An Analytical Framework for Trusted Machine Learning and Computer Vision Running with Blockchain

Tao Wang, Maggie Du, Xinmin Wu, Taiping He

SAS Institute Inc

Abstract

Machine learning algorithms often use data from databases that are mutable; therefore, the data and the results of machine learning cannot be fully trusted. Also, the learning process is often difficult to automate. A unified analytical framework for trusted machine learning has been presented in the literature to address both issues. It is proposed building a trusted machine learning system by using blockchain technology, which can store data in a permanent and immutable way. In addition, smart contracts on blockchain are used to automate the machine learning process. However, in such a blockchain framework, data efficiency is a big concern, because it is very expensive to store a large amount of data on blockchain. On the other hand, machine learning−based computer vision systems often rely on a lot of data. Therefore, to fully leverage a blockchain-based machine learning framework for computer vision systems, data efficiency issues must be addressed. This paper investigates how to enhance data efficiency in such a framework to bring computer vision systems to the edge. It presents a three-step approach. First, a lightweight machine learning model is trained on the server layer. Second, the trained model is saved in a special binary data format for data efficiency. Finally, the streaming layer takes these binary data as input and scores incoming new data in an online fashion. Real-time semantic segmentation for autonomous driving is used as an example to demonstrate how this approach works. This paper makes the following contributions. First, it improves the analytical framework for trusted computer vision systems based on blockchain. Second, the real-time semantic segmentation example shows how data-efficient learning for computer vision can be performed on the edge.

1. Introduction

Machine learning−based computer vision systems are playing a significant role in people's lives. They have been widely applied to retail, security, financial services, autonomous driving, production line automation, facial detection and recognition, health care, agriculture, intelligence, automation, and more. However, many such systems share some common concerns, including trustability, lack of automation, and data efficiency. First, it can be difficult to trust the results from a machine learning algorithm because machine learning algorithms use data from databases that are mutable. System administrators and illegal hackers can modify the data source, and this will eventually change the results, with or without notification. Second, it can be difficult to automate the machine learning pipeline. Currently, the machine learning pipeline is controlled and monitored mostly by human beings. Sometimes this process might begin or end at suboptimal times because of human involvement and the imperfect nature of human beings. Third, data efficiency is becoming a bigger issue than it used to be. We are currently in the Internet of Things (IoT) era, in which large amounts of streaming data are generated continuously. It is extremely impractical and inefficient to store all these data in a mutable database at a data center. Furthermore, most of these data can be irrelevant and useless. For example, in anomaly detection setting, only streaming data that contain anomaly events are worth storing or transmitting for further analysis; other data can be ignored and discarded. Note that some computer vision systems are not based on machine learning methods. Those systems are beyond the scope of this paper.

If trustability and automation are the primary goals, it is becoming obvious that blockchain [1] technology can greatly benefit machine learning−based computer vision systems. A blockchain is a continuously growing, single-linked list of immutable blocks (records) that are often secured using cryptography. It was invented by Satoshi Nakamoto [1] in 2008 as a public financial transaction ledger for use in the cryptocurrency Bitcoin [1]. Blockchain technology solved the Byzantine Generals Problem [3] and the double-spending problem [4, 5] by using a peer-to-peer (P2P) system without going through a trusted financial institution. This P2P network time-stamps transactions by hashing them using SHA-256 [9] into an ongoing chain of hash-based proof-of-work (PoW) [1], forming a block (record) that cannot be changed without redoing the PoW (also known as blockchain mining), which

requires substantial computing power. The longest blockchain with the highest combined difficulties serves not only as proof of the sequence of transactions witnessed but also as proof that it came from the pool that has the greatest computing power. With more and more computers added to the blockchain every day, it is increasingly difficult to hack the blockchain system unless the hacker overpowers the rest of the world, which is almost impossible. Therefore, people believe that the data that are stored in the Bitcoin blockchain are immutable and therefore can be fully trusted.

In addition to the data trustability that the Bitcoin blockchain provides, the Ethereum blockchain [2] provides automation capabilities. Whereas the Bitcoin blockchain is widely considered to be blockchain 1.0, the Ethereum blockchain is often considered to be blockchain 2.0. Ethereum uses blockchain technology not only as the foundation for cryptocurrency but also for decentralized applications (DApps) and decentralized autonomous organizations (DAOs). The Ethereum network provides a blockchain with a built-in, fully fledged, Turing-complete [6] programming language that can be used to implement so-called smart contracts. Smart contracts are essentially automated processes that can be used to encode arbitrary state transition functions, enabling you to create and run complicated systems (such as Facebook and Twitter, theoretically) on top of the Ethereum blockchain. The Ethereum blockchain opened a door to the largest development effort so far in the world of blockchain technology.

Although the Ethereum blockchain provides both trustability and automation capabilities, data efficiency remains a big concern. It is very expensive [16] to store large data sets on blockchain. Therefore, for real-time computer vision systems, moving the machine learning process to edge devices not only saves the device-to-cloud data round trip but also improves data efficiency. This is our motivation for writing this paper. With the massive amount of streaming data being generated and processed every day, edge computing has become an exciting facet of people's lives. Edge computing helps break the limits of cloud computing, particularly in dealing with computer vision. It is desirable to address the data efficiency issue to bring computer vision to the edge. However, edge devices usually have very limited memory and computing power. How can we train or score a machine learning model with such limitations? Based on an analytical framework [10] for trusted machine learning running with blockchain, this paper presents a three-step approach to improve data efficiency. First, a lightweight machine learning model is trained on the server layer. Second, the trained model is saved in a special binary data format for security and data efficiency. The blob of binary data is very compact and can be deployed to edge devices. Finally, the streaming layer takes this blob of binary data as input and scores incoming new data in an online fashion. We use real-time semantic segmentation for autonomous driving as an example to demonstrate how this approach works.

The rest of the paper is organized as follows. Section 2 reviews the prior art. Section 3 introduces the proposed computer vision architecture that serves as the backbone of our three-step approach. Section 4 presents a special binary data format and describes how to save trained machine learning models in this format. Section 5 uses real-time semantic segmentation for autonomous driving as an example to demonstrate the general workflow. Section 6 concludes the paper and suggests future research directions.

2. Review of prior art

In the literature, one paper [10] proposed to use blockchain to solve the trustability problem and suggested the use of a smart contract to solve the lack of automation problem for machine learning. The framework can be visualized using Figure 1.

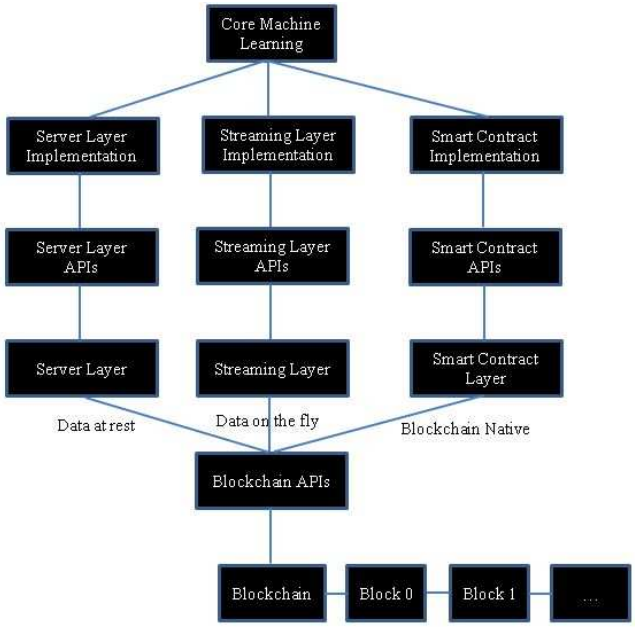

Figure 1. An analytical framework for trustable machine learning.

Let's look at each component of Figure 1 in more detail.

1. Core machine learning is the implementation of the machine learning algorithm in its native form. It often includes model initialization, model training, model validation, model scoring, model evaluation, model serialization, and model cleanup.

2. Server layer implementation is the implementation of the machine learning algorithm after code refactoring so that it can run on top of the server layer, which is often a cloud-based computing environment.

3. Streaming layer implementation is the implementation of the machine learning algorithm after code refactoring so that it can run on top of the streaming layer. A streaming layer is a computing environment that runs in sliding windows and discards old data after use.

4. Smart contract implementation is the implementation of the machine learning algorithm after code refactoring so that it can run on top of the smart contract layer. A smart contract [2] is simply an automated process. When a machine learning algorithm is implemented as a smart contract and is running on blockchain in a native way, the automation problem can be solved or largely alleviated.

5. Server layer APIs are, as you would expect, the APIs that are provided by the server layer. Currently, most server layer offerings come with SDK, which is a set of APIs that enable you to create applications to run on the server layer.

6. Streaming layer APIs are, as you would expect, the APIs that are provided by the streaming layer. Currently, most streaming layer offerings come with SDK, which is a set of APIs that enable you to create applications to run on the streaming layer.

7. Smart contract APIs are the APIs provided by the underlying smart contract layer. Currently, most smart contract layer offerings come with SDK, which is a set of APIs that enable you to create applications to run on the smart contract layer.

8. The server layer is a cloud-based computing environment that can train or score machine learning models.

9. The streaming layer is a computing environment that can train or score machine learning models in sliding windows.

10. The smart contract layer is a computing environment that can train or score machine learning models on blockchain as a native application.

11. Blockchain APIs are the APIs that are provided by the underlying blockchain. The server layer can obtain aggregated data from the blockchain via blockchain APIs. The streaming layer can obtain live data on the fly from the blockchain via blockchain APIs. The smart contract layer can obtain data from the blockchain via blockchain APIs in a native way.

12. The blockchain is a continuously growing list of immutable blocks.

Nowadays, most data that are stored on blockchain are financial transactions that often require fraud detection. However, in such applications, it can be hard to gather enough training data to train a good fraud detection model. One paper [11] proposed to use synthetic data generation to enrich training data.

Although the Ethereum blockchain−based analytical frameworks [10, 11] provide both trustability and automation capabilities, data efficiency remains a concern. In the IoT era, huge amounts of streaming data are generated continuously. On the other hand, it is extremely expensive and inefficient to store all these data in a mutable database or on blockchain. This paper investigates how to enhance data efficiency to bring machine learning−based computer vision systems to the edge. The details are discussed in the next section.

3. The proposed computer vision architecture

The three-step approach that this paper proposes is designed to bring machine learning−based computer vision system to edge devices. In the first step, a lightweight machine learning model is trained offline on the server layer. In the second step, the trained model is saved in a special binary data format for security and data efficiency. The blob of binary data is very compact and can be deployed on edge devices. In the third step, the streaming layer takes this blob of binary data as input and scores incoming new data in an online fashion.

Edge computing has three main advantages. First, it improves data efficiency and cost efficiency. With massive amounts of data generated and processed each day, it is extremely expensive and inefficient to store all these data in a mutable database or on blockchain. Furthermore, most of the data from edge devices can be completely irrelevant, with only a small portion worth storing. Therefore, with edge computing, after data stream in and are analyzed, the irrelevant data can simply be discarded. Second, edge computing improves security and privacy. It improves security by reducing the distance that data have to travel for storing and processing, thus reducing the risk of hackers intercepting the data during transmission. In the blockchain setting, only a very small portion of streaming data are saved on blockchain; most of data are analyzed and discarded on the edge. Third, edge computing increases speed. One of the driving forces for this type of computing is its speed. Without edge computing, an autonomous driving car would need to scan the road by using local cameras, send the images back for analysis and prediction, and wait to receive the analysis results in order to take the next move. Completing that entire process would take considerable time. In contrast, edge computing can reduce latency by completing all the steps on the car's computer. The processing algorithm runs locally, thus making it

possible to build more responsive applications that can achieve real-time reactions. Figure 2 shows the proposed computer vision architecture, including training on the server layer and scoring on the edge side.

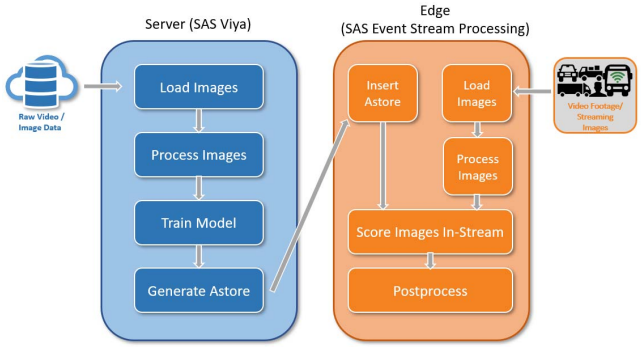

Figure 2. Proposed computer vision architecture.

This architecture has two major parts: model training that uses batches on the server side with SAS Viya [7], a cloud-based computing server; and edge computing with streaming data on edge devices enabled by SAS Event Stream Processing [8], a sliding-window-based streaming environment. The main design goal of this architecture is to ensure data efficiency. Under this architecture, a blockchain-based analytical framework [10, 11] is used to provide both trustability and automation capabilities.

Models are trained and validated on the server side before they are deployed to edge devices for scoring or prediction on the fly. In most cases, the training process consists of four steps. In the first step, images are loaded from raw files to create the training data. In the second step, the images that are used for training are processed. This can include image resizing, cropping, flipping, mutating, and possibly augmentation. In the third step, a deep learning model is built. In the fourth step, a special binary data format is used to save the trained model, which contains both model information and weight information. The special binary data format is called an analytic store (ASTORE) [11]. More details are discussed in Section 4.

After a model is trained, validated, and saved in ASTORE format, it can be readily deployed in edge devices by using SAS Event Stream Processing [8], a sliding-window-based streaming environment. A typical model contains four sliding windows. First, a source window takes the input images from either a CSV file or a video connector. Second, an image processing window is used when the images need preprocessing before they are ready for scoring or prediction. For example, if the input images are larger than allowed, then an image processing window is needed to resize the images in the designated dimensions. (Note that there is no image processing window in the example in Section 5 because image processing is not needed there.) Third, a model reader window reads in the model (in ASTORE format) and provides the model and associated parameter weights to the score window. Fourth, a score window takes the model as input and scores incoming images on the fly.

4. The ASTORE format for machine learning

The analytic store (ASTORE) format [11] is a binary data format for storing machine learning models. It was designed to be compact, unique, and immutable. Therefore, it is a natural choice for tasks that are related to blockchain.

An ASTORE model is essentially a serializable binary object. It contains a unique ASTORE key, which is universal and secured using cryptography. ASTORE format saves the machine model states, as well as all the information needed to reconstruct a model, into a platform-independent binary blob. This binary blob can be stored in a local file, a blob table in the cloud, a blob in the databases, or a blob on blockchain. An ASTORE blob contains model information, the entry function that is required to run this model, lists of input and output variables, and so on. All the information is compressed and serialized when the blob is created. It is unpacked and deserialized when the blob is loaded into memory. When an ASTORE is created, it can be transferred to and used on any platform in a portable way. Therefore, it is very flexible and can easily score new data in different environments.

A typical ASTORE contains following information:

- unique key
- model name and description
- machine learning algorithm information
- timestamps
- training parameters
- scoring functions and rules
- input variables, data types, and data formats
- output variables, data types, and data formats

One of the key features of the ASTORE format is that the unique key is generated and secured using cryptography. This key contains a string of characters such as 2580E6ABBCEE8B9C05689CDD952C60554A76A02E. The key ensures that each ASTORE data file is unique and immutable. If one bit of data is changed, the key is also changed. This makes the ASTORE format compatible with blockchain.

Other existing machine learning model formats like PMML [17], PFA [18], and ONNX [19] have potential security risks for commercial use since they are based on plain-text based files, which can be reverse-engineered. ASTORE's binary format makes it not possible to reverse-engineer. Another advantage of ASTORE is its blob is platform-independent, so it can be deployed to different

operating systems such as in local machine, in the cloud, on edge device, and even in GPU clusters.

5. Autonomous driving with real-time image segmentation

In this section, real-time semantic segmentation for autonomous driving is used as an example to demonstrate how data-efficient machine learning for computer vision can be performed on the edge.

In this example, a lightweight semantic segmentation model is trained that uses labeled street scenes and could potentially be deployed in vehicle cameras and sensors [12]. It demonstrates how to perform real-time semantic segmentation by using street scene images that are generated by the CARLA car simulator [13]. CARLA provides images in the RGB (red, green, blue) color space and labeled mask images that can be used to train models for autonomous vehicles. It is crucial for self-driving cars to segment objects on the street (such as other vehicles, pedestrians, road lines, and so on) so that the car can drive safely. The segmentation model has to be implemented in real time because any delays can lead to undesirable outcomes.

The training data set contains 4,800 color images that are all resized to 512 × 512. The mask images are of the same dimensions, with pixels labeled as belonging to one of the 13 predefined categories. These categories are Unlabeled, Building, Fence, Other, Pedestrian, Pole, Road line, Road, Sidewalk, Vegetation, Car, Wall, Traffic sign. Figure 3 shows four sample raw images and their corresponding mask images. Objects in the same category are marked with the same color. For example, all pixels that belong to the Car category are in medium blue, and all pixels that belong to the Road category are in green.

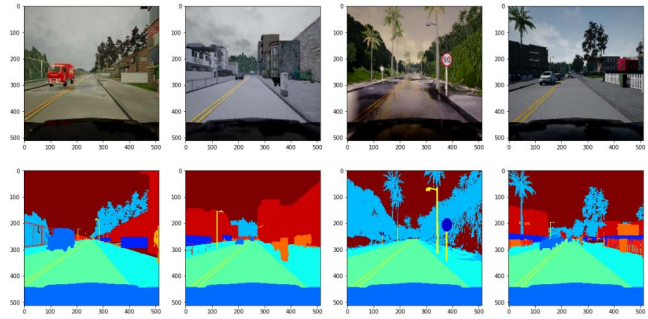

Figure 3. Training data visualization (raw images in the top row and ground truth masks in the bottom row).

The deep learning model architecture in this example is based on EfficientNet (also known as ENet) [12]. This architecture can be divided into several stages, and a diagram of each stage is shown in Figure 4. The initial block contains an input layer, followed by a 3 × 3 convolution layer with stride 2 (denoted by "s=2" in Figure 4) and a max-pooling layer, followed by a concatenation layer. In the downsampling bottleneck module [12], there is a 3 × 3 convolution layer with stride 2 to reduce feature sizes and an extra max-pooling layer, followed by a 1 × 1 expansion. In the upsampling bottleneck and regular modules, a 1 × 1 projection is used to reduce dimensionality. Then, the main convolution layer or transpose convolution layer (denoted by "Tconv" in Figure 4) is followed by another 1 × 1 expansion. In all modules, each convolution layer is followed by a batch normalization layer (not shown in Figure 4 to save space).

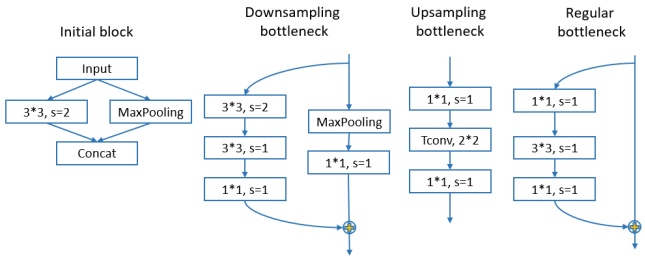

Figure 4. Diagram of each module in ENet.

The overall architecture is shown in the appendix. With input images of 512 × 512, the initial block and bottlenecks in the appendix downsample the feature size to 64 × 64. Further downsampling is avoided because reduced resolution hurts prediction accuracy. The last convolution layer adjusts the number of channels to match the number of categories in the data set.

This is a lightweight deep learning model for semantic segmentation, with only 0.2M parameters and 1.88 GFLOPS (billion floating point operations). The streaming model contains three windows in this example: a source window streams images in; a model reader window reads the trained model in ASTORE format; and a score window performs real-time scoring. The overall workflow (shown in Figure 5) of real-time semantic segmentation can be processed entirely on edge devices.

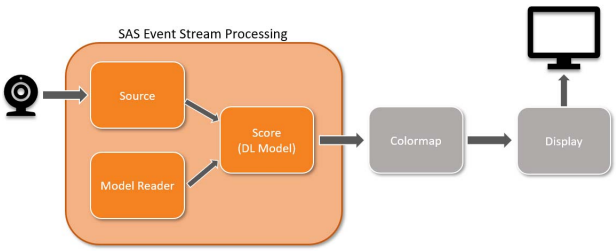

Figure 5. Real-time semantic segmentation on an edge device.

The test images are street scene images that are not used for training. In our experiment, the pixel accuracy on test images is 92.1%, meaning that fewer than 8% of pixels are misclassified. For some of the most important categories (road and car), the accuracies are 94.2% and 97.1%, respectively. Figure 6 shows the comparison among raw images (top row), ground truth images (middle row), and predicted images (bottom row).

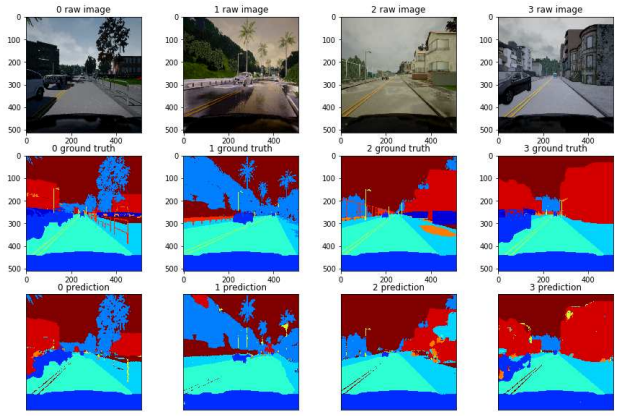

Figure 6. Predictions on test images. Top row: raw images. Middle row: ground truth. Bottom row: predictions.

Table 1 shows the computing power and scoring frames per second (FPS) on different NVIDIA devices, including Jetson AGX Xavier [14] and Jetson TX2 [15]. Scoring FPS values of 8 or higher are considered real-time values.

	Jetson AGX Xavier (edge)	**Jetson TX2 (edge)**
TFLOPS	11	1.5
FPS	21	10

Table 1. Computing power and scoring FPS.

6. Conclusion and future research

Ethereum blockchain−based analytical frameworks provide both trustability and automation capabilities for machine learning algorithms. However, data efficiency remains a concern. This paper presents a three-step approach to improve data efficiency in order to bring machine learning−based computer vision systems to edge devices. In the first step, a lightweight machine learning model is trained on the server layer. In the second step, the trained model is saved in a special ASTORE data format for security and data efficiency. In the third step, the streaming layer takes these ASTORE data as input and scores incoming new data in an online fashion. A real-time semantic segmentation for autonomous driving is used as an example to demonstrate how the approach works. In the future, we plan to further improve the ASTORE data format to enhance data efficiency and apply this analytical framework for trusted machine learning running with blockchain to more computer vision applications. We plan to provide a means to objectively and empirically study data efficiency aspects of the proposed framework. This is a key step towards fully integrating blockchain aspect of the proposed framework.

Acknowledgement

The authors would like to thank Ed Huddleston for his editing work and thank the anonymous reviewers for their reviews.

References

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Retrieved online at https://bitcoin.org/bitcoin.pdf, May 2018.

[2] Vitalik Buterin. A next generation smart contract and decentralized application platform. Retrieved online at https://whitepaper.io/document/5/ethereum-whitepaper, May 2018.

[3] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Trans. on Programming Languages and Systems* 4(3):382–401, 1982.

[4] Ivan Osipkov, Eugene Vasserman, Nicholas Hopper, and Yongdae Kim. Combating double-spending using cooperative P2P systems. doi:10.1109/ICDCS.2007.91, 2007.

[5] Jaap-Henk Hoepman. Distributed double-spending prevention. arXiv preprint arXiv:0802.0832v1, 2008.

[6] Andrew Hodges and Alan Turing. The Enigma. London: Burnett Books, 1983.

[7] Jonathan Wexler, Susan Haller, and Radhikha Myneni. An overview of SAS Visual Data Mining and Machine Learning on SAS Viya. SAS Global Forum, 2017.

[8] John Davis. How's Your Sports ESP? Using SAS Event Stream Processing with SAS Visual Analytics to Analyze Sports Data, SAS Global Forum, 2017.

[9] Henri Gilbert and Helena Handschuh. Security analysis of SHA-256 and Sisters. Selected Areas in Cryptography, 2003, pp. 175–193.

[10] Tao Wang. A unified analytical framework for trustable machine learning and automation running with blockchain. IEEE BigData Conference workshops, 2018.

[11] Tao Wang, Xinmin Wu, and Taiping He. Trustable and Automated Machine Learning Running with Blockchain and Its Applications, KDD Workshops, 2019.

[12] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation. arXiv preprint arXiv:1606.02147, 2016.

[13] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Valdlen Koltun. CARLA: An Open Urban Driving Simulator. arXiv preprint arXiv:1711.03938, 2017.

[14] NVIDIA Jetson AGX Xavier. Retrieved online at https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/, March 2020.

[15] NVIDIA Jetson TX2. Retrieved online at https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/, March 2020.

[16] A. Besir Kurtulmus and Kenny Daniel. Trustless Machine Learning Contracts; Evaluating and Exchanging Machine

Learning Models on the Ethereum Blockchain, arXiv preprint arXiv:1802.10185, 2018.

[17] Robert Grossman, Stuart Bailey, Ashok Ramu, Balinder Malhi, Philip Hallstorm, Ivan Pulleyn, and Xiao Qin. The management and mining of multiple predictive models using the predictive modeling markup language, Information and Software Technology, vol. 41, no. 9, pp. 589–595, 2002, DOI: 10.1016/S0950-5849(99)00022-1.

[18] Jim Pivarski, Collin Bennett, and Robert Grossman. Deploying analytics with the Portable Format for Analytics (PFA), KDD 2016.

[19] Kim Hazelwood, et al., Applied machine learning at Facebook: A datacenter infrastructure perspective, International Symposium on High-Performance Computer Architecture (HPCA), 2018.

[20] Ravi Kiran Rama, et al., Trusted Multi-Party Computation and Vefiriable Simulations: A Scalable Blockchain Apporach. arXiv preprint arXiv:1809.08438, 2018.

Appendix: The Overall Model Architecture

Stage	**Name**	**Type**	**Output Size**
0	Initial	Initial	256 × 256 × 16
1	BNeck1.0	Downsampling	128 × 128 × 64
	BNeck1.1–BNeck1.4	Regular	128 × 128 × 64
2	BNeck2.0	Downsampling	64 × 64 × 128
	BNeck2.1–BNeck2.4	Regular	64 × 64 × 128
3	BNeck3.1–BNeck3.4	Regular	64 × 64 × 128
4	BNeck4.0	Upsampling	128 × 128 × 64
	BNeck4.1–BNeck4.2	Regular	128 × 128 × 64
5	BNeck5.0	Upsampling	256 × 256 × 16
	BNeck5.1	Regular	256 × 256 × 16
6	BNeck6.0	Upsampling	512 × 512 × 16
	Conv	Convolution	512 × 512 × 13