCO dataset [24]. We choose the quantized versions of SSD MobileNetV1 and V2 since the Edge TPU requires a quantized model.

Training and test data are converted to TFRecord format where the data is stored in binary record sequences since the models from TensorFlow Model Detection Zoo require data in TFRecord format. The default input size is 300x300 for the mobile nets, it is 640x640 for the ResNet-50 FPN model. The models are re-trained for 25000 steps with the mini-batch size of 24 for the mobile nets, and 8 for the ResNet-50 FPN because 24 images of size 640x640 do not fit into the GPU memory. We run the inference on a video file by extracting images in real-time and document the average time. In the process, ten thousand images were extracted from the video used for testing the models on CPU, GPU, and TPU.

Table II shows that mobile nets give similar COCO mAP [10] values which are 29.84 for the SSD MobileNetV1 Quantized and 29.48 for the SSD MobileNetV2 Quantized, while SSD Resnet-50 FPN gives 46.4. Also, SSD Resnet-50 FPN is much better at detecting medium and small objects due to the input resolution and more layers. Detecting small objects can be a challenging task. We can see in Fig. 6 that the model with the Resnet-50 base network is more accurate than the rest of the models. From the results (see Table II and Table III), we can see that although the SSD Resnet-50 FPN model is more accurate, it is two times slower than MobileNetV1, three times slower than MobileNetV2 on GPU and the mobile nets can be much faster on Edge TPU. Hence, we can say that the SSD Resnet-50 FPN model is quite far from meeting the real-time requirements.

On comparing, the inference times between CPU, GPU and Edge TPU across all the three models, we can see that the

TABLE II: COCO MAP ON THE TEST SET

Model	COCO mAP	COCO mAP (Large)	COCO mAP (Medium)	COCO mAP (Small)
SSD MobileNetV1 Quantized	29.84	61.35	28.98	4.41
SSD MobileNetV2 Quantized	29.48	64.09	29.11	3.95
SSD ResNet-50 FPN	46.37	69.49	48.56	13.64

TABLE III: PERFORMANCE OF INFERENCE TIME (ms) AND FPS ON THE CPU, GPU AND THE EDGE TPU

Model	Device	capture (ms)	pre-process (ms)	inference (ms)	post-process (ms)	total (ms)	FPS
SSD MobileNetV1 Quantized	CPU	5.17	0.61	52.63	21.12	79.53	12.57
	GPU	2.61	0.2	39.49	23.77	66.07	15.13
	Edge TPU	12.91	9.23	**13.41**	0.35	35.90	**27.86**
SSD MobileNetV2 Quantized	CPU	4.84	0.62	43.77	20.06	69.3	14.43
	GPU	2.52	0.2	24.16	21.8	48.69	20.54
	Edge TPU	13.03	9.45	**8.54**	0.33	31.34	**31.91**
ResNet-50 FPN	CPU	4.9	0.61	232.5	15.74	253.76	3.94
	GPU	2.85	0.2	125.3	16.14	144.5	6.92

SSD MobileNetV2 Quantized model on Edge TPU has the best inference time and can process the most images in the least time (approximately 32 images per second, see Table III and Fig. 5). We can also see that frame capturing by [7], pre-processing takes more time (since image resizing on an ARM CPU requires more time) on the Edge TPU.

Total inference times can be improved further by parallelizing frame capturing, pre-processing, inference, and post-processing using threads and synchronized queues. We are able to process 51 images per second on the Edge TPU with the SSD MobileNetV2 Quantized model after parallelization. However, the performance after parallelization is mainly constrained by the task of frame capturing, as the task of frame capturing is the most time-consuming (as can be seen in Table III).

VII. Conclusion

In this paper, we present a new dataset for detecting IndyCar race cars, and we evaluate and compare real-time inference of detection models on edge TPU and CPU, GPU servers in terms of accuracy and inference time. The results also show that more images need to be labeled to detect objects more accurately. Currently, the best inference time for race car detection is obtained by SSD MobileNetV2 on the Edge TPU amongst the models used in our experiments and the task of frame capturing from video-stream on an ARM CPU is the bottleneck of real-time object detection while running the model on Edge TPU. The experiments also bring to the attention, the model constraints on Edge TPU as the Edge TPU only supports TensorFlow Lite models. If a model has operations that are not supported in TensorFlow Lite, the model cannot be converted. This can be a limiting factor as results with better accuracy could be produced with other models that are currently not supported by Edge TPU.

VIII. Future Work

Our research is motivated and tested by real-world automotive vehicle applications, and this paper describes a recent work of our collaboration with the IndyCar company on an important application to Indianapolis 500 format racing with car and track sensors. The software testbed involves distributed offline training and online inference for event detection over time-series from sensors in our earlier publications and image data analysis from video feeds here. Image detection models can achieve significant results when trained with a sufficient amount of data. For future work, we will label more images to improve detection performance. However, data scarcity is one limitation for special anomaly events such as car crash where data augmentation will be explored using GAN-based methods.

IX. Acknowledgement

We gratefully acknowledge the support from NSF CIF21 DIBBS 1443054: Middleware and High Performance Analytics Libraries for Scalable Data Science, Science and NSF EEC1720625: Network for Computational Nanotechnology (NCN) Engineered nanoBio node, NSF OAC 1835631 CINES: A Scalable Cyberinfrastructure for Sustained Innovation in Net-work Engineering and Science, and Intel Parallel Computing Center (IPCC) grants. We would also like to express our appreciation for La'Andrea Gates, TaJuan Beckworth, Sumeet Mishra, and the FutureSystems team.

References

[1] Indycar series. https://en.wikipedia.org/wiki/IndyCar_Series. [Online; accessed 30-Oct-2019].

[2] Tensorflow detection model zoo. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md. [Online; accessed 21-October-2019].

[3] Tensorflow models - mobilenet. https://github.com/tensorflow/models/tree/master/research/slim/nets/mobilenet. [Online; accessed 30-October-2019].

[4] Tensorflow models - mobilenetv1. https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.md. [Online; accessed 30-October-2019].

[5] What is indycar? https://www.indycar.com/Fan-Info/INDYCAR-101/What-Is-INDYCAR. [Online; accessed 30-Oct-2019].

[6] Mlperf:fair and useful benchmarks for measuring training and inference performance of ml hardware, software, and services, 2019. https://mlperf.org.

[7] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[8] T. Choudhari. Prioritized task scheduling in fog computing. In *Acmse Conference*. ACM, 2018.

[9] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314. ACM, 2011.

[10] C. Consortium. Mscoco evaluation protocol. [Online; accessed 31-October-2019].

[11] Coral. System-on-module datasheet. https://coral.withgoogle.com/docs/som/datasheet/#power-consumption. [Online; accessed 31-October-2019].

[12] N. De Rita, A. Aimar, and T. Delbruck. Cnn-based object detection on low precision hardware: Racing car case study. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 647–652. IEEE, 2019.

[13] Z. Dong, Y. Liu, H. Zhou, X. Xiao, Y. Gu, L. Zhang, and C. Liu. An energy-efficient offloading framework with predictable temporal correctness. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, page 19. ACM, 2017.

[14] Z. Fan. Prolonging lifetime via mobility and load-balanced routing in wireless sensor networks. In *2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–6. IEEE, 2009.

[15] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Guo. Tiresias: A {GPU} cluster manager for distributed deep learning. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, pages 485–500, 2019.

[16] C. Gupta, A. S. Suggala, A. Goyal, H. V. Simhadri, B. Paranjape, A. Kumar, S. Goyal, R. Udupa, M. Varma, and P. Jain. Protonn: Compressed and accurate knn for resource-scarce devices. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1331–1340. JMLR. org, 2017.

[17] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[18] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[19] H.-N. Hu, Q.-Z. Cai, D. Wang, J. Lin, M. Sun, P. Krähenbühl, T. Darrell, and F. Yu. Joint monocular 3d vehicle detection and tracking. *arXiv preprint arXiv:1811.10742*, 2018.

[20] B. Huang, M. Boehm, Y. Tian, B. Reinwald, S. Tatikonda, and F. R. Reiss. Resource elasticity for large-scale machine learning. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 137–152. ACM, 2015.

[21] Y.-H. Kao, B. Krishnamachari, M.-R. Ra, and F. Bai. Hermes: Latency optimal task assignment for resource-constrained mobile computing. *IEEE Transactions on Mobile Computing*, 16(11):3056–3069, 2017.

[22] A. Kusupati, M. Singh, K. Bhatia, A. Kumar, P. Jain, and M. Varma. Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. In *Advances in Neural Information Processing Systems*, pages 9017–9028, 2018.

[23] D. Li, D. Zhao, Q. Zhang, and Y. Zhu. An autonomous driving experience platform with learning-based functions. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1174–1179. IEEE, 2018.

[24] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[25] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[26] X. Lyu, H. Tian, C. Sengul, and P. Zhang. Multiuser joint task offloading and resource optimization in proximate clouds. *IEEE Transactions on Vehicular Technology*, 66(4):3435–3447, 2016.

[27] F. Messaoudi, A. Ksentini, and P. Bertin. On using edge computing for computation offloading in mobile network. In *GLOBECOM 2017-2017 IEEE Global Communications Conference*, pages 1–7. IEEE, 2017.

[28] O. Munoz, A. Pascual-Iserte, and J. Vidal. Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading. *IEEE Transactions on Vehicular Technology*, 64(10):4738–4755, 2014.

[29] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference*, page 3. ACM, 2018.

[30] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[31] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[32] S. Sardellitti, G. Scutari, and S. Barbarossa. Joint optimization of radio and computational resources for multicell mobile-edge computing. *IEEE Transactions on Signal and Information Processing over Networks*, 1(2):89–103, 2015.

[33] T. Sen and H. Shen. Machine learning based timeliness-guaranteed and energy-efficient task assignment in edge computing systems. In *2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC)*, pages 1–10. IEEE, 2019.

[34] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[35] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[36] Tzutalin. Labelimg. https://github.com/tzutalin/labelImg. [Online; accessed 31-October-2019].

[37] H. Wang, J. Gong, Y. Zhuang, H. Shen, and J. Lach. Healthedge: Task scheduling for edge computing with health emergency and human behavior consideration in smart homes. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 1213–1222. IEEE, 2017.

[38] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li. Mobile-edge computing: Partial computation offloading using dynamic voltage scaling. *IEEE Transactions on Communications*, 64(10):4268–4282, 2016.

[39] C. Widanage, J. Li, S. Tyagi, R. Teja, B. Peng, S. Kamburugamuve, D. Baum, D. Smith, J. Qiu, and J. Koskey. Anomaly detection over streaming data: Indy500 case study. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 9–16. IEEE, 2019.

[40] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, et al. Gandiva: Introspective cluster scheduling for deep learning. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 595–610, 2018.

[41] X. Yang, Y. Guo, and Y. Liu. Bayesian-inference-based recommendation in online social networks. *IEEE Transactions on Parallel and Distributed Systems*, 24(4):642–651, 2012.

[42] S. Yao, Y. Zhao, H. Shao, S. Liu, D. Liu, L. Su, and T. Abdelzaher. Fastdeepiot: Towards understanding and optimizing neural network execution time on mobile and embedded devices. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, pages 278–291. ACM, 2018.

[43] Y. Yao, M. Xu, Y. Wang, D. J. Crandall, and E. M. Atkins. Unsupervised traffic accident detection in first-person videos. *arXiv preprint arXiv:1903.00618*, 2019.

[44] H. Zhang, L. Stafman, A. Or, and M. J. Freedman. Slaq: quality-driven scheduling for distributed machine learning. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 390–404. ACM, 2017.