Towards a Dynamic Edge AI Framework applied to autonomous driving cars

G. Muratore[2], J. A. Rincon[1], V. Julian[1], C. Carrascosa[1], G. Greco[2],G. Fortino[3]

[1] Universitat Politècnica de València. VRAIN. Valencian Research Institute for Artificial Intelligence {jrincon, vinglada, carrasco@dsic.upv.es}
[2] Department of Mathematics and Computer Science (DeMaCS), University of Calabria, Via P. Bucci, 87036 Rende (CS), - Italy
{mrtgpp92l01l063v@studenti.unical.it, gianluigi.greco@unical.it }
[3] Department of Informatics, Modeling, Electronics and Systems (DIMES) University of Calabria, Via P. Bucci, 87036 Rende (CS), Italy
{g.fortino@unical.it}

Keywords: Autonomous Driving, Internet of Things, Edge AI, Intelligent Agents

Abstract. This work proposes an innovative solution in the field of Edge AI in order to efficiently exploit new hardware components available on the market at low cost. Edge AI means that algorithms are processed locally on a hardware device. The algorithms use data (sensor data or signals) that are created on the own device. The idea of this paper focuses on demonstrating the validity of the proposed solution by implementing an autonomous driving system that exploits communication between intelligent agents. In this case, our self-driving cars are equipped with a low-cost device that allows you to recognise objects along the way and consequently take actions by running a machine learning model. The presence of a machine learning model also allows the developer to modify it by extending the flexibility and application possibilities of the proposed solution.

1 Introduction

In the last decade we have been able to observe how Artificial Intelligence (AI) has advanced continuously, playing an important role in different areas of knowledge such as medicine [1], robotics [2], or autonomous cars [3], among others. In all these applications were necessary large computing units, high-performance servers, capable of executing millions of calculations per second. Some examples are Cloud services such as *Amazon Web Services (AWS)*[4] (that introduced its *Elastic Compute Cloud* and was the first public Cloud Computing service available), *Microsoft Azure*[5] (that was announced as *Azure*) and *Google Cloud*

[4] https://aws.amazon.com/es/
[5] https://azure.microsoft.com/es-es/

Platform[6]. However, so much computing power has some drawbacks, such as the space they need and their high energy consumption.

At the same time as AI evolved, electronics and microelectronics evolved as well. This evolution has made it possible to create smaller and more powerful devices capable of accessing sensors, actuators and being connected to the Internet. This is what we now call the Internet of Things (IoT) [4] [5], which allows almost any everyday element (refrigerators, lights, televisions, etc.) to be connected to the Internet. However, there are still some cases in which complex processes are required in which it is necessary to use some AI tools. Therefore, these devices are seen in the need to use the cloud as an AI computing tool and return to the device the results obtained. This process in some cases is too costly, due to the time required to send such data. It is for this reason that some of these devices have evolved to what we know today as EDGE Computing [6], [7], [5], [8]. Edge Computing allows all data generated by IoT devices to be processed where they have been generated, thus avoiding sending data from point A (fridge) to a farther B point (cloud or data centre). This means that the refrigerator collects and processes the data at its place of origin, thus avoiding the massive sending of information to the cloud. According to Cisco, some 50 billion IoT devices will be connected by 2020, but all these volumes of data are passive if they cannot be analysed or interpreted.

This data processing in most cases, are used to perform simple actions on the systems. However, some of these actions could be performed within each of the devices, to do this it is necessary that these devices have the ability to use AI techniques such as machine learning models. These models would help detect patterns in the lower layers of the system, thus avoiding the massive sending of information to higher layers. This would decrease response times, as well as, the massive sending of information to the same point.

These devices capable of performing these actions at a low level is what we know today as *Edge AI*. Edge AI enables the creation of intelligent solutions in real-time using deep learning techniques. These solutions must have a number of key features, such as energy efficiency, low-cost and a balance between precision and energy consumption. Currently, deep learning techniques are conventionally deployed in centralised computing environments. However, these applications have some limitations such as costs generated by energy consumption, and latency in the network due to massive data sending. To address these limitations, Edge Computing, often referred to as *Artificial Edge Intelligence*, has been introduced in which calculations are performed locally from data acquired from various devices or sensors. The challenges of meeting the requirements for implementing Edge AI are to ensure high accuracy of the algorithms while having low power consumption. However, this would not be possible without the latest hardware innovations, including central processing units (CPUs), graphics processing units (GPUs), application-specific integrated circuits (ASICs) and system-on-a-chip (SoC) accelerators, which have made Edge AI possible. Thanks to these

[6] https://cloud.google.com/

advances there are applications such as the one presented by [9], in which they detect apples in real-time using Edge AI or Smart Parking using Edge AI [10].

This article presents a tool based on Edge AI, which incorporates deep learning techniques, allowing the user to modify the models online.

2 System Description

This section describes the framework developed, which enables interaction between intelligent entities and Edge AI devices. This framework provides the developer with a series of tools that allow him to train models of deep learning, dynamic change of learned models and communication between intelligent entities. One of the most important characteristics of the framework presented in this article is the ability to dynamically modify learned models. Allowing the developer to deploy the system anywhere and through a WiFi network send the new model. This dynamism makes this framework an ideal tool for applications in smart homes, smart cities, health care, among others. Another important feature is the ability to be used in different low cost and energy consumption devices such as Rasberry Pi [7], Beagle Bone [8] and any device that incorporates Linux.

The Figure 1 shows the diagram of the proposed framework. Which incorporates two types of intelligent entities or agents. The first is the edge agent, this agent is located within the environment, then we have the manager agent, this agent is in charge of the actions at high level as the training of the models. Each of the agents presented in this framework was carried out using the SPADE platform and these are explained below.

2.1 Manager Agent

The manager agent is the one in charge of performing the actions at a high level, that is, he is in charge of performing the tasks that require high computational performance. Such as the training of the different deep learning models and the transmission of those models to the Edge Agents. This agent is composed by a two-state machine (figure 2, in the first state it takes a set of data to learn.

It is in this state, where the agent performs the image pre-processing tasks, the resizing and the division of the data set in three: training, validation and testing. Once this is done, the agent starts the training process of the Mobilenet network, the result of this training is a model. To which a series of transformations must be made, so that it can be embedded inside the Edge device. These transformations consist in converting from a *.h5 file to a *.kmodel file. For this it is necessary to perform intermediate transformations such as *.h5 to *.pb and then from *.pb to *.tflite and this last one to *.kmodel as can be seen in the Figure 3

[7] https://www.raspberrypi.org/
[8] https://beagleboard.org/bone

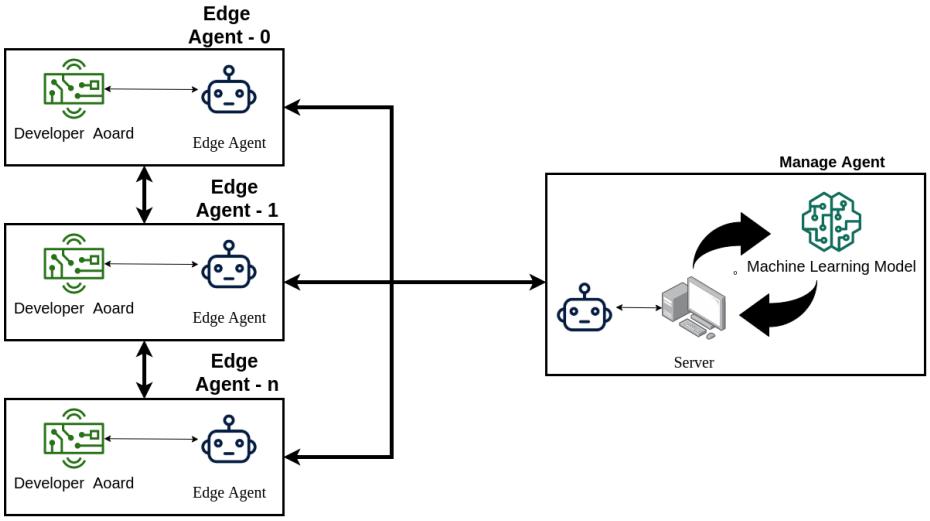

Fig. 1: Framework Diagram

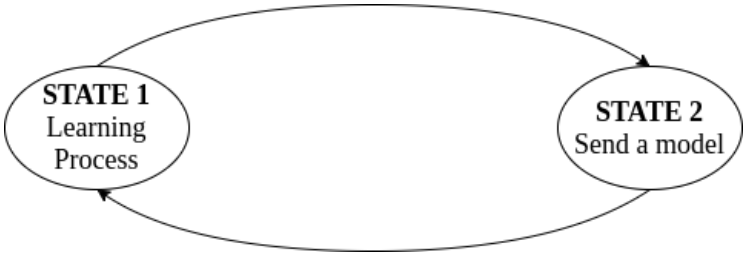

Fig. 2: State machine of Manager Agent.

Once the model is obtained the agent manager goes to state two, in this state the agent waits for a request from the edge agents. If in this state the agent receives a request from an edge agent, it sends the new model to the agent.

2.2 Edge Agent

The Edge agent is located within the environment, it is in charge of interacting with the environment. This interaction is done using sensors or actuators, which are connected to the system. The sensors allow the agent to introduce information to the machine learning model. the results provided by the model are used by the actuators to interact in the environment.

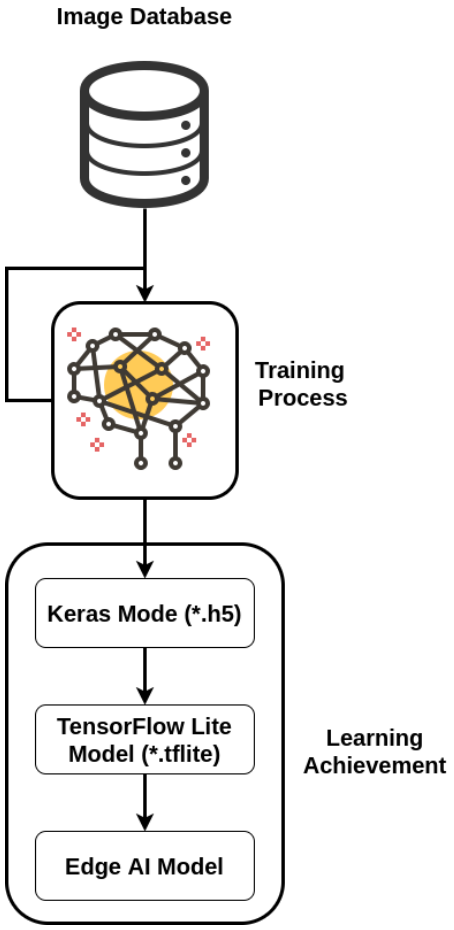

Fig. 3: Process to get the learning achievement.

This agent is composed by a state machine with two states (Figure 4). The first state is the main process, in this state the agent executes the actions. These actions will be determined by the application in which it is being used. It is in this state where the agent is perceiving the information of the environment, using the different sensors connected to it. This information is then used by the machine learning model, the result obtained from this model will serve the agent to interact with the environment.

The second state of the edge agent is in charge of making the requests to the agent manager, in order to know if there is a new model available. If there is a new model, the Edge Agent receives a message containing a URL. This URL allows the agent to download the model and the labels associated to this

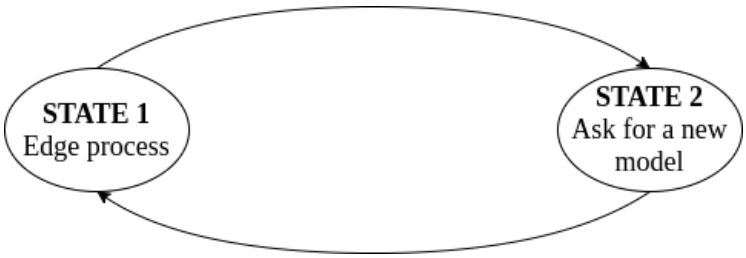

Fig. 4: Edge Agent State Machine

model. Once the template has been downloaded, Edge Agent can now use the new template.

3 Case study

This section describes a case study, in which we will use the system proposed above. The case study focuses on the use of autonomous cars, capable of dynamically modifying the learning models. This dynamism will allow the car to identify objects within the road, acting according to the results delivered by the model.

The Edge agents are built using a Raspberry Pi 3 - Model B and a Grove AI Hat for Edge Computing. Each car is equipped with 2 different cameras one for the Raspberry Pi and the other for the Edge device. The camera used by the Raspberry is in charge of performing the autonomous driving, performing a line tracking. The second camera is in charge of classifying the objects within the environment and sends the processed information to the Raspberry. This information contains the probabilities of the objects to be classified, extracting the maximum probability and associated index. once these two elements are in place the Raspberry can know which object is being classified and acts according to the programmed behavior. This action can be to completely stop the car, reduce the speed or increase the speed.

The figure shows in more detail the main objective of the case study. In this case we have an edge agent represented by a car, which has as its initial model the detection of a horse on the road. Once the edge device detects the horse, it sends the information to the Raspberri and this causes the car to stop. spend a time (t), the model changes and now the object to be classified is an other car, once Edge device detects the police car the Edge Agent reduces its speed.

To train the neural network of the proposal presented, the first thing is to build a dataset. In a first approach it was decided to perform this task automatically, our system will be given the class names and the system will download the images from Google. After several tests, we decided to give up this idea because too many images were not suitable for classification tasks and the network

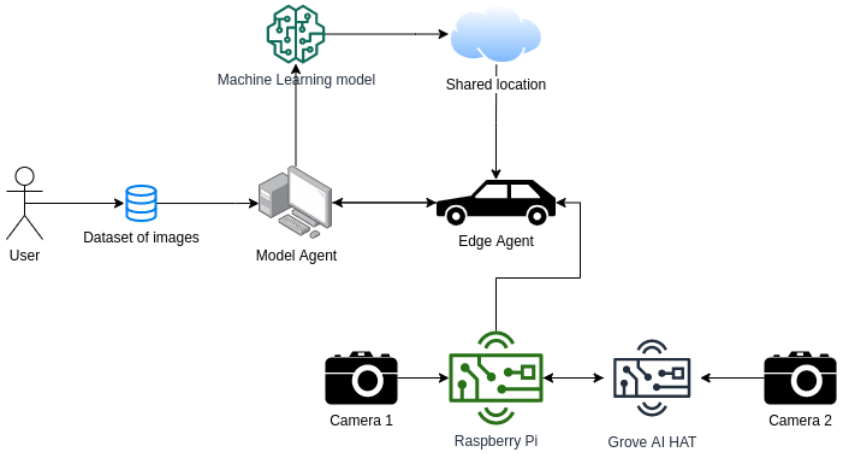

Fig. 5: Case study diagram

performance was very low. For this reason, we decided to use part of the WSID-100 [11] data set. The Manage Agent uses the data set to train the network, to perform such training the agnete manager divides the data set into a proportion of 80% for training, 10% for testing and 10% for validation. As a result of this training process, a machine learning model and a firmware suitable for cars are obtained. Once you have the trained model, the Agent manager loads this model into a shared resource so that it is available for Edge Agents (cars).

The Figure 7 shows the accuracy of our training process, it can be seen that in these two graphs the accuracy is very high, which indicates that the classification process is good.

After the evaluation of the model the agent starts a series of transformation on it to make it suitable for EdgeAI devices. In particular the first transformation convert our *H5* model in a *TensorFlow Lite* model [12]. However this conversion is not yet sufficient in fact the agent has to convert the obtained *TensorFlow Lite* model in a *Kmodel* file through *nncase*[9], a neural network compiler for Kendryte K210 AI accelerators[10]. To flash the model in an EdgeAI device we need also a custom firmware. The agent create the firmware *bin* file exploiting the Kendryte K210 standalone SDK[11]. To do this process is needed to provide the *Kmodel* file and to put some information in the *C* source file of the firmware.

An important information needed is the memory location where is located the machine learning model. To flash properly the firmware and the *Kmodel* file into the EdgeAI device is needed *Kflash*[12], a Python-based Kendryte K210 UART ISP utility. In order to do this we need a *Json* file that reports the details

[9] https://github.com/kendryte/nncase
[10] https://kendryte.com/
[11] https://github.com/kendryte/kendryte-standalone-sdk
[12] https://github.com/kendryte/kflash.py

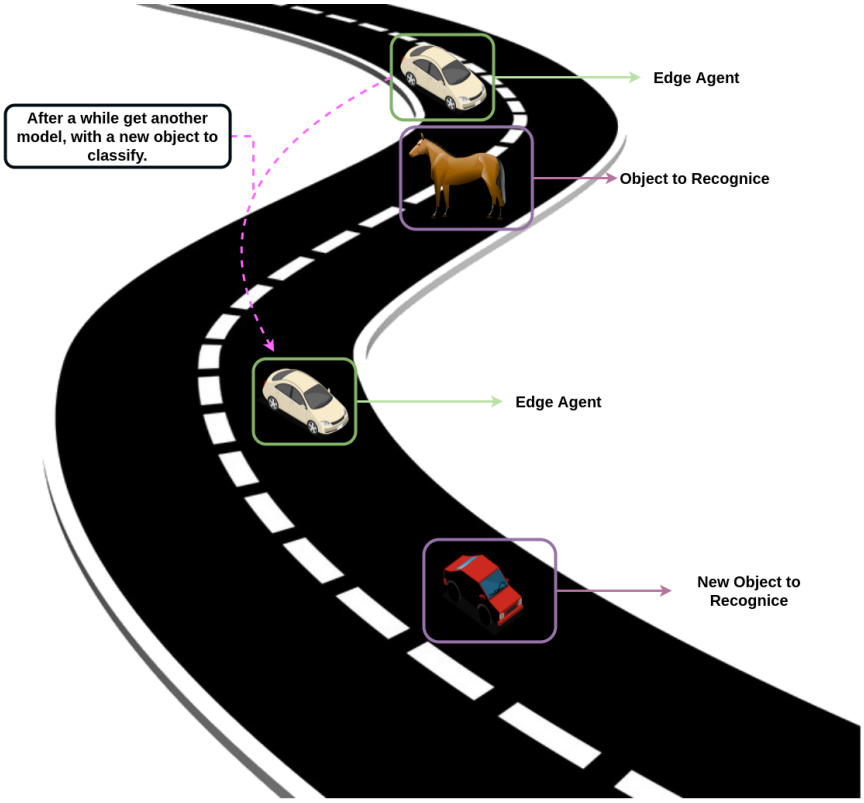

Fig. 6: Case study Approach

and the size of the two files in a standardised way. The agent then packs the bin, the *Kmodel* and the *Json* files in a *kfkpg* file.

Once obtained a file ready to be flashed in the EdgeAI device, the agent uploads it on a shared resource reachable from Edge Agent then it goes into *State 2*. Now, Car Agent can request for a new model to the Model Agent. If the Model Agent has calculated a new model it sends an URL and a list of labels to the car. The car stops driving to download and burn the model in the HAT. When it finishes to burn the model returns to the driving mode. If a new model is not available the car returns to drive without the burning process. If this is the case the Edge Agent receives a message containing an URL to the *kfpkg* file and a list of labels.

The *kfpkg* file contains three files.

1. A *Bin* file – This file is the firmware of the IOT device and comes from the compilation of source code in the Kendryte K210 standalone SDK
2. A *Kmodel* file – This is the file of the Machine Learning model in the special format *Kmodel*.

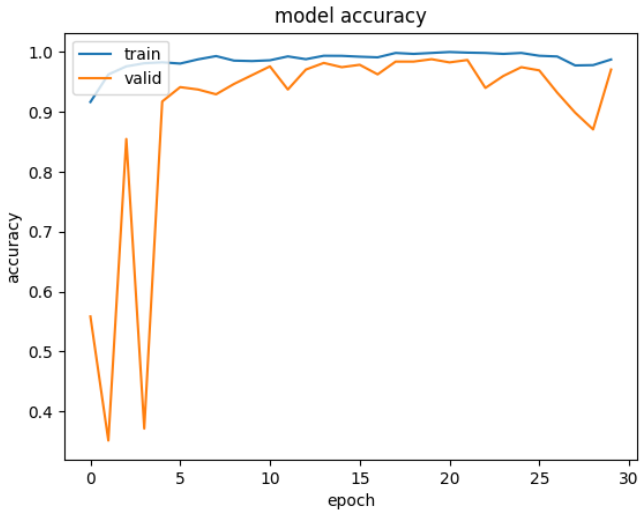

Fig. 7: Accuracy Graph

3. A *Json* file – This is a file in which are listed in a formatted way the file that the tool has to flash and the location of memory where to flash each file. We noticed that the device arise errors when the file are not burned in an aligned way so the address are calculated in a proper way.

4 Conclusions and future work

This article presents a tool that allows the integration of two Edge AI technologies for the classification and dynamic modification of deep learning models. This dynamic modification of models, allows us to perform the classification of objects within the Edge device. In this way, the developed system does not need to send the information obtained to the servers for analysis, thus reducing the latency in obtaining answers. Our system was built using a low cost and low energy consumption development system. As future work, tests are being carried out that will allow us to use our tool in other scenarios, as well as the possibility that the systems can learn other types of input data, such as sounds or sensor signals.

5 Acknowledgements

This work was partly supported by: ERASMUS+ Programme, KA1 Istruzione Superiore, Carta Erasmus+: 29388-EPP-1-2014-1-IT-EPPKA3-ECHE, ACCORDO PER LA MOBILITÀ ERASMUS PER STUDIO - a.a. 2019/2020, Progetto n°

2019-1-IT02-KA103-061203 - CUP: H25J19000080006, Generalitat Valenciana (PROMETEO/2018/002).

References

1. Anthony Chang. The role of artificial intelligence in digital health. In *Digital Health Entrepreneurship*, pages 71–81. Springer, 2020.
2. Li Yang, Tony L Henthorne, and Babu George. Artificial intelligence and robotics technology in the hospitality industry: Current applications and future trends. In *Digital Transformation in Business and Society*, pages 211–228. Springer, 2020.
3. Hamid Khayyam, Bahman Javadi, Mahdi Jalili, and Reza N Jazar. Artificial intelligence and internet of things for autonomous vehicles. In *Nonlinear Approaches in Engineering Applications*, pages 39–68. Springer, 2020.
4. He Li, Kaoru Ota, and Mianxiong Dong. Learning iot in edge: Deep learning for the internet of things with edge computing. *IEEE network*, 32(1):96–101, 2018.
5. Ricardo S Alonso, Inés Sittón-Candanedo, Sara Rodríguez-González, Óscar García, and Javier Prieto. A survey on software-defined networks and edge computing over iot. In *International Conference on Practical Applications of Agents and Multi-Agent Systems*, pages 289–301. Springer, 2019.
6. Tian Wang, Yaxin Mei, Weijia Jia, Xi Zheng, Guojun Wang, and Mande Xie. Edge-based differential privacy computing for sensor–cloud systems. *Journal of Parallel and Distributed Computing*, 136:75–85, 2020.
7. Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *arXiv preprint arXiv:1905.10083*, 2019.
8. Inés Sittón-Candanedo, Ricardo S Alonso, Juan M Corchado, Sara Rodríguez-González, and Roberto Casado-Vara. A review of edge computing reference architectures and a new global edge proposal. *Future Generation Computer Systems*, 99:278–294, 2019.
9. Ruimin Ke, Yifan Zhuang, Ziyuan Pu, and Yinhai Wang. A smart, efficient, and reliable parking surveillance system with edge artificial intelligence on iot devices. *arXiv preprint arXiv:2001.00269*, 2020.
10. Vittorio Mazzia, Aleem Khaliq, Francesco Salvetti, and Marcello Chiaberge. Real-time apple detection system using embedded systems with hardware accelerators: An edge ai application. *IEEE Access*, 8:9102–9114, 2020.
11. Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.
12. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.