iBranchy: An Accelerated Edge Inference Platform for IoT Devices$^{\diamond}$

Santosh Kumar Nukavarapu*
nukavarapuskk@vcu.edu
Virginia Commonwealth University
Richmond, Virginia, USA

Mohammed Ayyat*
ayyatma@vcu.edu
Virginia Commonwealth University
Richmond, Virginia, USA

Tamer Nadeem
tnadeem@vcu.edu
Virginia Commonwealth University
Richmond, Virginia, USA

ABSTRACT

With the phenomenal growth of IoT devices at the network edge, many new applications have emerged, including remote health monitoring, augmented reality, and video analytics. However, securing these devices from different network attacks has remained a major challenge. To enable more secure services for IoT devices, threats must be discovered quickly in the network edge and efficiently dealt with within device resource constraints. Deep Neural Networks (DNN) have emerged as solution to provide both security and high performance. However, existing edge-based IoT DNN classifiers are neither lightweight nor flexible to perform conditional computation based on device types to save edge resources. Dynamic deep neural networks have recently emerged as a technique that can accelerate inference by performing conditional computation and, therefore, save computational resources. In this work, we design and develop an accelerated IoT classifier *iBranchy* based on a dynamic neural network that can perform quick inference with fewer edge resources while also providing flexibility to adapt to different hardware and network conditions.

CCS CONCEPTS

• Security and privacy → Mobile and wireless security; **• Computing methodologies → Neural networks**.

KEYWORDS

Internet of things, Deep neural networks, Dynamic networks, Security

ACM Reference Format:
Santosh Kumar Nukavarapu, Mohammed Ayyat, and Tamer Nadeem. 2021. iBranchy: An Accelerated Edge Inference Platform for IoT Devices$^{\diamond}$. In *The Sixth ACM/IEEE Symposium on Edge Computing (SEC '21), December 14–17, 2021, San Jose, CA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3453142.3493517

*Both authors contributed equally.

$^{\diamond}$This material is based upon work supported in part by the US National Science Foundation under Grant No. CNS-1764185 and the Commonwealth Cyber Initiative - an investment in the advancement of cyber R&D, innovation, and workforce development. For more information about CCI, visit www.cyberinitiative.org.

1 INTRODUCTION

In recent years there has been significant growth in the number of IoT devices. These devices are deployed in proximity to the data sources at the network edge and support various machine learning-based applications, including remote health monitoring, augmented reality, and video analytics. To secure these devices from different types of network attacks [1, 2], the IoT/edge applications typically exploit machine learning models such as Deep Neural Network (DNN) models for different security functionalities such as traffic classification, device identification, and anomaly detection. However, the performance of these models when running on edge-based IoT devices will be significantly impacted by the limitations of the device resources which will reflect on the performance of these devices. Therefore, it is highly desirable to develop techniques to optimally accelerate the inference computations of DNN models in order to enable real-time applications and conserve energy for edge devices/IoT.

Very recently, different types of DNNs referred to as Dynamic DNNs (D^2NN) have been proposed [7] to provide low-latency and power-saving on IoT/edge devices. In contrast to traditional DNNs, dynamic DNNs are capable of performing conditional computations and selectively activate just sections of the network model, whereas traditional DNNs use the entire network model in the computation even when only a certain portion of the network is sufficient to make the inference. For example, BranchyNet [17], one of the popular D^2NN models, terminates its computation and infers early if the earlier layers of the network have sufficient confidence without requiring all of the subsequent layers to participate in computation, thus reducing inference latency and power consumption. The typical architecture of BranchyNet model consists of a network with multiple layers and different branches. Every branch of the network is followed a classification output component known as exit point. During inference, the early termination at a branch happens only if the exit point has adequate confidence to make the inference. Recent research has demonstrated that for the vast majority of inputs, the model will exit at the early exit point of the network during inference without requiring computation from the rest of the network [17]. These D^2NN models, which offer characteristics like on-demand computing, hardware adaptability, and fewer resource constraints, are therefore gaining popularity for constructing and developing high-performance IoT/edge applications [5, 13].

Motivated by the above observations, in this paper, we design and develop a dynamic DNN (D^2NN) IoT classifier based on BranchyNet as a model classifier to classify actual IoT and non-IoT devices and evaluate the model with edge-based constraints such as inference time and power consumption. We summarize the contributions of this paper as follows:

- We design and develop *iBranchy*, a dynamic early exit multi-class classifier for classifying IoT devices based on their network traffic.
- We evaluate the features of *iBranchy* using both real IoT and non-IoT devices.

2 BACKGROUND

2.1 Static Compression vs Dynamic DNNS (D^2NN)

In recent years there has been significant research over accelerating machine learning models for edge deployments using different approaches [3] such as compression [4, 6] and knowledge distillation [8]. In compression-based techniques, the original DNN network may be pruned in different ways to remove any insignificant parts of the network. For example, in [6] authors compress the network using quantization method while authors in [4] compress the convolutional layers through a redundancy approach. However, in knowledge distillation technique [8], instead of compressing the neurons or weights of an existing complex DNN network model, a new lightweight network called student model is designed with very few layers and is trained to learn from the knowledge representations outputs of the pre-trained complex teacher model. This method has shown to be very effective with high performance for resource constraints and an insignificant drop in accuracy.

One of the major disadvantages of the above accelerated methods based on compression or distillation is that they are static with respect to computation and inference time as all the components of the DNN network will participate in inference phase computation irrespective of the type of input or environmental conditions. In contrast to static networks, dynamic networks D^2NN can change their internal structure or parameters during the inference phase, giving them greater flexibility and better adaptability to the underlying use case [7]. The D^2NN models accomplish this dynamic flexibility and adaptability through a conditional computation architecture that allows them to only selectively activate particular sections of their network based on context, such as input data or environmental conditions. Early-Exit-based models [7, 17] are one of the most popular categories of the D^2NN networks that have a multi-exit design where an exit is an early inference point attached to selected components of the network and can be conditionally activated based on the complexity of the input data. Therefore, in this paper we use BranchyNet [17] a very well adapted Early-Exit-based D^2NN model that is gaining popularity for edge scenarios to implement our D^2NN based IoT network classifier.

2.2 BranchyNet

BranchyNet is an Early-Exit-based D^2NN model that supports an early inference of certain input samples using multi-branch and multi-exit design. Like a traditional DNN classifier, the BranchyNet network architecture consists of a multi-layer network followed by a softmax layer for output predictions. However, in addition to the main network, a small network called branches are added to the outputs of different layers of the main network. These branches, similar to the main network are also followed by softmax layer. The outputs from the different softmax of the different branches and the main network are called as exits. The multi-exit approach implemented in BranchyNet is based on the observation that the earlier layers of the network can perform inference for most of the input samples, thus allowing most of the inputs to exit early and thus reducing the overall network computation and reducing the average runtime and power computation.

In BranchyNet training a softmax cross entropy loss function is used for minimizing the network misclassification rate similar to traditional DNNs. However, the overall loss of the network is calculated using a weighted loss function consisting of losses at different exits of the network. The choice of the weights for each exit-specific loss is a hyper-parameter and impacts the model performance. During the inference phase, an input sample exits the network only if it is predicted with a confidence that is within the confidence threshold assigned to that particular exit. More specifically, BranchyNet uses an entropy-based confidence threshold where a sample exits from a particular exit only if it was predicted with entropy less than the threshold assigned for that particular exit. If the entropy of the input sample is larger than the given threshold, the sample is sent to the next exit for inference and the process continues till the sample reaches the final exit at the end of the main network.

The choice of thresholds at different exits is a run-time hyper-parameter that impacts model inference phase performance. The lesser threshold value at an exit will ensure the samples predicted with high entropy to be pushed to later exits, thus improving the model's accuracy at the cost of early inference. Two approaches are proposed in BranchyNet for threshold selection, in the first approach, different threshold values could be tested in an iterative method and finally choosing the best configuration based on user requirements such as higher accuracy with an increase in inference time or lower accuracy with a decrease in inference time. The second approach is to use a neural network-based approach that can automatically fine-tune the threshold values. In this paper, we implement the first approach to pick the threshold values based on performance requirements.

3 RELATED WORK

Given the significance of IoT device classification, some of the recent IoT network classifiers [15, 16] utilize either probabilistic model [15] or traditional machine learning based unsupervised model [16] for classifying network based IoT devices, but no deep learning-based solutions. However, in this work, given the growing popularity of deep learning-based classifiers that provide the features such as automatic feature extraction and better accuracy from raw network data [9, 11, 12], we build D^2NN based model for IoT classification that augments traditional DNN based features with additional features that can support edge-based resource constraints. For example, existing IoT classifiers that use different deep learning based approaches such as CNN-RNN [9], autoencoders-bayesian modeling [12] and semi-supervised GAN [11] require all of their DNN network entitiess to participate in computation for realizing their system. In contrast, our approach can conditionally activate only selected sections of the DNN network to save energy and inference time.

Other D^2NN related advances include developing a partition method [13] with BranchyNet and distributed DNN techniques [18] across the cloud and the edge devices for better performance and faster response times. Similarly, FlexDNN [5] - a Early-Exit based

Table 1: The list of IoT and non-IoT devices used

Device Name	Device Type
Smart Things	Hub
Amazon Echo	Speaker
iHome	Speaker
Triby Speaker	Speaker
Netatmo Welcome	Camera
TP-Link Day Night Cloud Camera	Camera
Samsung SmartCam	Camera
Dropcam	Camera
Insteon Camera	Camera
Withings Smart Baby Monitor	Camera
Nest Dropcam	Camera
Belkin Wemo Switch	Acutator
TP-Link Smart Plug	Acutator
Light Bulbs LiFX Smart Bulb	Acutator
NEST Protect Smoke Alarm	Sensor
Netatmo Weather Station	Sensor
Withings Smart Scale	Sensor
Blipcare Blood Pressure Meter	Sensor
Withings Aura Smart Sleep	Sensor
Belkin Wemo Motion Sensor	Sensor
PIX-Star Photo-frame	Digital Frame
Laptop	Non-IoT
Macbook	Non-IoT
iPhone	Non-IoT
Samsumg Galaxy Tab	Non-IoT

model accelerates video analytics on resource-constrained devices. However, our work significantly differs from other edge-based D^2NN models as we design, implement, and evaluate *iBranchy* for IoT-based network devices which is more challenging due to the fact that network data is not simple as traditional image or set of frames based video data. Furthermore, we apply an encoding scheme based on network flows based on our previous work [11] to implement BranchyNet for IoT devices effectively. Very recently, in [10] authors use a context-based approach that selects the most appropriate anomaly detection model from the hierarchy of models deployed at the edge, cloud, and device to meet edge-based resource constraints, and it is implemented using sensor and power consumption datasets. However, unlike their work which use distributed models, we implement a single model classifier based on BranchyNet that can uniquely identify IoT/Non-IoT devices using IoT network traffic.

4 *IBRANCHY* FRAMEWORK

In this paper, we design and implement *iBranchy*, an IoT network classifier for edge-based systems. Figure 1 shows an overview of the *iBranchy* framework that consists of two components a network flow encoding component and an accelerated device discovery component. The new devices' network flows are encoded through the network encoding component and sent to the accelerated device discovery component, a BranchyNet based Early-Exit multi-class classifier, to identify the new device class (device name). We discuss below implementation of each of these components in detail.

Network Flow Encoding: The network flow encoding component encodes the network flows of the devices by mapping the raw network packets of a flow into a three-dimensional array based on our previous work [11]. This encoded flow array is given as an input to the accelerated device discovery component to extract the hidden features and uniquely identify the device. The flow encoding is implemented in multiple steps, first, we take the raw byte streams of the packets in a flow as the features and convert them into their equivalent hexadecimal integer representation. Next, similar to the three-dimensional encoding of a non-gray image with pixel intensities as features and multiple channels represented as a 2D array used for image classification, we create a three-dimensional encoding for the hexadecimal representation of the flow. However, each channel in this encoding is a 2D packet hex stream with a dimension of 56*56 to accommodate the maximum size of a MAC packet of 1500 bytes, and all packets are arranged in arrival sequence. The number of packets in a flow is a configurable parameter and can be adjusted based on the model's performance. In our experiments, we take the first 5 packets of the flow and append zeros if the flow has packets shorter than that, an approach similar to [12]. It is to be noted that for the flow encoding, we do not consider any network-dependent header data such as IP address, etc. to enable network independent model deployment. This flow-based encoding scheme using raw packet streams is a more efficient solution for BranchyNet based model, which uses CNN layers and can extract spatial features of the network byte data much efficiently.

Accelerated Device Discovery: The *iBranchy* accelerated device discovery component uniquely identifies a device into its corresponding class. However, unlike other popular DNN based IoT classifiers where all the layers participate in computation [9, 11, 12], *iBranchy* may exit at different stages of the model with probable very early exits that can significantly decrease the inference time and consume less energy. To design the *iBranchy* model we first design a baseline DNN IoT classifier and then, using the baseline DNN, we build an IoT based BranchyNet version with newly added branches where a branch is a set of layers or a very small DNN with early exit point as discussed in section 2.

Figure 1 shows the overall architecture of the *iBranchy* model, which consists of multiple convolutional layers followed by a linear layer with final exit and two branch exits. In *iBranchy*, we try different design configurations and choose the early exits at Branch #1 and Branch #2 at the CONV layer #1 and CONV layer #3, respectively. At the inference stage, if the entropy for a branch of a testing sample is less than the runtime threshold, the inference can be made early without further computation from subsequent layers. We chose a lightweight design for the branches with linear layers to add less overhead at run time for edge deployments but that could be further optimized. In *iBranchy*, the probability of a network layer to participate in the inference computation is dependent on the threshold values set for different exits. Therefore, at run time, based on the environment requirements such as battery power (low or high) or Wi-Fi signal strength *iBranchy* can be configured dynamically to exit more samples at early layers to save edge resources. However, choosing very low threshold values can significantly impact the inference accuracy and therefore needs to be optimized based on the operating constraints and is out of the scope of this work. In the next section, we evaluate *iBranchy* for edge based deployment requirements.

5 EVALUATION

In our experiments, we extract 56,000 flows from the combined network flows of twenty-one IoT devices and four non-IoT devices as shown in Table 1 from the publicly available UNSW [15] dataset

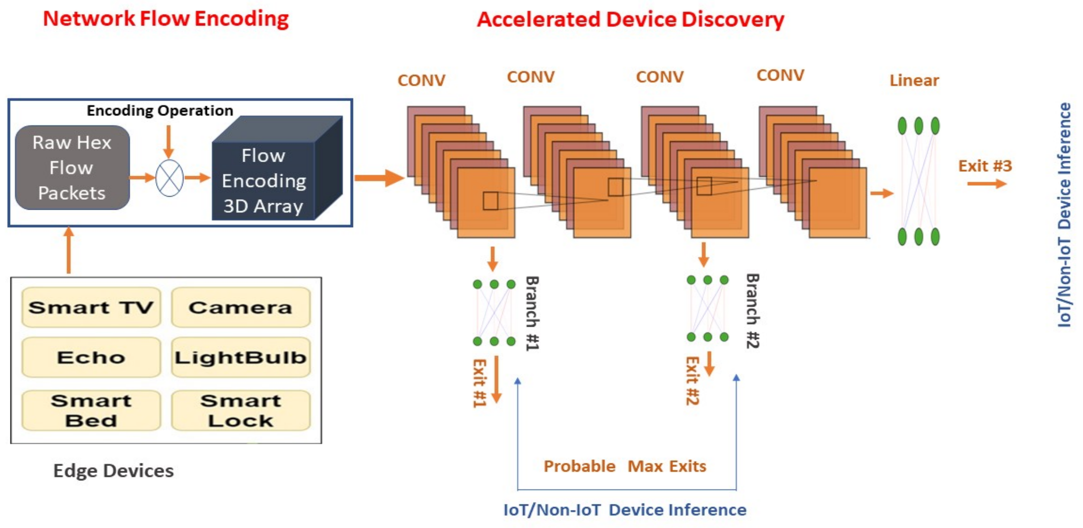

Figure 1: ***iBranchy*** **Accelerated Edge Classifier**

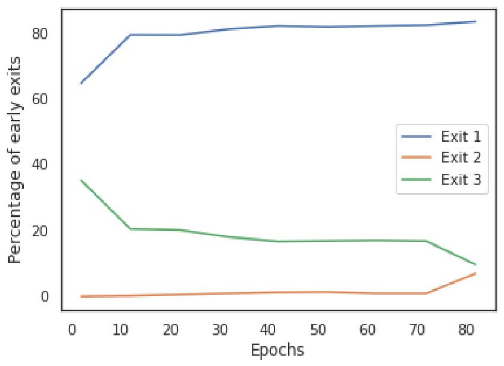

Figure 2: The change in ***iBranchy*** **exits through training**

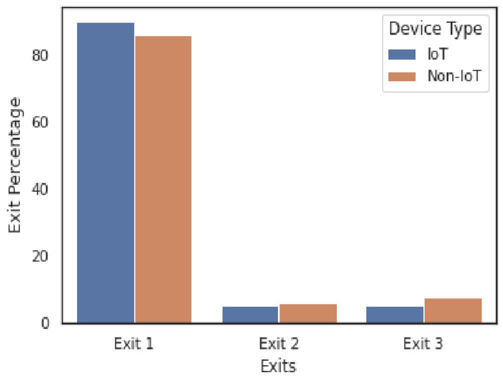

Figure 3: The percentage of exits of ***iBranchy*** **for different device types**

to evaluate and validate *iBranchy* for different edge-based requirements. The raw pcap files are processed to create flows for encoding with network-independent header fields and payload data similar to our previous work [11]. We implement our model using PyTorch [14] and also using code repositories [17, 19]. We use an NVIDIA-based GPU Server with 32 GB RAM to perform our experiments, we also use 80% of data for training and 20% for testing our model. For evaluating *iBranchy*, we consider multiple performance metrics and flexibility-based scenarios. First, we assess the ability of *iBranchy* to exit in earlier exits for the majority of the testing data. Next, we measure *iBranchy* in terms of resource utilization and power consumption and finally assess its flexibility to provide a trade-off between accuracy and edge resources based on the context. We discuss each of these evaluations below.

5.1 Significance of Early Exit on IoT devices Network Traffic

Figure 2 shows the percentage of exits from the different exits of *iBranchy* during the training. The number of exits from Exit #1 increases as the training progress, while the number of exits from Exit #3 or the final exit of the network decreases. Therefore, *iBranchy*, over time, learns to extract the significant features for device type inference in the early layers of the DNN. Figure 3 shows the percentage of exits for each device type. The number of exits for device types from Exit #2 is not that significant compared to the number of exists from Exit #1. The less significant exists from Exit #2 could be either because of the choice of the network design or the second branch has not captured enough features for the remaining of the device type samples. *Therefore, the distribution of device exits across different branches is determined by the network design and features of the IoT network dataset.*

5.2 Performance of *iBranchy*'s Edge Deployment

Table 2 shows the runtime performance of *iBranchy*. The results showcases that it performs significantly better with edge-based requirements compared to our baseline DNN. The power consumption of the *iBranchy* model layers decreases by 35.84% than the baseline model layers. Therefore, *iBranchy* uses fewer network components for computing the inference when compared to the baseline IoT DNN model. Moreover, the run time for *iBranchy* model layers is faster than the baseline model layers by about 34.79%. Furthermore, the accuracy drop for the *iBranchy* model is not that significant, as it only drops by about 2%. *Therefore, iBranchy achieves efficient accuracy with fewer resources and is more efficient for edge-based IoT network classification.*

Model	Accuracy	Power (%)	Inference Time (s)
Baseline DNN	0.9303	100	0.0569
iBranchy	0.9146	64.16	0.0371

Table 2: Performance of edge deployment with 11000 samples

5.3 Flexibility and Adaptability of *iBranchy* to Hardware and Network Conditions

Given that BranchyNet can be configured dynamically to increase or decrease the entropy threshold values during runtime [17], a similar threshold runtime setting can be used to dynamically configure *iBranchy* for various efficiency schemes. For example, *iBranchy* can change to a low-efficiency scheme with low power consumption during low battery status and switch back to a high-efficiency scheme when the battery is high later. Similarly, during low bandwidth due to bad Wi-Fi, the *iBranchy* can change to a low-efficiency scheme with low inference time at the cost of accuracy and switch back to high efficiency with good Wi-Fi. *Therefore, iBranchy is flexible to adapt to different edge resource contexts and choose the appropriate efficiency scheme.* . However, testing different configurations of the thresholds of *iBranchy* for efficiency and performance trade-offs is out of the scope of this work.

6 DISCUSSION

In this paper, we design and implement an D^2NN based Early-Exit classifier *iBranchy* that can do accelerated inference for edge-based IoT and non-IoT device types with fewer resources. Our framework is flexible enough to support a context-driven network traffic classification system based on the edge environment state such as battery status or network conditions given the conditional computation capabilities of the D^2NN models. Moreover, since we implement *iBranchy* using network encoding component that uses raw network bytes of the device network traffic as input, *iBranchy* can support various device categories from multiple vendors without requiring manual feature engineering. Furthermore, given our encoding process considers network header independent fields, *iBranchy* can seamlessly integrate with different network environments. We believe that our results give some early insights on applying D^2NN such as *iBranchy* for edge-based IoT classifiers. For example, the significant number of IoT devices flowing from Exit #1 shows that the later layers can be cumbersome, requiring a better network design. In our future work, we plan to extend our work to more significant device types with more complex features such as devices from the same vendors and analyze the distribution of these device types over the different exits.

REFERENCES

[1] Abbas Acar, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, and Selcuk Uluagac. 2020. Peek-a-boo: I see your smart home activities, even encrypted!. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 207–218.

[2] Noah Apthorpe, Dillon Reisman, Srikanth Sundaresan, Arvind Narayanan, and Nick Feamster. 2017. Spying on the smart home: Privacy attacks and defenses on encrypted iot traffic. *arXiv preprint arXiv:1708.05044* (2017).

[3] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2017. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282* (2017).

[4] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*. 1269–1277.

[5] Biyi Fang, Xiao Zeng, Faen Zhang, Hui Xu, and Mi Zhang. 2020. Flexdnn: Input-adaptive on-device deep learning for efficient mobile vision. In *2020 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 84–95.

[6] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. 2014. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115* (2014).

[7] Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. 2021. Dynamic Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), 1–1.

[8] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[9] Manuel Lopez-Martin, Belen Carro, Antonio Sanchez-Esguevillas, and Jaime Lloret. 2017. Network traffic classifier with convolutional and recurrent neural networks for Internet of Things. *IEEE Access* 5 (2017), 18042–18050.

[10] Mao V Ngo, Tie Luo, and Tony QS Quek. 2021. Adaptive Anomaly Detection for Internet of Things in Hierarchical Edge Computing: A Contextual-Bandit Approach. *arXiv preprint arXiv:2108.03872* (2021).

[11] Santosh Kumar Nukavarapu and Tamer Nadeem. 2021. Securing Edge-based IoT Networks with Semi-Supervised GANs. In *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. IEEE, 579–584.

[12] Jorge Ortiz, Catherine Crawford, and Franck Le. 2019. DeviceMien: network device behavior modeling for identifying unknown IoT devices. In *Proceedings of the International Conference on Internet of Things Design and Implementation*. 106–117.

[13] Roberto G Pachecom and Rodrigo S Couto. 2020. Inference time optimization using branchynet partitioning. In *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 1–6.

[14] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019), 8026–8037.

[15] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijenayake, Arun Vishwanath, and Vijay Sivaraman. 2018. Classifying IoT devices in smart environments using network traffic characteristics. *IEEE Transactions on Mobile Computing* 18, 8 (2018), 1745–1759.

[16] Arunan Sivanathan, Hassan Habibi Gharakheili, and Vijay Sivaraman. 2019. Inferring IoT Device Types from Network Behavior Using Unsupervised Clustering. In *2019 IEEE 44th Conference on Local Computer Networks (LCN)*. IEEE.

[17] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. 2016. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2464–2469.

[18] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. 2017. Distributed deep neural networks over the cloud, the edge and end devices. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 328–339.

[19] Neta Zmora, Guy Jacob, Lev Zlotnik, Bar Elharar, and Gal Novik. 2019. Neural network distiller: A python package for dnn compression research. *arXiv preprint arXiv:1910.12232* (2019).