EdgeEye - An Edge Service Framework for Real-time Intelligent Video Analytics

Peng Liu
University of Wisconsin-Madison
pengliu@cs.wisc.edu

Bozhao Qi
University of Wisconsin-Madison
bqi2@wisc.edu

Suman Banerjee
University of Wisconsin-Madison
suman@cs.wisc.edu

ABSTRACT

Deep learning with Deep Neural Networks (DNNs) can achieve much higher accuracy on many computer vision tasks than classic machine learning algorithms. Because of the high demand for both computation and storage resources, DNNs are often deployed in the cloud. Unfortunately, executing deep learning inference in the cloud, especially for real-time video analysis, often incurs high bandwidth consumption, high latency, reliability issues, and privacy concerns. Moving the DNNs close to the data source with an edge computing paradigm is a good approach to address those problems. The lack of an open source framework with a high-level API also complicates the deployment of deep learning-enabled service at the Internet edge. This paper presents EdgeEye, an edge-computing framework for real-time intelligent video analytics applications. EdgeEye provides a high-level, task-specific API for developers so that they can focus solely on application logic. EdgeEye does so by enabling developers to transform models trained with popular deep learning frameworks to deployable components with minimal effort. It leverages the optimized inference engines from industry to achieve the optimized inference performance and efficiency.

CCS CONCEPTS

• **Computer systems organization → Client-server architectures**;

KEYWORDS

Distributed System, Edge Computing, Computer Vision, Deep Learning

ACM Reference Format:
Peng Liu, Bozhao Qi, and Suman Banerjee. 2018. EdgeEye - An Edge Service Framework for Real-time Intelligent Video Analytics. In *EdgeSys '18: International Workshop on Edge Systems, Analytics and Networking, June 10–15, 2018, Munich, Germany*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3213344.3213345

1 INTRODUCTION

Deep learning with Deep Neural Networks (DNNs) is a hot topic in both academia and industry. It is widely used in different areas, such as speech recognition, language translation, image recognition, and recommendation. DNN-based approaches have achieved big improvement in accuracy over traditional machine learning algorithms on computer vision tasks. They even beat humans at the image recognition task [12]. In this paper, we present our work on building a real-time intelligent video analytics framework based on deep learning for services deployed at the Internet edge.

Nowadays, camera sensors are cheap and can be included in many devices. But the high computation and storage requirements of DNNs hinder their usefulness for local video processing applications in these low-cost devices. For example, the GoogLeNet model for image classification is larger than 20MB and requires about 1.5 billion multiply-add operations per inference per image [26]. At 500MB, the VGG16 model is even larger [25]. It is infeasible to deploy current DNNs into many devices with low-cost, low-power processors.

A commonly used approach to using deep learning in low-cost devices is offloading the tasks to the cloud. Deep learning has two parts: training and inference. Cloud computing fits the requirements of deep learning training very well because the training process requires a large amount of data and high throughput computing. However, the inference process, which is used in intelligent video analytics, requires low latency that cloud computing, in many cases, cannot provide. Also, continuously uploading high-bandwidth video data to cloud servers wastes network resources. If many people are uploading video data simultaneously, Internet congestion could occur. Moreover, if the network connection is interrupted, the services relying on cloud computing will be down. Finally, the data used in inference is generally more sensitive than the training data. Based on these observations, we believe cloud computing is not appropriate for live video analysis. It is ideal to analyze the live video stream locally if possible.

Edge computing is proposed to solve those problems in cloud computing. By deploying shared resources at the Internet edge and pushing computation close to the data sources, edge computing can benefit many applications requiring low latency and high privacy. We propose an edge service framework for real-time intelligent video analytics — EdgeEye. We name the framework EdgeEye because it is like the eyes of third-party edge services and devices. It helps them to ***see*** and ***understand*** the ambient environment through the video signal. When compared with cloud computing solutions, EdgeEye uses significantly less bandwidth because the system does not need to upload high bandwidth video to the cloud continuously. It also further reduces the total computation resource consumption because there is no need to encrypt the video when the analysis happens locally.

ParaDrop [15] is used in our work to manage the edge services using the EdgeEye API. ParaDrop is an edge computing platform

EdgeSys '18, June 10–15, 2018, Munich, Germany

ACM ISBN 978-1-4503-5837-8/18/06…$15.00
https://doi.org/10.1145/3213344.3213345

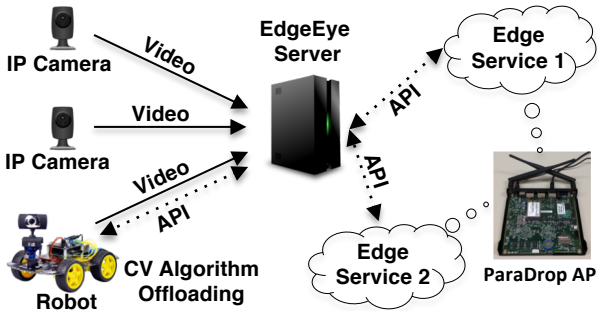

Figure 1: Overview of the EdgeEye deployment in a home environment. All the devices are on the same home network and can connect to each other directly. Two edge services and the robot can offload real-time video analysis tasks to the EdgeEye server using the EdgeEye API and get results with low latency.

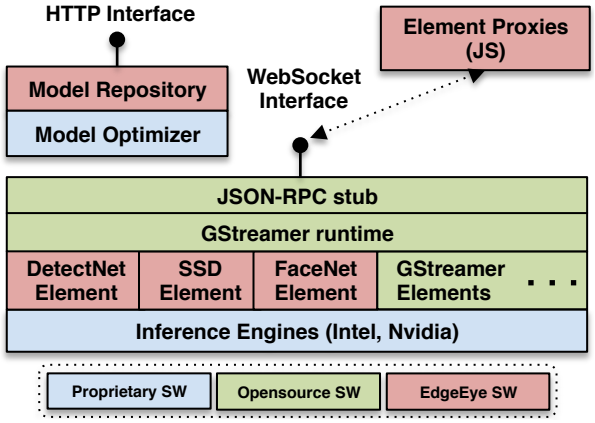

Figure 2: An overview of the EdgeEye software architecture. We extensively use open source components to build EdgeEye. The different colors indicate the different sources of the modules.

built with WiFi access points. It provides an API to manage the edge nodes as well as edge services running in the edge nodes. We can also use other edge computing platforms like resin.io [23] for the same purpose. The EdgeEye API provides a high level abstraction to hide the details of diverse machine learning frameworks like Caffe [14], TensorFlow [1], Torch [8], and MXNet [7]. The API covers different use cases, e.g., object detection, face verification, and activity detection. We implement the API with WebSocket protocol for low latency. EdgeEye uses a modular and extensible design, and we build the system with an open-source media framework — GStreamer [11]. We can easily reuse many open-source plugins of GStreamer to build complex pipelines.

Current research efforts on deep learning for computer vision focus more on training than inference. The inference performance will not be optimal if we directly use the deep learning frameworks to execute a DNN. Chip vendors like Intel [13] and Nvidia [20] provide optimized inference engines for CPUs and GPUs. EdgeEye leverages those implementations to implement highly efficient inference services.

Figure 1 gives a high-level overview of the EdgeEye in a home environment. Two example edge services are deployed on the ParaDrop access point (AP). The EdgeEye server has powerful CPU and GPU to analyze live video streams in real-time. The IP cameras in this figure can be standalone wireless cameras or cameras of other devices, e.g., baby monitors. Figure 1 also shows a robot with a camera. With EdgeEye's capability of real-time video analytics, the robot can see and understand the environment and then make decisions accordingly. The cost to build the robot will be reduced because it does not need to have a high-performance processor. The edge services and the robot use the EdgeEye API to manage video analysis sessions with given video sources. They specify parameters and deep learning models to be used by the analysis session. And the analysis results will be sent back to them in real-time.

Contributions: i) We propose an edge service framework for real-time video analytics applications — EdgeEye, which provides a high-level abstraction of some important video analysis functions based on DNNs. Applications can easily offload the live video analytics tasks to the EdgeEye server using its API, instead of using deep learning framework specific APIs. ii) We build a prototype of EdgeEye with Caffe and Nvidia's TensorRT. Our preliminary evaluations indicate that EdgeEye provides higher inference performance than using the deep learning frameworks directly. iii) This paper presents an example application built with EdgeEye to show the capabilities of the framework.

2 SYSTEM DESIGN

Usually, developers build deep learning applications with various open source deep learning frameworks, e.g., Caffe, TensorFlow, and MXNet. These frameworks provide a wide variety of APIs and language bindings. We believe that providing uniform task-specific interfaces will simplify developers' work. Computer vision related tasks always include some supportive tasks, such as image input/output, decoding, image rescaling, and format conversion. Application frameworks need to provide such primitives with optimized performance, so that developers need not integrate and optimize other third-party libraries for these supportive tasks.

2.1 EdgeEye Architecture

Figure 2 gives a high level overview of the EdgeEye framework. EdgeEye needs to be installed on a machine with high-performance CPU and GPU to guarantee real-time processing. It adopts the GStreamer framework to manage the video analysis pipeline and reuses open source elements to handle some supportive tasks, such as format conversion and rescaling. The EdgeEye elements are implemented based on the inference engines from chip vendors for the best possible performance and efficiency. Along with the GStreamer pipeline architecture, EdgeEye provides a WebSocket API based on JSON-RPC and also element proxies to control EdgeEye elements through the API. EdgeEye also provides an HTTP API to manage deep learning models used by EdgeEye elements. The model repository integrates the model optimizers, which profile the deep learning models from deep learning frameworks, such as

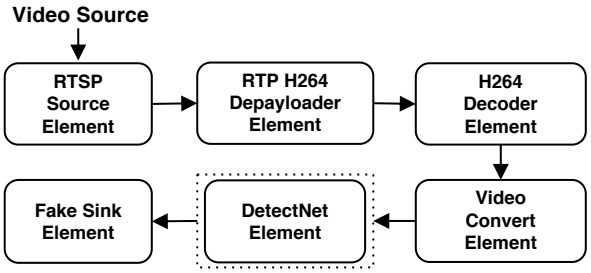

Figure 3: An object detection pipeline including the DetectNet element.

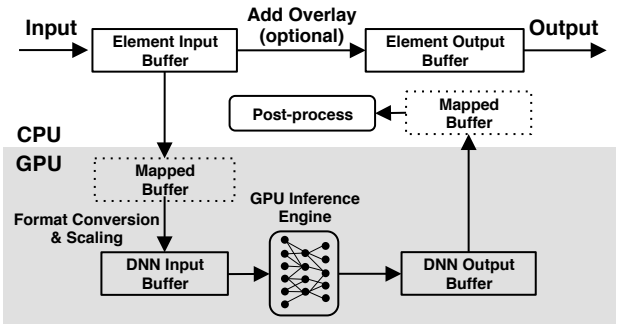

Figure 4: Internal of the DetectNet element.

Caffe and TensorFlow, and generate the deployable files that the inference engines can load and execute directly.

2.2 DNN Model Management

Deep learning models store the structure, weights, and biases of DNNs. These models are often tens or even hundreds of megabytes in size. Different deep learning frameworks store them in different formats. For example, a deep learning model built with Caffe has a *.prototxt file and a *.caffemodel file, while the same model built with TensorFlow has one *.pb file. The model management module manages the models used by the EdgeEye GStreamer elements. When applications upload a new model to the model repository, the optimizer will profile the model and generate the optimized deployable model file for the corresponding EdgeEye element.

2.3 Video Analysis Pipeline

We use the GStreamer media framework to implement the video analysis pipeline. A pipeline is a chain of elements. Every element besides the source element processes video data received from the element before it. It then either forwards the processed data to the next element in the pipeline or discards the data. The pipeline and plugin mechanisms available in GStreamer simplify the creation of complex media processing pipelines. Figure 3 shows an example pipeline to detect objects with DetectNet neural network. The non-DetectNet elements are from the GStreamer community and handle data reading, parsing, and decoding. The DetectNet element (inside the dashed box) executes deep learning models in the Nvidia GPU inference engine using the TensorRT library [20].

2.4 EdgeEye Element Design

EdgeEye elements are model and inference engine specific, i.e., we have SSD-INTEL, SSD-NVIDIA, YOLO-INTEL, and YOLO-NVIDIA elements for SSD (Single Shot multi-box Detector) [16] and YOLO [22] DNNs running on the Intel and Nvidia inference engines. EdgeEye hides the difference of these DNN models in network structure and input/output format. Based on the message mechanism of GStreamer, EdgeEye exposes the functionality through a task-specific interface, so that developers can get the video analysis functionality with minimal effort. Developers need to specify the model files (network structure, weights, and biases) to load into the inference engine, as well as specify other model-specific input and output parameters. The output of the DNN model can either be embedded in the video stream (overlay), or sent to pipeline manager through messages. These messages can then be forwarded to applications through the JSON-RPC interface.

DetectNet is an example DNN architecture in the Nvidia Deep Learning GPU Training System (DIGITS) [19]. Its data representation is inspired by YOLO. DetectNet uses GoogLeNet internally with some modifications. Similar to YOLO, it overlays an image with a regular grid. The grid has spacing slightly smaller than the smallest object we wish to detect, and the DNN generates information for every grid square. The method to generate output bounding boxes and class labels is totally different from YOLO. As shown in Figure 4, we have to execute the post-processing steps including bounding box clustering to generate the final result. More detailed information is available in [17]. Figure 4 also shows the buffer management in the element implementation. We use Nvidia CUDA mapped memory feature to simplify the memory movement between CPU and GPU. The DetectNet element uses TensorRT to control the GPU Inference engine, such as initialize the inference engine, create inference runtime, load the serialized model, create inference execution context, and execute the inference engine.

The hardest parts of an element implementation are in the buffer management and post-processing, both of which are on the CPU side. If TensorRT supports all the layers of a DNN model, we can easily load the serialized model and create the execution context. Unfortunately, TensorRT does not support some layers, e.g., Leaky ReLU, which is used in YOLO. We use the TensorRT plugin API to implement those unsupported layers ourselves.

All EdgeEye elements share a similar structure. Aided by the GStreamer plugin architecture, we can easily integrate DNNs into GStreamer pipelines for different tasks by developing EdgeEye elements for them.

2.5 EdgeEye API

Through the JSON-RPC based API, applications can control both the pipeline and the individual EdgeEye elements. The API is task-specific. For example, the interface for object detection requires an input parameter to specify the DNN model used by the EdgeEye element, and the output is the detection results including bounding boxes and confidence level of the detection. The interface is independent of the underlying implementation of EdgeEye elements. Application developers does not need to care about the

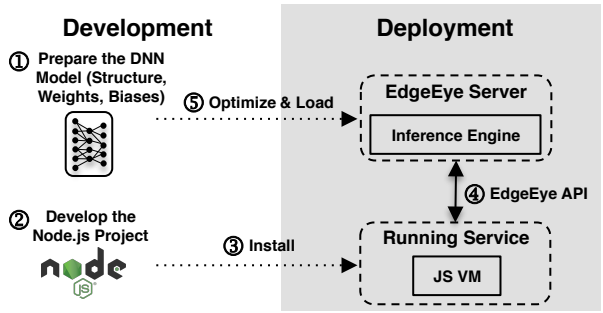

Figure 5: An example service powered by EdgeEye.

deep learning framework (Caffe, TensorFlow, etc.) and model (DetectNet, YOLO, SSD, etc.) used to implement the object detection function. EdgeEye provides a Javascript wrapper for those interfaces. Javascript is widely used to develop IoT applications, even for resource-constrained devices [5, 24]. The following code snippet shows how to use the Javascript wrapper to configure a DetectNet element to detect dogs. We ignore the code to load the deep learning model file and construct the pipeline. The application creates a DetectNet element instance through the handle of the pipeline. If succeeds, it sets an event handler to receive detection results.

```
pipeline.create('DetectNet', {
  model: dogDetectorModel
}).then(element => {
  element.on('objectDetected', event => {
    // work with the detection result
    // (event.boundingBoxes, event.confidence)
  });
});
```

2.6 Work with EdgeEye

We can leverage EdgeEye from either edge services or standalone devices. This paper only discusses edge services. Figure 5 illustrates the steps to develop an EdgeEye powered edge service. First, we need to design and train the deep learning model with the deep learning framework we choose. Next, as usual, we develop the edge service logic with Javascript. The Javascript code will be combined with the trained deep learning model and other assets to build a deployable edge service. In the deployment phase, users install the service into the edge computing platform, which uses the EdgeEye API to upload the model to the EdgeEye server and starts the execution of the deep learning inference engine.

3 IMPLEMENTATION

We have prototyped the EdgeEye on a desktop with Ubuntu 16.04 LTS. The desktop has an Intel i7-6700 CPU @ 3.40GHz, a Nvidia GeForce GTX 1060 6GB, and 24GB system RAM. We installed TensorRT 3.0.4 and CUDA 9.0 on the desktop.

We used Kurento [9] to implement the GStreamer pipeline management and JSON-RPC interface. Kurento is an open source WebRTC [4] media server. It provides the tools to generate the stubs on both server and client side for the JSON-RPC based API. Although its main goal is to support WebRTC related features and it provides many unrelated features for EdgeEye, its modular design can be reused to implement an edge computing framework. We use Node.js to implement the model repository and the HTTP interface.

We have implemented the DetectNet element for object detection with C/C++ based on TensorRT and CUDA and are working on the SSD and YOLO elements. The current prototype only uses Nvidia's GPU inference engine, but we will develop elements using Intel's inference engines as well. We also plan to develop elements for other purposes, such as semantic segmentation and activity detection.

TensorRT provides several optimizations for the GPU inference engine including layer & tensor fusion, FP16 and INT8 precision calibration, and kernel auto-tuning. These optimizations improve the inference performance significantly. Therefore, by leveraging these optimizations in TensorRT, EdgeEye can achieve higher inference performance than Caffe with the same DNN model. We evaluated the performance of the DetectNet element with a dog detector model. The FP16 optimization of the TensorRT model optimizer was enabled in the test, and we used two video files with the same content (dogs), encoding (H.264) and length (2 minutes, 29.97 frames per second), but different resolutions (1280x720 and 640x360 respectively). We ran each test 10 times and averaged the results. For comparison purpose, we also tested Caffe (20180403-snapshot from Github [6]) with the same DNN model to get the average forward pass time. The forward pass time is the time taken for a batch of data to flow from the input layer of the network to the output layer of the network. Based on the average forward pass time, we calculated the FPS (Frames Per Second) values for Caffe framework. These values are the theoretical upper bound performance we can achieve with Caffe, because we omit overheads related to video decoding, buffer movement, and video scaling in the test. Three Caffe configurations were used in the test: i) the master branch of Caffe executing with GPU, ii) the master branch of Caffe executing without GPU and no optimization for CPU, iii) the Intel branch of Caffe executing with optimizations for Intel CPUs. We used batch size = 1 for all the tests because we want low latency.

Table 1: Inference Speed Test Results

Configuration	Mean FPS
1. EdgeEye (video: 1280x720)	**55**
2. EdgeEye (video: 640x360)	**76**
3. Caffe (GPU, CUDA 9.0)	54
4. Caffe (CPU, w/o optimization)	0.49
5. Caffe (CPU, w/ optimization)	8.6

Table 1 shows the results. We can see EdgeEye gets the highest FPS because of the optimizations in TensorRT, even though it needs to do a lot of extra work. In conclusion, GPU acceleration is very important to get high-performance inference, and the optimized inference engine (TensorRT) can further boost the performance. Inference accuracy data is not provided here because we use Nvidia's optimizer for TensorRT directly. Nvidia claims their GPUs can deliver massive performance improvements with the near-zero loss in accuracy [21].

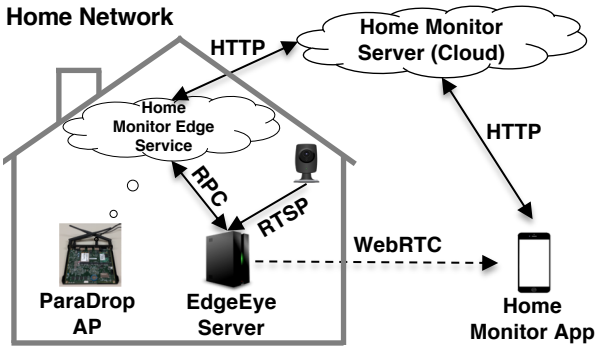

Figure 6: An example application built with EdgeEye.

4 EXAMPLE APPLICATION

In this section, we introduce the design and implementation of an EdgeEye powered application — *home monitor*. Home monitor is built for the ParaDrop platform. It can also be deployed in other edge computing platforms with minor modifications.

As shown in Figure 6, the application has three major components: an edge service deployed in a ParaDrop AP, a cloud server, and a mobile app.

The edge service uses the EdgeEye API to orchestrate the video analysis tasks and receive analysis results. The video stream is received from an IP camera with RTSP protocol support. The edge service also connects to the cloud server for WebRTC signaling. With the WebRTC support, users can define rules that edge service follows to set up a P2P connection to the mobile app and stream real-time video as well as analysis results to users with minimum latency.

The most important responsibility of the cloud server is WebRTC signaling. Other responsibilities include account management and rules management.

We develop the mobile app with the React-native framework. With the mobile app, we can receive real-time notification from the edge service, and view the real-time video stream from cameras remotely. Users can also specify some rules about the detection results.

The home monitor application currently only supports object detection. With other EdgeEye elements, we can easily support more use cases. For example, we can develop a face verification element to support a smart lock application. With EdgeEye framework, video streams do not need to leave the home network for analysis. This approach saves bandwidth, provides reliable and low latency service, and also enables high privacy protection.

5 DISCUSSION

Chip vendors, IP vendors, tech giants, and startups are building and shipping specialized processors for deep learning applications [27]. Some companies are also building products equipped with these kinds of specialized processors. For example, at the AWS re:Invent global summit 2017, Amazon announced DeepLens, a deep learning enabled wireless video camera for developers [2]. Google Clips is another wireless smart camera with machine learning enabled [10]. The advance in specialized processors can help EdgeEye to improve its performance. Unlike Amazon DeepLens and Google Clips, EdgeEye's resources are shared by multiple smart devices. In addition, EdgeEye's flexibility and multi-tenancy enable developers to try new algorithms easily at the edge.

The development of EdgeEye is still ongoing. Current work focuses on Convolutional Neural Networks (CNNs). We plan to support activities analysis in video streams through Recurrent Neural Networks (RNNs).

For now, we only work on computer vision applications in the smart home scenario, but EdgeEye's architecture is also useful for other scenarios, such as city deployment and workplaces.

6 RELATED WORK

Using edge computing for live video analytics is a hot research topic. Ananthanarayanan et al. discuss the performance requirements of a wide range of video analytics applications, such as traffic, self-driving and smart cars, personal digital assistants, surveillance, and security. They conclude that a geographically distributed architecture of public clouds and edges is the only feasible approach to meeting the strict real-time requirements of large-scale live video analytics [3]. Zhang et al. introduce VideoStorm, a video analytics system that processes thousands of video analytics queries on live video streams over large clusters [28]. Zhang et al. present Vigil, a real-time distributed wireless surveillance system that leverages edge computing to support real-time tracking and surveillance [29]. Not targeting any specific application, EdgeEye is a generic edge computing framework for real-time video analytics. Its goals are usability, flexibility, and high efficiency.

Many research efforts on DNNs primarily focus on improving the training speed and inference accuracy. While application developers care more about inference speed and easy to use interfaces. Zhao et al. propose a system called Zoo [30] to migrate this gap. They build Zoo based on a numerical computing system written in OCaml language. EdgeEye shares the similar goal with Zoo, but we build the system based on available work from industry and open source community.

Nvidia provides an inference framework - DeepStream SDK [18]. DeepStream simplifies the development of video analytics applications on Nvidia platforms by providing TensorRT, hardware video codecs, and other related primitives into an optimized API. EdgeEye provides a higher level abstraction of the inference engine than DeepStream. In addition, it provides necessary functions for edge service development and deployment.

7 CONCLUSION

In this paper, we present EdgeEye, a flexible, extensible and efficient edge computing framework to develop real-time video analytics applications. EdgeEye has a modular design based on GStreamer media framework, and it provides tools to deploy and execute DNN models in an easy and efficient way. By providing a high-level API, EdgeEye simplifies the development and deployment of deep learning based video analytics applications. We discuss the motivations to design such a framework, and introduce its implementation. We also present an example application built with the EdgeEye framework to show its capability and extensibility. EdgeEye is an ongoing

work, we are working on more elements for different computer vision tasks based on different inference engines.

8 ACKNOWLEDGMENTS

We are grateful to our shepherd, Dr Richard Mortier, and the anonymous reviewers whose comments helped bring the paper to its final form. All authors are supported in part by the US National Science Foundation through awards CNS-1345293, CNS-14055667, CNS-1525586, CNS-1555426, CNS-1629833, CNS-1647152 and CNS-1719336.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A System for Large-Scale Machine Learning.. In *OSDI*, Vol. 16. 265–283.

[2] Amazon. 2018. The world's first deep learning enabled video camera for developers. Retrieved March 29, 2018 from https://aws.amazon.com/deeplens/

[3] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. 2017. Real-Time Video Analytics: The Killer App for Edge Computing. *Computer* 50, 10 (2017), 58–67.

[4] Adam Bergkvist, Daniel C Burnett, Cullen Jennings, Anant Narayanan, and Bernard Aboba. 2012. Webrtc 1.0: Real-time communication between browsers. *Working draft, W3C* 91 (2012).

[5] Bocoup. 2018. Johnny-Five: The JavaScript Robotics and IoT Platform. Retrieved March 29, 2018 from http://johnny-five.io/

[6] BVLC. 2018. Caffe: a fast open framework for deep learning. Retrieved March 29, 2018 from https://github.com/BVLC/caffe

[7] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. 2015. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274* (2015).

[8] Ronan Collobert, Samy Bengio, and Johnny Mariéthoz. 2002. *Torch: a modular machine learning software library.* Technical Report. Idiap.

[9] Luis López Fernández, Miguel París Díaz, Raúl Benítez Mejías, Francisco Javier López, and José Antonio Santos. 2013. Kurento: a media server technology for convergent WWW/mobile real-time multimedia communications supporting WebRTC. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2013 IEEE 14th International Symposium and Workshops on a.* IEEE, 1–6.

[10] Google. 2017. Google Clips. Retrieved March 29, 2018 from https://store.google.com/us/product/google_clips?hl=en-US

[11] GStreamer. 2018. GStreamer: open source multimedia framework. Retrieved March 26, 2018 from https://gstreamer.freedesktop.org/

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision.* 1026–1034.

[13] Intel. 2017. Intel's Deep Learning Inference Engine Developer Guide. Retrieved March 26, 2018 from https://software.intel.com/en-us/inference-engine-devguide

[14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia.* ACM, 675–678.

[15] Peng Liu, Dale Willis, and Suman Banerjee. 2016. Paradrop: Enabling lightweight multi-tenancy at the network's extreme edge. In *Edge Computing (SEC), IEEE/ACM Symposium on.* IEEE, 1–13.

[16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. Ssd: Single shot multibox detector. In *European conference on computer vision.* Springer, 21–37.

[17] Nvidia. 2016. DetectNet: Deep Neural Network for Object Detection in DIGITS. Retrieved March 28, 2018 from https://devblogs.nvidia.com/detectnet-deep-neural-network-object-detection-digits/

[18] Nvidia. 2018. NVIDIA DeepStream SDK. Retrieved March 27, 2018 from https://developer.nvidia.com/deepstream-sdk

[19] Nvidia. 2018. NVIDIA DIGITS, Interactive Deep Learning GPU Training System. Retrieved March 28, 2018 from https://developer.nvidia.com/digits

[20] NVIDIA. 2018. NVIDIA TensorRT - Programmable Inference Accelerator. Retrieved March 26, 2018 from https://developer.nvidia.com/tensorrt

[21] NVIDIA. 2018. TECHNICAL OVERVIEW: NVIDIA DEEP LEARNING PLATFORM. Retrieved March 29, 2018 from https://images.nvidia.com/content/pdf/inference-technical-overview.pdf

[22] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 779–788.

[23] resin.io. 2018. Resin.io homepage. Retrieved March 26, 2018 from https://resin.io/

[24] Samsung. 2018. IoT.js - A framework for Internet of Things. Retrieved March 29, 2018 from http://iotjs.net/

[25] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[26] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. 2015. Going deeper with convolutions. Cvpr.

[27] Shan Tang. 2018. A list of ICs and IPs for AI, Machine Learning and Deep Learning. Retrieved March 29, 2018 from https://basicmi.github.io/Deep-Learning-Processor-List/

[28] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance.. In *NSDI*, Vol. 9. 1.

[29] Tan Zhang, Aakanksha Chowdhery, Paramvir Victor Bahl, Kyle Jamieson, and Suman Banerjee. 2015. The design and implementation of a wireless video surveillance system. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking.* ACM, 426–438.

[30] Jianxin Zhao, Richard Mortier, Jon Crowcroft, and Liang Wang. 2017. User-centric Composable Services: A New Generation of Personal Data Analytics. *arXiv preprint arXiv:1710.09027* (2017).