Offloaded Execution of Deep Learning Inference at Edge: Challenges and Insights

Swarnava Dey (Author)
Embedded Systems and Robotics
TCS Research & Innovation
Kolkata, India
Email: swarnava.dey@tcs.com

Jayeeta Mondal (Author)
Embedded Systems and Robotics
TCS Research & Innovation
Kolkata, India
Email: jayeeta.m@tcs.com

Arijit Mukherjee (Author)
Embedded Systems and Robotics
TCS Research & Innovation
Kolkata, India
Email: mukherjee.arijit@tcs.com

***Abstract*—Efforts to leverage the benefits of Deep Learning(DL) models for performing inference in resource constrained embedded devices is very popular nowadays. Researchers worldwide are trying to come up with software and hardware accelerators that make pre-trained DL models suitable for running the inference phase in these devices. Apart from software and hardware accelerators, DL model partitioning and offloading to Cloud or Edge network servers is becoming more and more practicable with increasing importance of Edge Computing. DL inference workflow partitioning and offloading can augment software / hardware acceleration for improved latency and energy efficiency in resource constrained embedded systems. Efficacy of a computation offloading to system is dependent on proper profiling of timing and energy required for processing DL algorithms. In this work we implement a DL inference offloading system using Raspberry Pi 3 based robot vehicle with a hardware accelerator from Intel (Neural Compute Stick). We report the workload partitioning approach, detailed experimental results and performance improvements achieved. We demonstrate that the current approach of DL execution profiling without considering dynamic system load of the edge device results in sub-optimal partitioning of DL algorithm and provide a solution approach to that.**

***Index Terms*—Deep learning, Acceleration, Embedded, Inference, Edge Computing, Fog Computing**

I. Introduction

Deep Learning [1] (DL) has acted as an accelerator to our journey towards a data driven world by minimizing the role of domain specific and tedious *feature engineering* and allowing high performing pre-trained models to be used for newer datasets with relatively low re-training. However DL inference phase, which is used for classification and/or prediction requires fair amount of system resources (compute, memory, cache etc.) to run in real time. Efforts to leverage the benefits of DL models for performing inference in resource constrained embedded devices is in focus since past few years. Researchers worldwide are trying to come up with software and hardware accelerators that make pre-trained DL models suitable for running the inference phase in these devices. Software transformations typically include model compression [2]–[4], quantization [5], [6], approximation, network pruning and coming up with new smaller network architectures [7] with comparable efficacy. Hardware friendly optimizations like matrix multiplication factorization, data path optimization, parallel operations (e.g. convolutions) are some of the areas that attracted attention [8], [9] - paving the way for commercially available hardware accelerators for DL inference. Partitioning and distributing workload is successfully applied to accelerate diverse embedded applications [10]–[14]. DL model partitioning and offloading to Cloud or Edge network servers is a complementary approach to these accelerations and can be applied over and above these transforms. Such partitioned DL inference is becoming more and more practicable with increasing importance of Edge Computing and may be required in following scenarios:

- while using a large pre-trained model that may not fit hardware accelerator memory,
- inference rate required is not achievable even using accelerators and
- the embedded device has certain other tasks to perform along with DL inference.

In this work we focus specifically in reviewing , analysing and implementing DL inference offloading technology. We discuss the DL inference offloading process, state of the art (SoA) and its technology enablers in detail. We implement a DL inference offloading system using Raspberry Pi 3 [15] (Rpi) based robot vehicle, USB connected Intel Movidius Neural Compute Stick [16] (NCS), a hardware accelerator for DL inference.We report the workload partitioning approach, detailed experimental results and performance improvements achieved. We demonstrate that the current approach of DL execution profiling without considering dynamic system load of the edge device results in sub-optimal partitioning of DL algorithm and provide a solution approach to that. Throughout this article we use the term *DNN* to refer to any deep neural network like CNN [17], RNN etc., *edge device* to refer to the network end embedded systems like robots, autonomous vehicles, wearable devices and *edge server* to refer to the untapped resources in network edge that can act as offloading servers, e.g. smartphones, gateways, MEC servers [18]. The rest of this paper is organized as follows: in section II DNN Execution Offloading and the enablers for that are discussed in detail along with the SoA. In section III we present the experimental setup, experiments performed and the detailed inferences. Section IV concludes the paper.

II. DNN Execution Offloading

In this section we discuss the major technical enablers for accelerating DNN inference time by partitioning between constrained embedded edge devices and edge/Cloud servers.

A. Execution Profiling

Profiling DNN execution time and power requirement is critical for designing effective offloading strategies, by taking informed decisions about whether or not to offload part of a DNN inference pipeline. The *de facto* standard in this regard is the layer-wise profiling and offloading of DNNs. Profiling the inference process layer by layer for each and every DNN is impracticable and hence researchers try to build generalized prediction models for predicting execution time, power etc. for different types of layers on different classes of hardware as depicted in Fig. 1. Linear regression based models are popular in this context as those are lightweight in terms of system resource consumption and can be easily built from structure of the network and different configuration parameters.

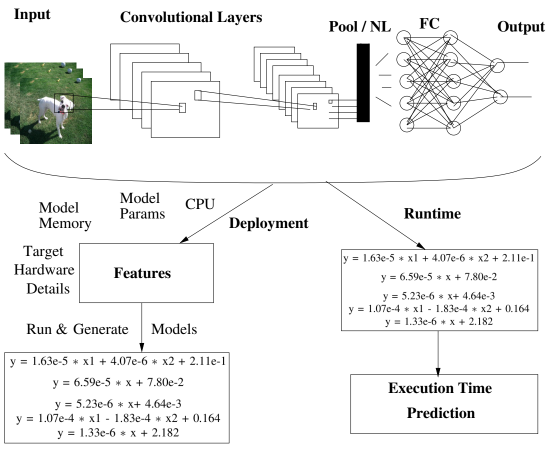

Figure 1. Execution profile and prediction - standard practice

Neurosurgeon [19] uses configuration parameters like input and output size, number of neurons and number of convolution operation across all filters (for Convolutional layers) and represent the layer-wise workload of different layer types in Giga FLOPS (Floating Point Operations / Second). It uses these models to predict the execution time of any layer of that type during at DNN inference time.

In FastDeepIoT [20] authors show that the relationship between network structure and configuration is non-linear and trivial linear regression models over the entire model are unable to predict execution time accurately. Source of these nonlinear relationship are code level optimizations in DNN inference libraries (data alignment, parallel operations, loop unrolling etc.) and memory / cache access patterns. To solve the above problem, authors use a tree structured linear regression model that divides the network into discrete regions, where within each region standard linear regression can be used to build prediction models for DNN layer execution time prediction. The independent variable is formed using features from FLOPs; input, output and intermediate memory; and number of parameters in the model for fully connected, convolution and recurrent type layers.

In [21] the model size is added to the standard explanatory variables in this context and provide the regression equations for layer latency prediction.

In [22] computation and Bluetooth communication latency, energy is profiled for wearable devices and smartphones. The prediction models are built using decision trees and linear regression. In this work the smartphone play the role of offloading server.

In [23] authors present extensive results and insights on running CNN and FC networks on different SoCs used in mobile phones, however they do not build any prediction models of execution time and power consumption.

B. Partitioning Strategy for Offloading

DNN partitioning is an important enabler for offloaded execution. Partition points in a DNN data flow graph can be determined with target of energy and/or latency minimization. As shown in Fig. 2, partitioning can be targeted for Edge-Cloud hierarchy.

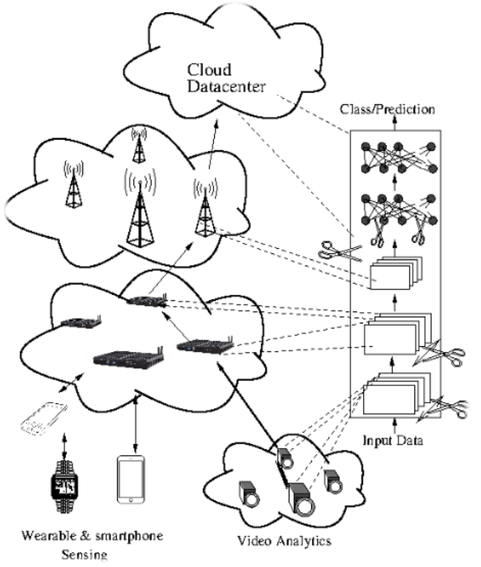

Figure 2. Partitioning points targeted for Edge-Cloud hierarchy

The standard approach for layer wise partitioning is predicting the processing latency, data communication latency and energy for each layer of a DNN and finding the optimum point that minimizes both. As this decision is made at the data source, i.e., constrained edge device - lightweight optimization algorithms are required.

In [24] authors propose a design guideline for partitioning a CNN at the end of the last convolutional layer where the data communication requirement is less and the fully connected layers requiring high run-time memory, are executed in the server side.

In [25] the models trained using BranchyNets [26] are used for implementing the partitioning. To implement such partitioning in real systems, the computation graphs of common run-time libraries like TensorFlow, Caffe needs to be partitioned in any case.

The standard strategy for partitioning used in [19], [22], [25] is finding out the cumulative latency/energy for each partition at each layer level and perform linear search to get the minimum out of those.

III. Experiments on DNN Execution Offloading

A. Experimental Setup

For our experiments we use two types of configurations. In the first configuration I Raspberry Pi 3 [15] is used as the edge device and smartphones, laptops, Nvidia Jetson Tx2 [27] are used as edge servers. In the second configuration II Raspberry Pi 3 [15] with Intel Movidius Neural Compute Stick [16], a hardware accelerator for DNNs, is used as the edge device and Nvidia Jetson Tx2, Intel Xeon based server class machines are used as edge servers. FLOPs benchmark is done through LINPACK [28], emphVFP Bench(iOS). We have observe that there is close correlation between the FLOPs benchmark and DNN execution profile in most of the cases. However, for fine grained profiling we use the techniques mentioned in section II-A.

Table I
Edge Device and Edge Server configurations-I

Edge Server	GFLOPs	Edge Device	GFLOPs
Rpi-3b	2	Rpi-3b	4
Rpi-3b	2	iPhone-6	10
Rpi-3b	2	Xeon-E3-1246 PC	20
Rpi-3b	2	Jetson Tx2	50

Table II
Edge Device and Edge Server configurations-II

Edge Server	GFLOPs	Edge Device	GFLOPs
Rpi-3b + NCS	36	Intel i7-7500U PC	44
Rpi-3b + NCS	36	Xeon-E5-2695 PC	130
Rpi-3b + NCS	36	Xeon-E5-2696 PC	330

We selected three different pre-trained models for our experiments as given in table III along with the edge device only execution time in the two different configurations. Of these three models Squeezenet [29] is fastest - suitable for running inference in constrained edge devices (1.24M params). Inception V3 [31] is the most sophisticated 311 layer model (23.83M params), requiring huge system resources and Alexnet [30] mid-sized CNN with 25 layers.

Table III
DNNs used for Benchmarking

Net Name	Layers	Time(ms) Rpi-3b	Time(ms) Rpi+NCS
Squeezenet	69	2032	48
AlexNet	25	3200	91
Inception-v3	311	7000	326

B. Benchmark Results and Discussion

In this section we discuss the DL partitioning algorithm and the effect of partitioning a few popular pre-trained networks. Let us consider that the processing latency at Edge device and server node for each layer l of a N layered DNN is given by P_l^e and P_l^c respectively. For the data transfer latency let us consider that the output activation of a layer l be of size D_l bytes. We consider the channel establishment time to be α, for *one* packet (Maximum transfer Unit) of M bytes and β to be the transmission time per byte. Based on layerwise profiling and the regression models (detailed in section II-A) built from those benchmark results, we implement a function *predict_latency* that can predict the processing latency of a DNN layer at runtime. Using the above parameters the following linear search algorithm 1 finds the optimum point of partition: We predict the partition points and use those for

Algorithm 1: Algorithm for Partition Selection

```
1  Algorithm: PartitionSelection
   Input: Parameters $M$, $\alpha$, $\beta$, finite sets of: edge
          processing latency of each layer
          $P^e = \{P_1^e, P_2^e, \dots P_n^e\}$, Cloud processing
          latency of each layer $P^c = \{P_1^c, P_2^c, \dots P_n^c\}$,
          output transfer size for each layer
          $D = \{D_1, D_2, \dots D_n\}$, and properties and
          configurations for processing latency prediction
          for each layer
          $L\_cfg = \{L\_cfg_1, L\_cfg_2, \dots L\_cfg_n\}$
   Output: The layer number around which partitioning
           yields best latency or energy
2  begin
3      for $i \leftarrow 1$ to $N$ do
4          $P_i^e \leftarrow predict\_latency(L\_cfg_i)$;
5          $T_i \leftarrow \frac{D_i}{M} \cdot \alpha + D_i \cdot \beta$;
6      for $i \leftarrow 1$ to $N$ do
7          $P_{total}^e \leftarrow 0$;
8          $P_{total}^c \leftarrow 0$;
9          $PART \leftarrow 0$;
10         for $j \leftarrow 1$ to $i$ do
11             $P_{total}^e \leftarrow P_{total}^e + P_i^e$;
12         for $j \leftarrow i + 1$ to $N$ do
13             $P_{total}^c \leftarrow P_{total}^c + P_i^c$;
14         $PART_j \leftarrow P_{total}^e + P_{total}^c + T_j$;
15     return $argmin\{PART\}$
```

making the offloading decision for each of the CNNs and measure the resulting inference rate. In our experiments, we emulate different interconnection network speeds on top of a Wi-Fi network. The Alexnet CNN is always partitioned at the first pooling layer after the first convolution in both the configurations - with a slow Rpi and much faster Rpi with NCS connected via USB. the partition point selection with respect to the Alexnet architecture is shown in Fig. 3 and Fig. 4 The base

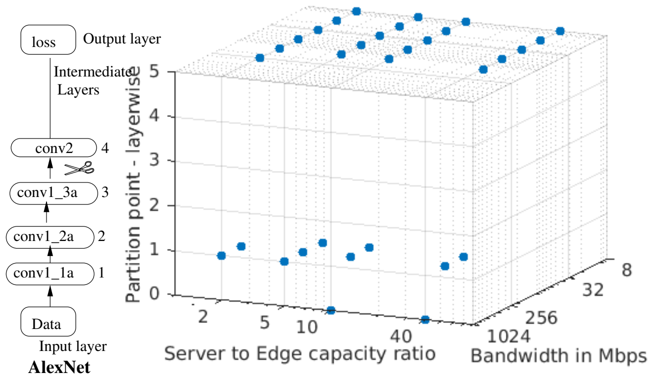

Figure 3. Configuration-I - Alexnet - Partition point selection

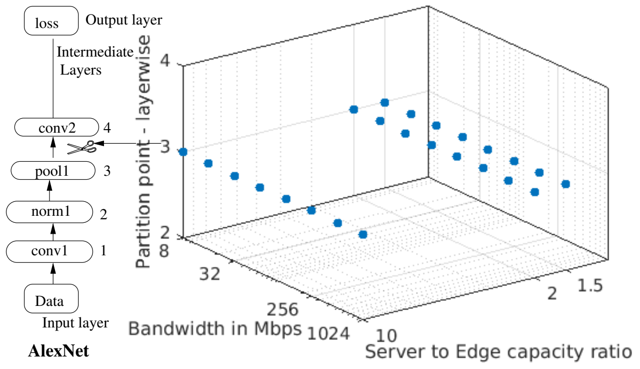

Figure 4. Configuration-II - Alexnet - Partition point selection

inference rate for Alexnet in Rpi is 0.3fps in Configuration-I (Fig. 5) which increases to around 3fps with offloading. Though the edge device inference rate in configuration-II

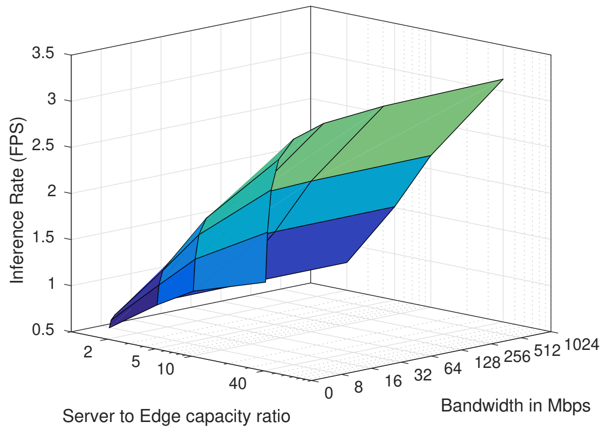

Figure 5. Configuration-I - Alexnet - Inference rate

is around 11fps, it also gets six fold improvement due to offloading (Fig. 6. The inference rates shown in the results are not frame processing rates, which are much slower as due to other activities like getting image frames from Rpi camera, getting output and generating bounding box and displaying frames in an video object detection setup.

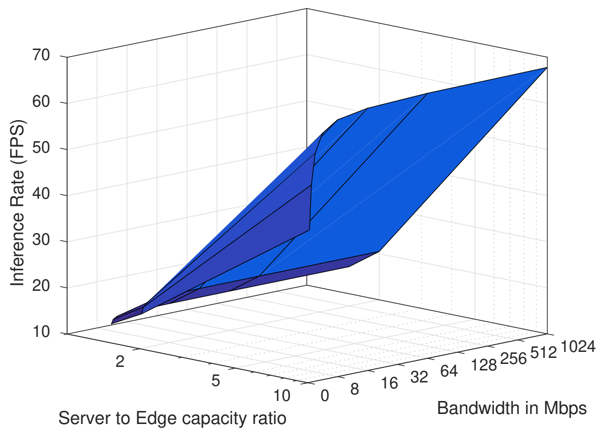

Figure 6. Configuration-II - Alexnet - Inference rate

The trend for Squeezenet partitioning is also in the first pooling layer (Fig. 7) with the only exception when the feature map upload time is very high, where the partition point is chosen at *fire5* module (Fig. 8). The base inference rate

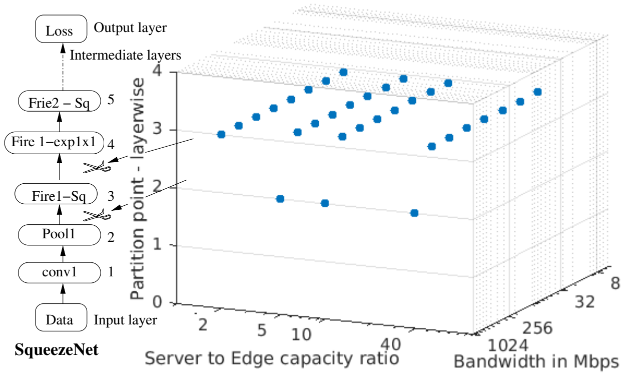

Figure 7. Configuration-I - Squeezenet - Partition point selection

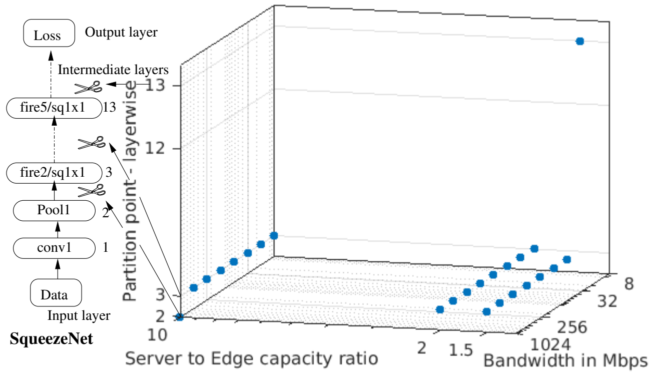

Figure 8. Configuration-II - Squeezenet - Partition point selection

for Squeezenet in Rpi is 0.5 fps in configuration-I (Fig. 9) which gets *4x* improvement due to offloading (Fig. 10). The

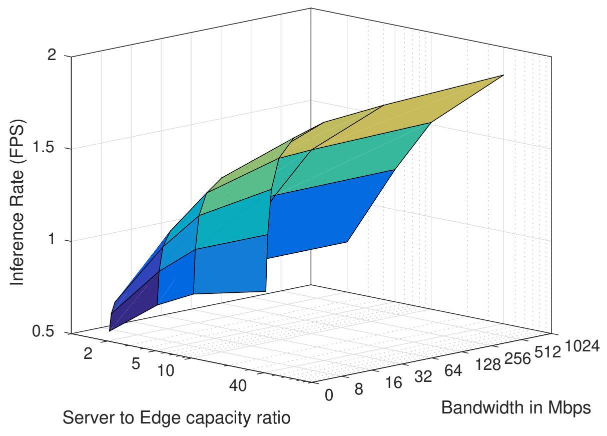

Figure 9. Configuration-I - Squeezenet - Inference rate

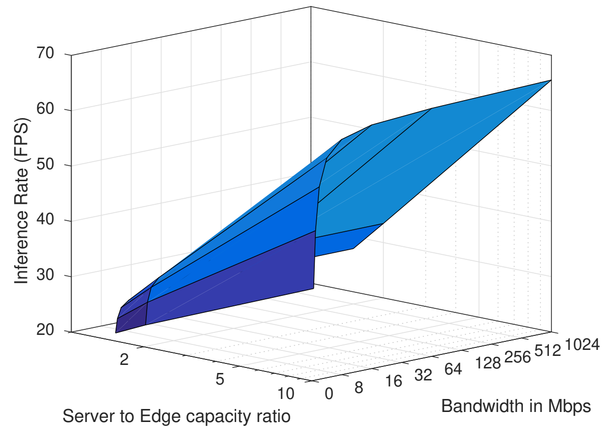

Figure 10. Configuration-II - Squeezenet - Inference rate

Rpi+NCS configuration, denoting high end edge device gets *3x* improvement.

The Inception V3 CNN in Configuration-I and II shows a trend of full offloading or offloading just after first convolution if bandwidth and processing power of the edge server are high (Fig 11). However, when the bandwidth is lower the Edge processing is preferred for Configuration-II with NCS (Fig. 12) (processing happens till mixed-2 inception module) and somewhat preferred for Configuration-I (processing happens till conv4). This model, in configuration-II has a base inference rate of 3fps (Fig.13) and gets around *8x* improvement (Fig.14). From this we conclude that larger the size of the model, more is the scope of overall latency reduction due to offloading.

From our experiments we infer that, if the Edge Server is fast (two to ten times) and network bandwidth is more than 16Kbps , offloading will *always* improve the performance. We observe that the effective bandwidth for uploading the feature

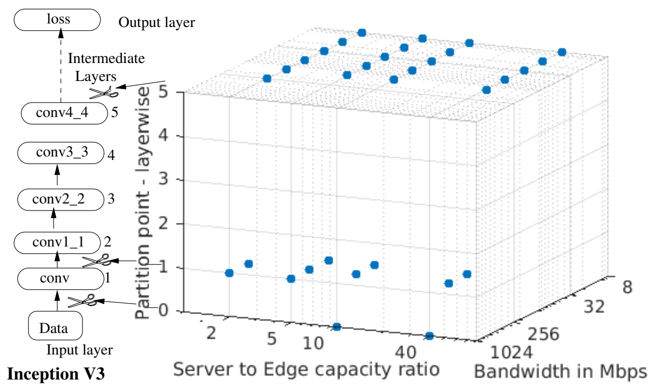

Figure 11. Configuration-I - Inception V3 - Partition point selection

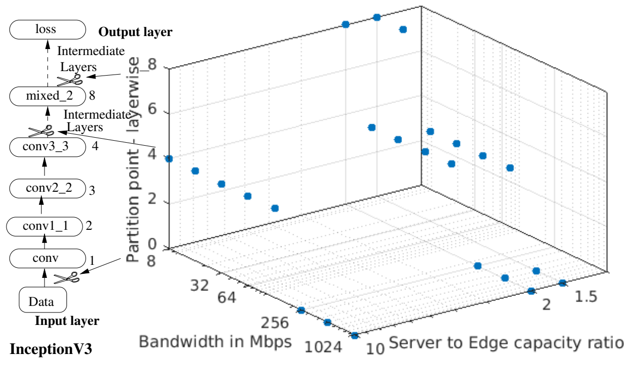

Figure 12. Configuration-II - Inception V3 - Partition point selection

maps from the Rpi, was quite slow and ranged between 8 Kbps to 1 Mbps maximum. This is in sharp contrast to the standard uplink speed of 4G and Wi-Fi connections. The connection setup time and power management of the wireless transceivers, seems to be the reason for this effective speed. From this we can draw the following conclusions:

- If the effective upload speeds are high, as observed in *5G* application scenarios [32] - Enhanced Mobile Broadband(eMBB), Massive Machine Type Communication (mMTC) and Ultra-reliable and Low Latency Communications (uRLLC), DNN partitioning will possibly be required only in case of network outage and unavailability - in all other cases DNN execution will be offloaded fully in one hop connected edge server.
- There is a need for transforming the DNN inference pipeline into stream processing to gain high uplink bandwidth advantage. In its current form DNN inference acts on a single frame grabbed from input video camera.
  In our experimental setup given in Fig. 15, we perform a object recognition test on a Raspberry Pi3 + NCS based robot vehicle, developed by us. This vehicle navigates and simultaneously recognizes objects. We use the partition point based offloading scheme on this device but achieve only around 5fps (refer Fig. 16), much lesser than the

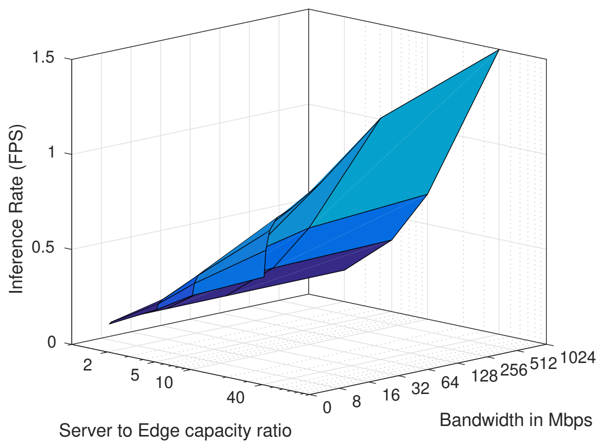

Figure 13. Configuration-I - Inception V3 - Inference rate

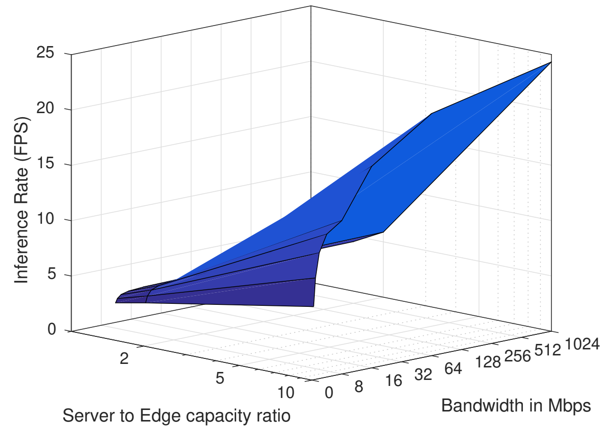

Figure 14. Configuration-II - Inception V3 - Inference rate

expected 15fps inference rate that we observed in our experiments.

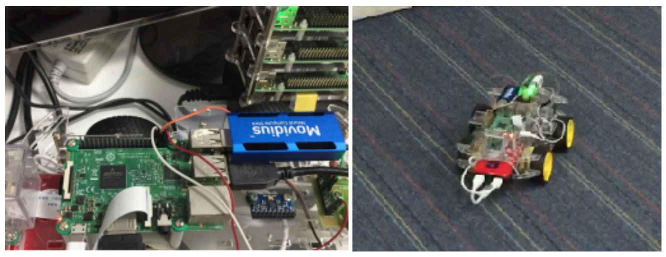

Figure 15. Raspberry Pi3 with NCS based robot vehicle that navigates and recognises objects

C. Design Guideline for DNN Profiling

On checking the system load details we observe that the navigation algorithm, camera handling take up CPU and memory, increasing the latency of the partial DNN inference pipeline. This is common in most of the embedded devices

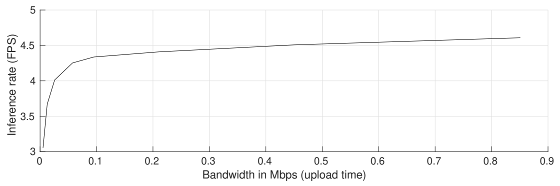

Figure 16. Inference rate for Object recognition by navigating robot vehicle

which follow *sense-analyze-respond* cycle. Based on this we recommend a design guideline of taking into account the dynamic system load of edge device along with the predicted DNN execution latency while choosing the partition points, which none of the earlier systems do. Such profiling can be achieved in *two* ways:

- Taking system load as a parameter to the regression models built in section. II-A and running the DNN profiling under simulated load scenarios
- Use the system load factor to scale the processing latency in edge and Cloud resource.

Though load scenarios in a embedded environment is often known (e.g. video processing, display handling and navigation for our robot), it is often impracticable to track all such scenarios. Thus we use system load as a factor to scale the processing latency in edge device. As explained in the article [33], for a Rpi based system we use Linux Operating System's (OS) *loadavg* command to get processor load. As we are using a run-time partition selection for a task that has a processing latency much lesser than a minute, it is appropriate to use a *loadavg* for last *one* minute. That value denotes the cumulative number of processes that are either running currently or are ready for running in last *one* minute. We use *loadavg* value averaged by the number of cores (Quadcore for Rpi 3) and use that factor to scale up the edge processing latency, when load average is more than *one*. We modify the partition selection algorithm 1 by adding a line after line *four*, as depicted in Fig. 17, where average load is obtained using a suitable OS function call. By applying this scaling the edge execution latency takes into account the dynamic system load and we achieve around 10fps in our Rpi robot setup (Fig. 15), which is *2x* faster than the implementation using edge processing latency values as is.

```
2 begin
3     for $i \leftarrow 1$ to $N$ do
4         $P_i^e \leftarrow predict\_latency(L\_cfg_i)$;
5         $P_i^e \leftarrow \frac{P_i^e}{loadavg\_per\_core}$;
6         $T_i \leftarrow \frac{D_i}{M} \cdot \alpha + D_i \cdot \beta$;
7     for $i \leftarrow 1$ to $N$ do
8         $P_{total}^e \leftarrow 0$;
```

Figure 17. Modified partition Selection Algorithm

IV. Conclusion

In this paper we implemented a execution offloading system for Deep Learning inference phase. We experimented with a embedded System on Chip along with a DL inference accelerator and demonstrated that intelligent offloading can be useful for both resource constrained embedded systems, as well a more capable edge devices. We discussed the State of Art in this research area and outlined several challenges and insights, which when taken into account, may result in a better system design for DNN inference offloading. In future we would like to design a DNN profiling system that precisely models dynamic system load of Internet of Things devices that follow *sense-analyze-actuate* cycle and use that for offloaded execution of different DNN variants.

References

[1] Bengio Y: "Learning Deep Architectures for AI". Now Publishers Inc., Hanover, MA, USA; 2009.

[2] S. Han, H. Mao, and W. J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." arXiv preprint arXiv:1510.00149, 2015.

[3] S. Bhattacharya and N. D. Lane. "Sparsification and separation of deep learning layers for constrained resource inference on wearables". In Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM, pages 176189. ACM, 2016

[4] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy. "MCDNN: an approximation-based execution framework for deep stream processing under resource constraints". In Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys16), pages 123136, 2016.

[5] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," arXiv preprint arXiv:1806.08342 , 2018

[6] Antonio Polino, Razvan Pascanu, Dan Alistarh, "Model compression via distillation and quantization" arXiv preprint arXiv:1802.05668 , 2018

[7] 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.

[8] Kamel Abdelouahab, Maxime Pelcat, Jocelyn Serot, Franois Berry, "Accelerating CNN inference on FPGAs: A Survey", arXiv preprint arXiv:1806.01683

[9] Chen Zhang, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong, "Caffeine: towards uniformed representation and acceleration for deep convolutional neural networks," In Proceedings of the 35th International Conference on Computer-Aided Design (ICCAD '16). ACM, New York, NY, USA, Article 12, 8 pages. doi:10.1145/2966986.2967011, 2016

[10] S. Dey and R. Dasgupta, "Fast Boot User Experience Using Adaptive Storage Partitioning," 2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, Athens, 2009, pp. 113-118, doi:10.1109/ComputationWorld.2009.121

[11] Swarnava Dey and Arijit Mukherjee. 2016. "Robotic SLAM: a Review from Fog Computing and Mobile Edge Computing Perspective," In Adjunct Proceedings of the 13th International Conference on Mobile and Ubiquitous Systems: Computing Networking and Services (MOBIQUITOUS 2016). ACM, New York, NY, USA, 153-158. doi:10.1145/3004010.3004032

[12] Ansuman Banerjee, Arijit Mukherjee, Himadri Sekhar Paul, and Swarnava Dey. 2013. "Offloading work to mobile devices: an availability-aware data partitioning approach," In Proceedings of the First International Workshop on Middleware for Cloud-enabled Sensing (MCS '13). ACM, New York, NY, USA, , Article 4 , 6 pages. doi:10.1145/2541603.2541605

[13] Swarnava Dey, Arijit Mukherjee, Arpan Pal, and P. Balamuralidhar. 2018. "Partitioning of CNN Models for Execution on Fog Devices," In Proceedings of the 1st ACM International Workshop on Smart Cities and Fog Computing (CitiFog'18), ACM, New York, NY, USA, 19-24. doi:10.1145/3277893.3277899

[14] S. Dey and A. Mukherjee, "Implementing Deep Learning and Inferencing on Fog and Edge Computing Systems," 2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Athens, 2018, pp. 818-823. doi:10.1109/PERCOMW.2018.8480168

[15] *Raspberry Pi 3: Specs, benchmarks & testing*, https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks

[16] *Intel Movidius Neural Compute Stick*, ETSI, https://movidius.github.io/ncsdk/ncs.html.

[17] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. 2014. "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition." In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW '14). IEEE Computer Society, Washington, DC, USA, 512-519. DOI=http://dx.doi.org/10.1109/CVPRW.2014.131

[18] "Multi-access Edge Computing", ETSI, http://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing.

[19] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang, "Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge.," SIGARCH Comput. Archit. News 45, 1 (April 2017), 615-629. DOI: https://doi.org/10.1145/3093337.3037698, 2017.

[20] Shuochao Yao, Yiran Zhao, Huajie Shao, ShengZhong Liu, Dongxin Liu, Lu Su, and Tarek 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 (SenSys '18), Gowri Sankar Ramachandran and Bhaskar Krishnamachari (Eds.). ACM, New York, NY, USA, 278-291. DOI: https://doi.org/10.1145/3274783.3274840, 2018.

[21] En Li, Zhi Zhou, and Xu Chen, "Edge Intelligence: On-Demand Deep Learning Model Co-Inference with Device-Edge Synergy," In Proceedings of the 2018 Workshop on Mobile Edge Communications (MECOMM'18). ACM, New York, NY, USA, 31-36. DOI: https://doi.org/10.1145/3229556.3229562, 2018

[22] Mengwei Xu, Feng Qian, and Saumay Pushp, "Enabling Cooperative Inference of Deep Learning on Wearables and Smartphones", eprint:arXiv:1712.03073, 2017

[23] Nicholas D. Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar, " An Early Resource Characterization of Deep Learning on Wearables, Smartphones and Internet-of-Things Devices," In Proceedings of the 2015 International Workshop on Internet of Things towards Applications (IoT-App '15). ACM, New York, NY, USA, 7-12. doi:10.1145/2820975.2820980

[24] J.H. Ko, T. Na, M.F. Amir and S. Mukhopadhyay, "Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms," eprint:arXiv:1802.03835, 2018.

[25] E. Li, Z. Zhou and X. Chen, "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy ," in Proceedings of the 2018 Workshop on Mobile Edge Communications, MECOMM@SIGCOMM 2018, pp. 3136 Budapest, Hungary, August 20, 2018

[26] Surat Teerapittayanon, Bradley McDanel, H. T. Kung, "BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks,", eprint: arXiv:1709.01686, 2017

[27] *Jetson TX2*, ETSI, https://elinux.org/Jetson_TX2.

[28] Jack Dongarra, "The LINPACK Benchmark: An Explanation," In Proceedings of the 1st International Conference on Supercomputing, Elias N. Houstis, Theodore S. Papatheodorou, and Constantine D. Polychronopoulos (Eds.), Springer-Verlag, London, UK, UK, 456-474, 1987.

[29] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and 1mb model size," arXiv preprint arXiv:1602.07360, 2016

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. "ImageNet classification with deep convolutional neural networks." In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12), F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.), Vol. 1. Curran Associates Inc., USA, 1097-1105.

[31] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. "Rethinking the Inception architecture for computer vision." arXiv preprint , 1512.00567, 2015. arxiv.org/abs/1512.00567

[32] "China Unicom Edge Computing Technology White Paper", https://builders.intel.com/docs/networkbuilders/china-unicom-edge-computing-technology-white-paper.pdf.

[33] Ray Walker, "Examining Load Average", Linux Journal, https://www.linuxjournal.com/article/9001.