https://doi.org/10.2352/ISSN.2470-1173.2019.15.AVM-033

Real-Time Traffic Sign Recognition using Deep Network for Embedded Platforms

Raghav Nagpal, Chaitanya Krishna Paturu, Vijaya Ragavan, Navinprashath R R, Radhesh Bhat, and Dipanjan Ghosh; PathPartner Technology Pvt Ltd (India)

Abstract

Road traffic signs provide vital information about the traffic rules, road conditions, and route directions to assist drivers in safe driving. Recognition of traffic signs is one of the key features of Advanced Driver Assistance Systems (ADAS). In this paper, we present a Convolutional Neural Network (CNN) based approach for robust Traffic Sign Recognition (TSR) that can run real-time on low power embedded systems. To achieve this, we propose a two-stage network: In the first stage, a generic traffic sign detection network localizes the position of traffic signs in the video footage, and in the second stage a country-specific classification network classifies the detected signs. The network sub-blocks were retrained to generate an optimal network that runs real-time on the Nvidia Tegra platform. The network's computational complexity and the model size are further reduced to make it deployable on low power embedded platforms. Methods like network customization, weight pruning, and quantization schemes were used to achieve an 8X reduction in computation complexity. The pruned and optimized network is further ported and benchmarked on embedded platforms like Texas Instruments Jacinto TDA2x SoC and Qualcomm's Snapdragon 820Automotive platform.

Keywords

ADAS (Advanced Driver Assistance Systems), TSR (Traffic Sign Recognition), CNN (Convolutional Neural Network), SSD (Single Shot multi-box Detector), Sparse, Distill, DSP (Digital Signal Processor), GPU (Graphics Processing Unit), CPU (Central Processing Unit), GFLOPS (Giga Floating-point Operations per Second), GMACS (Giga Multiply Accumulate operations per Second)

Introduction

Traffic signs along the roads are commissioned to inform, guide, and warn the automobile drivers. Typically, many of them will not be noticed because of the inattention of the drivers, vehicle speed, and poor visibility due to bad weather and by the headlights of oncoming vehicles in the night. Vision-based ADAS technologies can inform the automobile drivers about the road conditions, and support them inaccurately inferring about the environment around the ego vehicle. A robust traffic sign recognition system can help the drivers by informing and warning them appropriately to avoid accidents. Factors like lighting variations, occlusion of signs due to obstacles, and deformation of signs pose a challenge for a typical traffic sign recognition system. The need for executing this recognition system at real-time on low power embedded platforms without losing high recognition accuracy makes it more difficult to achieve. In this work, we propose embedded friendly CNN based Traffic sign recognition system to handle this problem.

Background

Recent years has seen a lot of progress in object detection methods by usage of convolution neural networks (CNNs). Recent object detection networks - such as Faster RCNN [1], R-FCN [2], Multibox [3], YOLO [5] and SSD [4] have shown promising results good enough to be deployed in end products and some of them are fast enough to be run on edge devices with small memory footprint. Typically, applications like traffic sign recognition requires real-time performance on edge devices. Many state-of-the-art methods that showed promising results on ImageNet and COCO object detection challenges [6], are optimized for higher accuracies, but are too slow for practical real-time use cases.

A subset of methods like R-FCN [2], SSD [4], and YOLO [5] offer a good tradeoff between decent accuracy and computational performance. Despite being relatively lightweight models, they are still not deployable on embedded platforms. In recent years SSD with Mobilenet [13] as feature extractor is one of the near real-time object detection method that can be deployed in edge devices. But, tasks like traffic sign recognition needs real-time performance, which these methods still fall short of.

As mentioned by A. Howard et al. [13], a different approach for obtaining small networks is shrinking, factoring or compressing pre-trained networks. Compression based on product quantization [7], hashing [8], pruning, Huffman coding [9] and etc. has been proposed in the literature. Additionally, various factorizations have been proposed to speed up the pre-trained network [10, 11]. Another method for obtaining a small network is distillation [12] which uses a large network to teach a smaller network. Some of these approaches are used in our method with modifications; and will be discussed in section Implementation Details.

Another interesting approach proposed by Denil et al. [21], is to exploit redundancy across filters and channels. He predicted 95% parameters in a DNN by exploiting the redundancy across filters and channels. Inspired by it, Denton et al. [22] achieved 2x speedups for the first two layers in a larger network. Both of these approaches used Low-Rank Approximation (LRA) with minimal accuracy drop. However, the network structure compressed by LRA is fixed; reiterations of decomposing, training/fine-tuning, and cross-validating are still needed to find an optimal structure for accuracy and speed trade-off. As the number of hyper-parameters in LRA method increases linearly with the layer depth.

Comparing to LRA, Sparse convolution algorithms proposed by Wie et al. [23] dynamically optimizes the compactness of networks. This approach is quite interesting because it doesn't change the overall network structure. A network is sparsified by training in iterations with high weight decay, and whenever the absolute value of weight falls below a predefined threshold, it is thresholded to zero. It also presented a method to recover the lost accuracy by fixing the zeroed-out coefficients as zeros and fine-tuning the non-zero coefficients. The sparsity induced is structurally constrained to get speed up on GPUs that use matrix multiplication to implement convolutions. In devices that do not use matrix multiplications for implementing convolution, a structural constraint may not be beneficial. Also, the sparsification approach used in the paper is extremely slow and time-consuming, making it difficult to use for large datasets. Manu et al. [16] proposed a quick method for sparsification and subsequent fine-tuning steps which converges in a reasonable number of iterations. This provides significant improvement in speed of thresholding compared to Wie et al. method.

Running inference on low-power embedded systems using 8-bit quantization can be much faster than floating point implementations. Quantization, as one of the key approaches, can effectively offload GPU, and make it possible to deploy on the fixed-point pipeline. Unfortunately, not all existing networks design are friendly to quantization. Quantized models have large accuracy gap against its float point models. Sheng et al. experimented with separable convolutions and analyzed the root cause of quantization loss. Where they proposed quantization friendly separable convolution architecture [24] with 8-bit inference with accuracy almost close to float pipeline. We use some of his customizations in our work.

It's now widely accepted that eight bits are enough for running inference on convolutional neural networks. But there's also a lot of evidence that it's possible to go lower than eight bits. These low bit networks are gaining a lot of traction.

In this paper, we propose a light-weight two stage traffic sign model which gives the real-time performance on embedded systems with good accuracy.

Implementation Details

In recent years, there have been several contributions towards improving traffic sign recognition using deep learning methodologies. However, these deep networks are large and computationally intense to deploy on embedded systems. We have designed a lightweight traffic sign recognition network that is able to achieve real-time performance on embedded platforms with good accuracy.

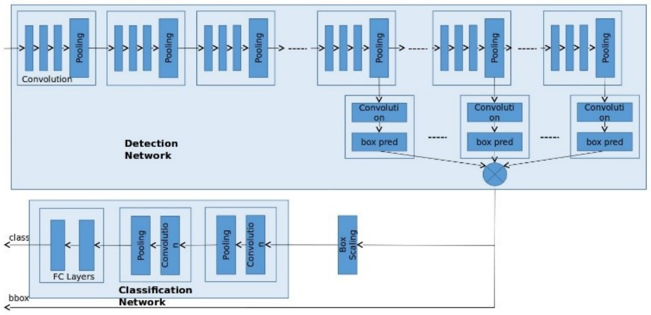


Figure 1. The above image illustrates the block diagram of the two phases of the network: a) a generic traffic sign detector (localizer) network and b) a country-specific classifier network.

Typically, traffic sign recognition involves detecting the position of traffic sign boards in road images and classifying them as a particular road sign. Though many state-of-the-art deep network architectures try to achieve both detection and classification together, we have chosen to split the network into two different phases: a) A generic traffic sign detector (localizer) network that can detect the position of any traffic signboard, and b) A country-specific classifier network that can classify the detected signs. This cascading arrangement eliminates the need for retraining the detector network for different countries. Only the classifier network needs to be retrained for various country-specific signs. The network structure is shown in Figure 1.

Two Stage Model

There are three main advantages of using a two-stage model (Figure 2). First, a generic detector for all types of traffic signs will be trained along with country-specific classifiers. We need to train the detector only once as the same can be reused for different countries. Also, the detector model being the computational heavy one, it saves retraining time in customizing the solution for different countries.

Second, it gives a computational advantage by running the classifier network for only those boxes that have a good probability score given by the detector. In architectures like SSD, the detector proposes more than 8K boxes in the default configuration. Running a class prediction for these boxes is computationally expensive, as in a given frame, we have 3 to 4 traffic signs at an average. So, by using a two-stage model, there is a significant saving in computation by running a separate lighter weight classifier (Lenet-5 in this case) to classify the traffic signs.

Third, by separating the classifier out the complexity of detector comes down, as the SSD feature computation is done for binary classification (Table 1).

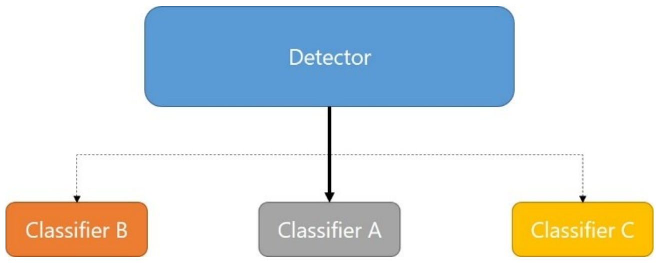


Figure 2. One detector with multiple classifiers

SSD feature computation: Table 1

Network	Computation of extra feature layers
Integrated SSD Network	K x K x (Number of Boxes x (Classes + 4))
Binary SSD Network	K x K x (Number of Boxes x (2 + 4))

*K - convolution filter size

Network Distillation and Customization

Embedded systems have the less computational power and less memory to execute analytics. Both of these limitations are not in favor of running deep neural networks which require high computation power. An obvious option to make deep networks less computationally expensive is to reduce the number of layers. But this affects the accuracy of the network significantly, as each layer contributes differently to the features.

To reduce the network size without affecting much accuracy, we propose a method of local retraining (local distillation). In this method, the sub-blocks between two consecutive pooling layers in the frozen model will be trained against a smaller sub-network with the same input and output feature dimensions. The rms loss function is used between the outputs of the frozen sub-block and the new custom defined sub-network. These sub-networks are trained locally with random noise as input. The new sub-networks are selected if they compute the same feature map outputs as that of the frozen model. Multiple sub-networks are trained and the one with the best performance is selected to replace the corresponding sub-block in the original network (Figure 3). We used random noise while training to prevent the sub-network from overfitting to particular input data.

The above process is repeated for all the sub-blocks and at the end, we have a smaller model which gives the same accuracy as the original one.

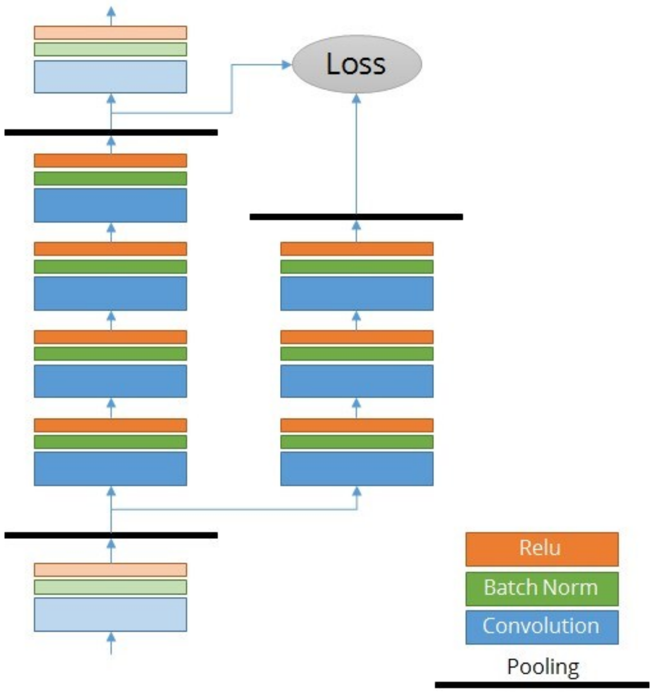


Figure 3. Sub-net level localized retraining

Detector Training

As a common practice, most of the deep learning methods reshape the input image to the square template before feeding it to model. But, in case of road scenarios objects like traffic signs cover a small portion of the image compared to other objects in the scene (approximately 5% of the image size). By resizing the input image to the square template, these small objects lose their structural features reducing the overall detection rate. To preserve the features of the small objects in the image, we maintained the aspect ratio of the template same as that of an input image. This helped in making the model more accurate.

The detector model was pretrained on Pascal VOC dataset [29], as pretraining on large generic datasets proved to help in better generalization.

The detector is trained on German Traffic Sign Detection Benchmark (GTSDB), and German Traffic Sign Recognition Benchmark (GTSRB) datasets [30]. GTSDB consists of 600 training and 300 testing sample images. GTSRB consists of 40,000 training and 10,000 testing image patches of traffic signs. Apart from using training samples provided by GTSDB, we have generated more sample images by pasting GTSRB patches on random noise canvas and used them for training the detector. This additional data helped in better localization and faster convergence.

While training we used data augmentation techniques like horizontal flip, color variation, brightness and contrast variations along with zoom-in and zoom-out operations. Data augmentation helped in making the detector model more accurate and robust to new environments. (same can be seen in the results section)

The detector network was trained with six scales (ranging from 0.05 to 0.5) and three aspect ratios (0.5, 1.0, 2.0) using Single Shot Multi-box Detector (SSD) method. The separable filter concept was utilized in detection network for faster feature generation.

Classifier Training

For the task of classification, we used 5 layered Le-Net [28] with two fully connected layers and a softmax layer. The network was trained on GTSRB dataset with augmentations like zoom-in and transparency variations, to make the model more accurate and robust. To decrease the class error, one extra class of “Not a Sign” is added, which helped in precision gain.

Embedded Optimization

The final frozen TSR network is still heavy and computationally intense to deploy on embedded processors with small memory footprint and processing capability. Thus, quantization becomes crucial for inference on low-power embedded platforms, which have a very limited budget for power and memory consumption. Such platforms often rely on fixed-point computational hardware blocks, such as Digital Signal Processor (DSP), to achieve higher power efficiency over float point processor, such as Graphics Processing Unit (GPU). The downside is that the quantizing core layers in the network causes large loss, and thus results in significant feature representation degradation in the 8-bit inference pipeline.

We used the Ristretto framework [20] developed by Gysel et al. to understand the impact on quantization loss. Ristretto framework internally analyses each layer for fixed-point (8-bit) implementation and gives an accuracy of the model for fixed-point implementation. During analysis, we observed a considerable drop in accuracy. Non-linear activation function (ReLU6) used in our network was the major root cause for quantization loss. ReLU6 encourages a model to learn sparse feature earlier. Clipping the signal at early layers will lead to quantization-unfriendly signal distribution, and thus largely decreases the SQNR of the layer output.

All the ReLU6 functions in our network are replaced with ReLU. Sub-net level localized retraining was performed to update convolution filter weights.

Hardware Specific Optimization and customization

We ported Traffic Sign Recognition (TSR) network to automotive grade embedded platforms from Texas Instruments

(TI) and Qualcomm. Network architecture is further customized to exploit underlying processor hardware acceleration, and to overcome implementation limitations.

Texas Instruments Platform

The setup consists of an evaluation platform made of TI's Jacinto TDA2x System-on-Chip (SoC). The TDA2x SoC incorporates a heterogeneous, scalable architecture that includes a mix of TI's TMS320C66x digital signal processor (DSP) generation cores, Vision AccelerationPac, ARM Cortex-A15 MPCore™, and dual-Cortex-M4 processors. TI's ecosystem offers,

- Caffe-Jacinto framework is a custom fork of Caffe that provides tools to train models with sparsity, resulting in low complexity models that can be used in embedded platforms.
- TI Device translation tool converts network models into an internal format best suited for use inside the TIDL library. Translation tool also converts model parameters (filter coefficients, bias) from floating-point to fixed-point values.
- Libraries for vision kernels on Vision AccelerationPac and DSP. One such library is the TI Deep learning Library (TIDL). TIDL is a suite of components that enables deep learning on TI embedded devices. TIDL has a highly optimized set of deep learning primitives that provide the best accuracy, speed, and memory usage trade-offs. TIDL is designed efficiently to take advantage of sparsity can run significantly faster by using such a model.

TSR network architecture is customized to overcome TIDL inference implementation limitations. For example, the TIDL Fully-Connected (FC) layer has limitations with respect to the number of input nodes supported. To overcome this limitation, the inputs to FC layers are sliced and processed by two smaller FC layers. The outputs of two FC layers were added element-wise to get the desired result. It's captured in Figure 4.

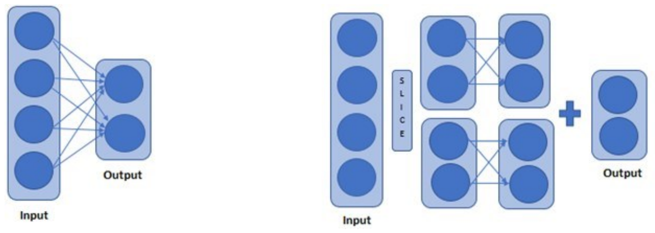


Figure 4. The above image illustrates how the fully connected (FC) layers were sliced and processed due to restrictions on the number of input nodes to the FC layer in embedded platforms.

Once the network architecture is frozen (customized network), network weights are further pruned by Using TI's Caffe-Jacinto framework. It is done using sparse re-training that consists of two steps: a) retraining the model with L1 regularization to reduce the standard deviation of the weights around zero mean and b) filtering the weights near to zero.

Post customizations and optimizations, we have a smaller model which gives a similar accuracy as the original model. The weights, activation and layer outputs of this sparsely trained model are quantized to 8-bit using TI Device translation tool. Quantization scheme used here is dynamic fixed point representation which further accelerates the processing. During inference execution,

- Feature extraction layers and SSD convolutional layers are executed from Vision AccelerationPac cores.
- Remaining SSD layers and classification layers are executed from DSPs.

Qualcomm Platform

The setup consists of an evaluation platform made of Qualcomm® Snapdragon™ 820A (automotive) processor [27]. The 820A SoC incorporates a heterogeneous, scalable architecture that includes a mix of CPU (Kryo Quad core), DSP (Hexagon™ 680 DSP) and GPU (Adreno™ 530 GPU). We used Snapdragon Neural Processing Engine (SNPE) [26] for running TSR inference on 820A platform. Snapdragon Neural Processing Engine (SNPE) is a Qualcomm Snapdragon software accelerated runtime for the execution of deep neural networks on CPU, DSP, and GPU.

Platform-specific network customizations not needed here, but TSR network has a layer which is not supported by SNPE inference. We added the layer implementation by using User Defined Layer (UDL) hooks provided by SNPE. SNPE workflow is captured in Figure 5.

For this porting activity, we used CPU and GPU cores of 820A processor. TSR network (With Embedded optimizations) is sliced in two parts. Part-1 network consists of Feature extraction layers from TSR network. Part-2 network consists of SSD and classification specific layers from TSR network. Part-1 network instance is executed from GPU and Part-2 instance is executed from CPU.

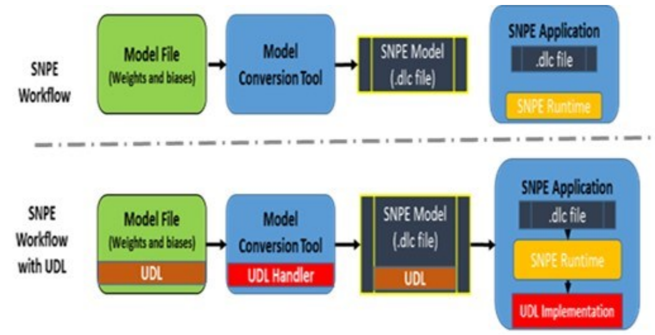


Figure 5. Illustrates SNPE workflow.
Ref. Snapdragon Neural Processing Engine SDK Reference Guide [26]

Experimental Results

A DNN based traffic sign recognition system that gives accuracy comparable to the state-of-the-art methods on standard datasets. It runs at real-time on Nvidia Tegra X2 and Qualcomm Snapdragon 820A processors.

Based on the embedded optimizations (weight pruning, quantization, and network customization) on top of the network pruning, the model size was reduced by a factor of 8x. The optimized model runs at real-time speed on low power SoC like Texas Instruments TDA2x utilizing DSP and vision accelerator. Refer to Table 2 for performance numbers. Performance numbers documented in Table 2 for Qualcomm Snapdragon 820A and TDA2x platforms are for 4 detections (Traffic Signs) in the given frame. The run-time numbers are subjected to vary with the number of detections. Figure 6 shows the precision-recall curves of the model trained with different combinations datasets and Figure 7 shows the precision-recall curves for the final optimized models on different hardware platforms. Results are generated on GTSDB test dataset.

Model complexity and performance: Table 2

Platform	Platform Computation power	Network	Network Model complexity	Run-Time Speed (milliseconds)
Tegra X2	~ 750 GFlops (GPU)	Network pruned Model	2.0 Giga Floating point operations per frame	42
Snapdragon 820A	~ 500 GFlops (GPU +CPU)	Network pruned + Embedded specific optimized Model	2.0 Giga Floating point operations per frame	48.5
TDA2x	~ 65 GMACs (2 DSP + 4 Vision Accelera-tionPac)	Network pruned + Embedded optimized model	0.25 Giga Multiply and accumulates per frame	49.5

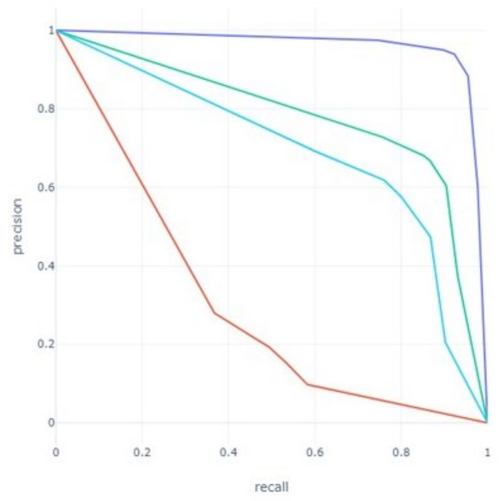


Figure 6. Precision recall curve for model trained with different datasets tested on GTSDB test dataset (Table-3)

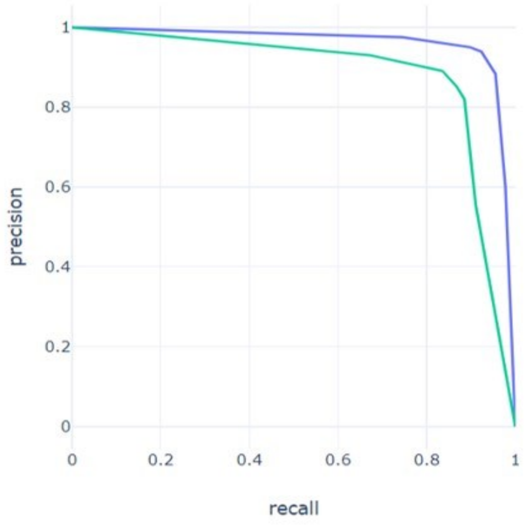


Figure 7. Precision recall curves for model before and after hardware specific optimizations tested on GTSDB test dataset (Table-4)

Model results with different training datasets (tested on GTSDB test dataset with 0.5 IoU): Table 3

Color	GTSDB Dataset	Augmentation	GTSRB patches	GTSRB patches stacked
🟥	✓			
🟦	✓	✓		
🟩	✓	✓	✓	
🟪	✓	✓		✓

Model results on hardware platforms (tested on GTSDB test dataset with 0.5 IoU): Table 4

Color	Platform	Inference speed
🟩	Texas Instruments TDA2x	20 fps
🟪	Qualcomm SDM 820	20 fps
🟪	NVIDIA Tegra X2	24 fps

Discussion and Conclusion

In the current Traffic Sign Recognition system explained, the two-stage model followed by local distillation of network reduce the computational complexity by maintaining the detection and classification accuracies.

Cascading arrangement of detection and classification networks eliminated the need for retraining of detection network that is relatively larger and computationally intensive. The smaller classification network can be quickly and efficiently retrained for any country-specific traffic sign set. This network arrangement also reduces the training cycles by large, compared to a composite network that trains on a combined loss function for both detection and classification.

While network pruning, locally retraining the network sub-blocks generated a very optimal network that had reduced computational complexity while maintaining the effectiveness of the same features.

Sparse re-training of the model further reduced the computational complexity, making it feasible to execute on embedded platforms like Texas Instruments TDA2x and Qualcomm 820A.

References

[1] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In NIPS, 2015.

[2] J. Dai, Y. Li, K. He, J. Sun, "R-FCN: Object detection via region-based fully convolutional networks", 2016.

[3] Erhan, D., Szegedy, C., Toshev, A., Anguelov, D.: Scalable object detection using deep neural networks. In: CVPR. 2014

[4] 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.

[5] Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: unified, real-time object detection. In: CVPR 2016

[6] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. L. Zitnick. Microsoft COCO: Common objects in ´ context. In ECCV. 2014

[7] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. arXiv preprint arXiv:1512.06473, 2015

[8] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. CoRR, abs/1504.04788, 2015

[9] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. CoRR, abs/1510.00149, 2, 2015.

[10] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. arXiv preprint arXiv:1405.3866, 2014.

[11] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. arXiv preprint arXiv:1412.6553, 2014.

[12] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531, 2015.

[13] A. 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.

[14] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167, 2015.

[15] Z. Zhu, D. Liang, S. Zhang, X. Huang, B. Li, and S. Hu. “Traffic-sign detection and classification in the wild”. In CVPR, pages 2110–2118, 2016.

[16] M.Mathew, K.Desappan, P.K.Swami, S.Nagori, "Sparse, Quantized, Full Frame CNN for Low Power Embedded Devices" IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 328-336, 2017.

[17] M.Mathew, K.Desappan, P.K.Swami, S.Nagori, B.M.Gopinath, "Embedded low-power deep learning with TIDL".

[18] P. Peng, Y. Mingyu and X. Weisheng, "Running 8-bit dynamic fixed-point convolutional neural network on low-cost ARM platforms," 2017 Chinese Automation Congress (CAC), Jinan, 2017, pp. 4564-4568.

[19] D. Williamson, "Dynamically scaled fixed point arithmetic," [1991] IEEE Pacific Rim Conference on Communications, Computers and Signal Processing Conference Proceedings, Victoria, BC, 1991, pp. 315-318 vol.1.

[20] P. Gysel, J. Pimentel, M. Motamedi and S. Ghiasi, "Ristretto: A Framework for Empirical Study of Resource-Efficient Inference in Convolutional Neural Networks," in IEEE Transactions on Neural Networks and Learning Systems, vol. 29, no. 11, pp. 5784-5789, Nov. 2018.

[21] Misha Denil, Babak Shakibi, Laurent Dinh, Marc' Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In Advances in Neural Information Processing Systems, pages 2148–2156. 2013.

[22] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In Advances in Neural Information Processing Systems, pages 1269–1277. 2014.

[23] Wei Wen, Chunpeng Wu, Yandan Wang, Learning Structured Sparsity in Deep Neural Networks, NIPS 2016.

[24] Tao Sheng, Chen Feng, Shaojie Zhuo, Xiaopeng Zhang, Liang Shen, Mickey Aleksic: A Quantization-Friendly Separable Convolution for MobileNets. CoRR abs/1803.08607 (2018).

[25] TDAx ADAS SoCs,http://www.ti.com/lsds/ti/processors/dsp/automotive_processors/tdax_adas_socs/overview.page.

[26] Snapdragon Neural Processing Engine SDK Reference Guide, https://developer.qualcomm.com/docs/snpe/index.html.

[27] Snapdragon 820 Automotive platform, https://www.qualcomm.com/products/snapdragon-820-automotive-platform

[28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[29] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes (VOC) Challenge. IJCV, pages 303–338, 2010.

[30] J. Stallkamp, M. Schlipsing, J. Salmen, C. Igel, "The German traffic sign recognition benchmark: a multi-class classification competition", Proc. IEEE IJCNN, pp. 1453-1460, 2011.

Author Biography

Raghav Nagpal *received his B.Tech degree in Instrumentation and Control Engineering from NIT Jalandhar, India. In his 5 years of work experience, Raghav has designed and developed algorithms for industrial automation, control, and perception. His research interest includes machine learning, deep learning, and Perception. Currently, Raghav is working as a Technical Lead at PathPartner Technology Pvt. Ltd.*

Chaitanya Krishna Paturu *received his Master of Technology degree in Communication Engineering from Vellore Institute of Technology, and Bachelor of Technology degree in Electronics & Communication from Jawaharlal Nehru Technological University, India. Currently he is a Technical Architect in PathPartner Technology Pvt. Ltd, working on machine learning, computer vision and deep learning.*

Vijaya Ragavan *received his BE degree in Electronics and Communication from University of Madras; and Masters in Electronics engineering from the Anna University, Chennai (2006). Since then he has worked in the Multimedia and ADAS Division at PathPartner Technology Pvt. Ltd. in, Bengaluru. His work was focused on the development of algorithms for ADAS, on low power heterogeneous systems.*

Navinprashath R R *holds a Bachelors from College of Engineering, Guindy, Chennai (2015). His interest lies in image signal processing pipelines and computational photography. He is currently with the camera team of Google. Previously Navinprashath was with digital imaging division at PathPartner Technology Pvt Ltd working on image signal pipelines and image enhancement (2015 - 2019)*

Radhesh Bhat *received his BE degree in Electronics and Communication from Visvesvaraya Technological University in 2004. Currently he is heading digital imaging division at PathPartner Technology Pvt. Ltd. His research interests include camera image signal processing pipeline, image quality, computer vision and deep learning.*

Dipanjan Ghosh *holds a Masters of Technology in Electronics & Communication from IIT Kharagpur, India and Bachelor of Technology in Electronics & Communication in North Eastern Regional Institute of Science and Technology India. He is co-founder and CTO of PathPartner Technology Pvt. Ltd.*

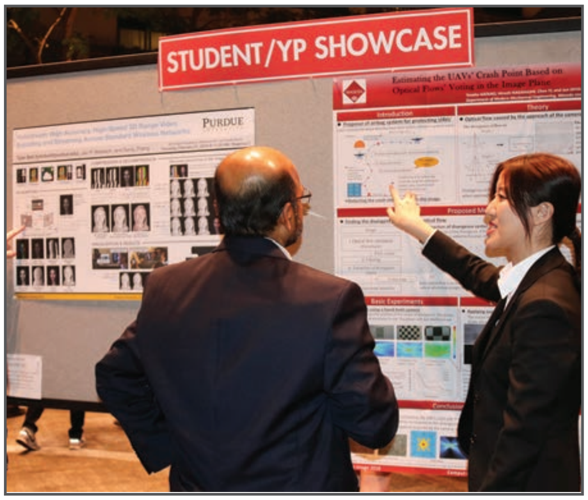
JOIN US AT THE NEXT EI!

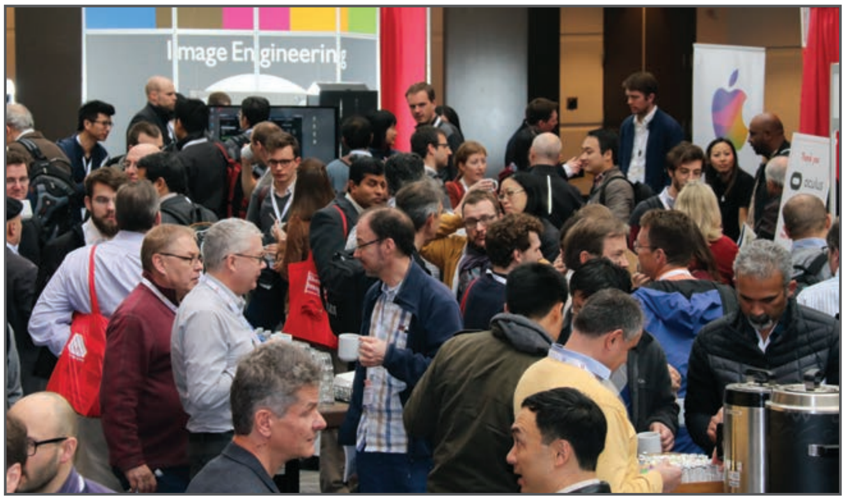
IS&T International Symposium on

Electronic Imaging

SCIENCE AND TECHNOLOGY

Imaging across applications . . . Where industry and academia meet!





AR is a fundamental and transformative interface and platform.
20+ years from my initial working demo to today's AR displays
Optimistic that it won't take another 20 to reach ubiquitous consumer AR!

• SHORT COURSES • EXHIBITS • DEMONSTRATION SESSION • PLENARY TALKS •
• INTERACTIVE PAPER SESSION • SPECIAL EVENTS • TECHNICAL SESSIONS •

www.electronicimaging.org

IS&T
imaging.org