2023 20th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON) | 979-8-3503-0052-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/SECON58729.2023.10287462

Class-Aware Neural Networks for Efficient Intrusion Detection on Edge Devices

Mohammed Ayyat, Tamer Nadeem and Bartosz Krawczyk
Department of Computer Science
Virginia Commonwealth University
Richmond, VA 23220, USA
{ayyatma, tnadeem, bkrawczyk}@vcu.edu

***Abstract*—The exponential growth of IoT and edge devices has led to their widespread use across various applications. However, the security of these devices remains a significant concern due to their vulnerability to a broad spectrum of cyber-attacks. Network Intrusion Detection Systems (NIDS) are crucial for identifying and mitigating such threats. Traditional NIDS approaches, while effective, struggle to detect sophisticated modern attacks and often require substantial computational power and memory, which may not be feasible for edge devices. Machine learning and neural network-based methods have demonstrated promising improvements in NIDS detection accuracy. Yet, their deployment on resource-constrained edge devices presents a challenge. This has led to the development of Dynamic Neural Networks, an approach that allows models to adapt according to the input, making them more efficient and lightweight. However, these networks are class-agnostic, rendering them unsuitable for handling cases with uneven classification priorities. In this paper, we introduce ClassyNet, a platform designed for efficient, class-aware NIDS on edge devices. ClassyNet leverages class-specific feature extraction and a class-specific neural network architecture to enhance intrusion detection efficiency. Experimental results indicate that our proposed approach matches the detection accuracy of traditional machine learning and neural network-based methods while significantly improving resource efficiency.**

***Index Terms*—Network Intrusion Detection Systems, Edge Deployment, Inference Latency, Early-Exit Neural Networks.**

I. Introduction

The rapid expansion of the Internet of Things (IoT) has resulted in a multitude of devices and applications that generate substantial amounts of data. This surge of IoT devices has broadened the potential attack surface for cyber threats, making the task of detecting and preventing network intrusions increasingly complex. As a result, secure communication on edge and IoT devices has become critical. In this context, Network Intrusion Detection Systems (NIDS) play a pivotal role in safeguarding IoT devices and networks. These systems function by classifying network traffic to identify any anomalies or specific classes of traffic that may indicate a threat. However, traditional NIDS methods often face limitations due to the computational resources available on edge and IoT devices. This has led researchers to explore the potential of Deep Neural Networks for enhancing the effectiveness and efficiency of NIDS in this increasingly data-rich environment.

The past decade has seen a significant surge in the application of Deep Neural Networks (DNNs). They have been effectively utilized in various domains such as image recognition [1], speech recognition [2], natural language processing [3], self-driven network protocols [4], synthesizing network traffic [5], and many other domains achieving impressive results. Furthermore, their efficacy in Network Intrusion Detection Systems (NIDS) has been well-demonstrated [6]–[8]. The proliferation of DNNs has been facilitated by the availability of vast amounts of data and advancements in computing power, particularly with the advent of graphics processing units (GPUs) and specialized hardware like tensor processing units (TPUs). However, as these neural networks evolve to become more complex and larger in size, the necessity for efficient and scalable deployment becomes increasingly critical.

To address the challenge of deploying DNNs on edge devices and the IoT, dynamic neural networks have been proposed [9]. Dynamic neural networks, particularly early-exit dynamic neural networks [10], allow a network to exit early if it has enough evidence to classify an input into a particular class, reducing the computational overhead of classification tasks. This enables more computationally intensive tasks to run on edge devices and the IoT, improving the performance and scalability of IoT applications.

Traditional early-exit dynamic neural networks such as BranchyNet [11] have been proposed to address the challenge of deploying DNNs on edge devices and the IoT. However, these networks do not take into account the importance of different classes in the classification task, which may result in suboptimal performance in detecting high-priority threats in network intrusion detection. To address this issue, we propose a novel approach named *ClassyNet*, which augments early-exit dynamic neural networks with class-based classification capabilities. Our approach assigns different priorities to different types of network threats, allowing high-priority threats to exit the network early for faster detection. ClassyNet builds on the benefits of BranchyNet, while also taking into account the importance of different classes in the classification task, enabling more efficient and effective management of network security.

This work was supported in part by the U.S. National Science Foundation under Grant OAC-2212424; and in part by the Commonwealth Cyber Initiative, an investment in the advancement of cyber research and development, innovation, and workforce development.

In this paper, we introduce ClassyNet, an innovative approach that enhances traditional early-exit dynamic neural networks such as BranchyNet [11]. Our approach is highly scalable and seamlessly integrates into existing network intrusion systems. We demonstrate its effectiveness in improving the speed and accuracy of high-priority threat detection through evaluations on multiple network intrusion datasets, including NSL-KDD [12], UNSW-NB15 [13], and CICIDS17 [14]. The paper begins with a brief background on dynamic neural networks and their role in intrusion detection (Section II). It then discusses the motivation behind ClassyNet and its potential use cases (Section III) and then provide an overview of ClassyNet's architecture (Section IV). Following this, we describe the datasets used for testing our approach (Section V). The paper concludes with an evaluation of ClassyNet on various datasets and testing scenarios (Section VI), and our final conclusions (Section VIII).

II. Background

A. Dynamic Neural Network

Dynamic deep neural networks approaches aim to mainly accelerate the underlying models by allowing them to alter their internal structure or parameters during the inference process by manipulating the network width, depth, or routing during runtime, thereby providing them with enhanced flexibility and superior adaptability to the underlying use case [9]. However, their susceptibility to adversarial attacks targeting resources remains a significant concern [15].

There are different types of dynamic neural networks, each designed to handle different types of input data. Sample-wise dynamic networks are used for processing inputs where the number of samples can vary, such as in speech recognition or natural language processing. These networks are designed to handle variable-length inputs and can adjust their structure based on the number of samples. Spatial-wise dynamic networks, on the other hand, are used for image processing and computer vision tasks. They can handle inputs of varying sizes and adjust their structure accordingly. For example, a spatial-wise dynamic network can process images of different resolutions or aspect ratios. Temporal-wise dynamic networks are used for processing sequential input data, such as in video processing or time-series analysis. They can handle inputs of varying lengths and adjust their structure based on the sequence length.

B. BranchyNet Overview

BranchyNet [11] is a dynamic exit solution that incorporates side branches into the core baseline neural network to permit premature exit of certain input samples. This strategy hinges on the fact that preliminary blocks of the network can accurately predict a sizable chunk of the data set. By facilitating an early exit for these data units, the overall computations performed by the network can be greatly decreased, thereby leading to lower average runtime and energy usage. Our implementation of BranchyNet closely mirrors the variation outlined in [16], specifically designed for IoT deployment

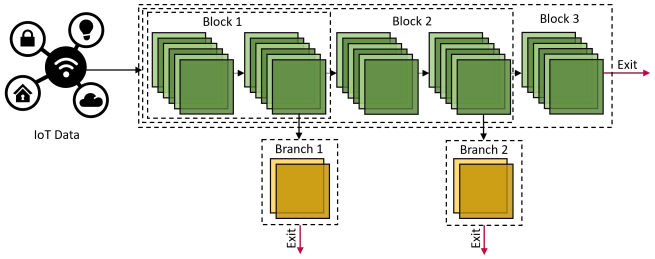


Fig. 1: Overview of BranchyNet architecture.

circumstances. The architecture of BranchyNet, inclusive of how the additional branches augment the foundational network and enable expedited inference, is displayed in Figure 1.

The BranchyNet training regimen involves solving a collective optimization problem predicated on the aggregate of all the classification loss functions tied to each exit point. The loss function steers the learning process by gauging the model's performance in response to the input data. During training, each early branch is allocated a weight to modulate its relative significance, with these weights guiding the model to prioritize certain branches.

In the inference process, the input sample is first fed into the partial network linked to the initial exit. If the exit generates an output that is less than a predetermined threshold, signaling high confidence, a label is attached to the sample, and the inference procedure is halted. Should the sample not pass the exit check, it moves to the subsequent block. This process repeats in an iterative manner until the sample takes an exit at one of the later exit points.

C. Network Intrusion Detection Systems

Network Intrusion Detection Systems (NIDS) scrutinize network traffic to identify and report anomalies based on pre-established detection parameters. NIDS discern intrusive behavior by contrasting it with legitimate user behavior. However, more sophisticated attacks often generate traffic that mimics user behavior, necessitating more intelligent NIDS. There are three primary types of traditional NIDS. Signature-based detection [17] examines network traffic to identify patterns that match known signatures or existing attacks. Anomaly-based detection [18] analyzes network traffic to detect patterns that deviate from normal or baseline behavior. It operates by collecting network traffic samples and applying statistical methods to analyze deviations; when a threshold is surpassed, it alerts the administrator of an anomaly. Lastly, stateful protocol analysis [19] involves comparing network traffic to recognized protocol profiles provided by vendors. This method is used to identify a random sequence of commands in both the network and application layer.

However, each of these methods has its own limitations when dealing with normal and abnormal traffic patterns. These limitations can lead to false positives, false negatives, processing slowdowns, increased CPU usage, and more. Traditional machine learning techniques, such as Naïve Bayes, Decision Trees, and Support Vector Machines, have improved detection accuracy. However, they require expert knowledge and may not perform optimally for multiclass problems with numerous

features. Moreover, self-learning intrusion detection systems and deep learning algorithms have further enhanced detection and classification capabilities. As previously mentioned, deep learning algorithms have shown success in speech recognition, image processing, natural language processing, and others.

III. Motivation

Current state-of-the-art early exit based models, such as BranchyNet, suffer from a limitation in that they are class-agnostic, which hinders their ability to effectively handle edge-specific conditions and contexts. For instance, a single classification application or service may be required to process inputs from classes with varying levels of importance and sensitivity, or to handle imbalanced inputs where a few specific classes make up the majority of the input population. The importance of classes can vary based on the user of the system and the specific use case, as the same classes within the same classification problem may have different degrees of importance due to variations in context. For example, an object classification model installed on a security camera may be trained to detect both humans and vehicles, but in certain situations, detecting humans may become more crucial than vehicles, or vice versa. This runtime-based relative importance is independent of data imbalance problems during training.

NIDS is a prime example of a classification problem that handles categories classes of varying degrees of sensitivity. There are lower priority attacks such as network probing activity and higher priority attacks such as an ongoing exploit for a serious vulnerability activity that has the potential for catastrophic harm. Detecting the ongoing exploit even a few milliseconds early can help the network deploy countermeasures faster and prevent large-scale network damage. The same holds true for personal health monitoring, as recognizing a heart attack is far more crucial than detecting slightly increased blood pressure.

This led us to the development of *ClassyNet*, a framework specifically designed for dynamic early exit classification models that are class-aware, tailored for use in NIDS. These models, particularly when combined with model splitting, offer several advantages. They can prioritize classes of network intrusions based on their severity, pushing high-priority threats such as ongoing serious vulnerability exploits to early exits. This strategy allows NIDS to more effectively meet its operational objectives by minimizing the inference path and thus the computation time.

Moreover, for edge devices with limited memory, inputs from high-priority intrusion classes can be processed using the partial model that resides on-device. This approach avoids the overhead of transmission and allows for faster response times to serious threats. As demonstrated in our results, we can design and train models with exits positioned at the very start of the network model. These models are capable of accurately classifying a significant proportion of the targeted intrusion samples, such as detecting an ongoing exploit even a few milliseconds early, which can help deploy countermeasures faster and prevent large-scale network damage. Therefore, by using early exit techniques, we can construct a neural network model for NIDS by maintaining only a small fraction of the model on the limited memory of the edge device for early inference. Meanwhile, more challenging samples, such as lower priority attacks like network probing activity, can be sent to the cloud for further analysis.

IV. ClassyNet

A. Model Overview

Our ambition is to architect a neural network that modulates its output so that most samples classified at a specific exit point belong to a manually adjustable, predefined set of classes. For instance, in a network intrusion detection system, our design could prioritize rapid responses to high-risk intruders.

ClassyNet diverges from traditional early-exit models by associating a predefined set of classes with each exit, rather than treating all classes equally across all exits. The subset of classes at a given exit includes all subsets from preceding exits and a new subset unique to that exit. The aim is to ensure that any input, belonging to one of these subsets, is likely to exit at the corresponding point. The allocation of class sets to each exit point is performed manually by the system operator at the start of the training process. We assume that the operator is somehow aware of which classes that require prioritization.

In order to adjust the traditional early exit models to favor specific classes at dedicated exits, we need a novel mechanism embedded in the neural network training procedure that is capable of promoting the classification of the subset classes with higher confidence. For this purpose, we introduce two complementary techniques to enhance early exit models:

- **Bag-of-Classes (BoC)**. In this approach, we consolidate all non-desired classes into a single class, referred to as the class bag. Samples belonging to non-desired classes are treated as one super-class, which is prevented from exiting the network at the designated exit. The objective is to train the network to first identify the boundaries between the desired classes, while deferring the non-desired classes to later exits. This method draws inspiration from binarization approaches commonly used in multi-class classification [20]. In the evaluation section, we refer to this approach as *ClassyNet* or *CN*.
- **Cost-sensitive loss matrix (CS)**. Here we use a cost matrix C during training. In this matrix, each pair of true and predicted labels is assigned a specific value (weight), which is then multiplied by its raw loss value before being reduced across the entire batch. This method allows us to assign higher penalties to errors within certain classes, thereby emphasizing the importance of correct classification of a given class at any specific exit of the neural network. Our work is highly inspired by [21] and other literature on instance-level and class-level costs for the loss function. Cost-sensitive learning is most commonly used for imbalanced datasets, where cost is seen as a penalization factor for mistakes made in minority classes. However, we argue that cost-sensitive approaches can be used beyond that in order to model importance among classes. Additionally, this cost

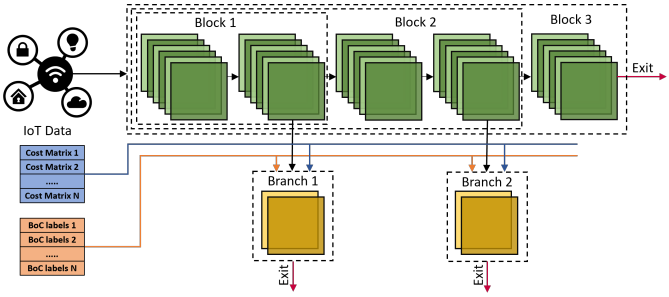


Fig. 2: ClassyNet model architecture highlighting the integration of an additional cost matrix and the process of sample relabeling within the supplementary branches.

matrix approach can be used in conjunction with Bag-of-Classes approach to achieve class-based early exit. In the evaluation section, we refer to the combination of the two approaches as *ClassyNet+* or *CN+*.

Figure 2 provides an overview of the ClassyNet model architecture. figure highlights the addition of the set of bag labels, which is necessary for the Bag-of-Classes approach, and the integration of the cost matrix, which is essential for the Cost-sensitive loss matrix. Moreover, the figure illustrates how both approaches can be seamlessly incorporated within the fundamental multi-exit architecture.

B. ClassyNet Training

The core of the training process relies on the loss function, acting as the primary performance evaluator for the model in relation to the provided input data. Serving as a navigational tool, the loss function assists in gauging the model's predictive accuracy based on the given input. The ClassyNet training procedure employs a joint training method anchored in the resolution of a single optimization problem, combining the outputs from all exits.

To better understand the composition of the loss function, let's introduce some notations. We denote an input sample as x, while its corresponding one-hot encoded label is represented by y. The predicted label of x at exit n is denoted as y'_n. We use K_n to specify the number of classes within the Bag-of-Class assigned to exit n, and N to denote the total number of exits. The training regimen commences by propagating the input x through the network, subsequently generating the output probability vector p_n at each individual exit n. Here

$$p_n = f_{exit_n}(x) \tag{1}$$

where f_{exit_n} is the function representing the transformation of sample x at any given exit n. p_n has a number of values equal to the number of classes K_n and indicates the likelihood that the sample belongs to any given class. p_n is then normalized by passing through a softmax layer to obtain the output logit vector $\hat{y}_n$ as shown in Eqn. 2 where $\|\|_1$ is the L1 norm operator.

$$\hat{y}_n = softmax(p_n) = \frac{exp(p_n)}{\|exp(p_n)\|_1} \tag{2}$$

$\hat{y}_n$ is then element-wise multiplied with the true label vector y resulting in a scalar value that corresponds to the correct class. This scalar value is then multiplied by a constant obtained from the cost matrix C_n assigned to exit n, which depends on the actual label y and the predicted label y'_n. The constant is used to weight whether the prediction is correct and whether the sample is exiting from its designated exit. To determine the predicted label y'_n, we select the class that has the highest probability from the softmax output. The final product of this process provides the loss value for a given sample x. When this operation is performed for every sample in a batch and the results are summed, it yields the total loss at the exit as shown in Eqn. 3.

$$L_{exit_n}(y, \hat{y}) = -\sum_{x \in batch} C_n(y, y'_n) y log \hat{y}_n \tag{3}$$

,

Our training method has a unique characteristic: if a sample from a batch decides to exit at any point during the training process, it will not proceed to subsequent exits. Instead, we assign a zero weight to its corresponding value in the cost matrix. This strategy diverges from conventional methods outlined in literature, but we found it yielded superior accuracy. To determine if a sample should exit, we compute the cross-entropy of the sample and compare it against a pre-defined threshold specific to that exit.

Finally, The global loss then is computed by multiplying the loss derived from each exit and the weight assigned to that exit. This is followed by the application of back-propagation to optimize the value of this global loss. Given that, the overall loss function for ClassyNet becomes

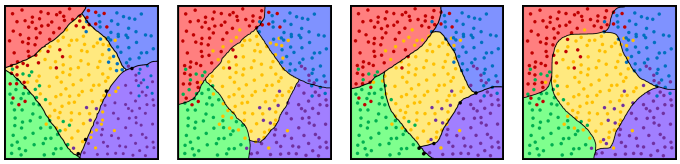
$$L_{ClassyNet} = \sum_{n=1}^{N} w_n L_{exit_n} \tag{4}$$

C. ClassyNet Inference

The inference process of ClassyNet can be summarized as follows. The classification network of ClassyNet initiates by processing the sample through the initial exit of the network model, which includes the first block of the network and the branch of the first exit. This generates a normalized probability vector, whose cross-entropy is immediately calculated and compared with a predefined threshold. If the computed cross-entropy is lesser than this threshold, a label is assigned to the sample, thereby concluding the inference procedure.

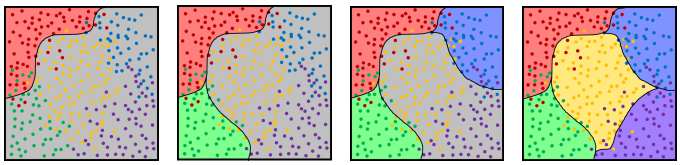
If we are using Bag-of-Classes, the process is slightly altered. Termination only occurs if the generated label is among the set of classes designated to the specific exit. If the label is associated with classes not assigned to the exit, the sample remains within the network, because further processing is required to identify its precise class.

The thresholds of the exits are assigned before the start of the inference process. They are using to determines the conditions for terminating the process at that exit. If a sample does not meet the threshold conditions for any reason, it continues through the network to the next block. This process repeats until the sample either meets an exit threshold or reaches the final exit, at which point it must exit. These thresholds control the balance between runtime and accuracy. Setting higher entropy thresholds may increase the number of early exits, but at the cost of reduced overall model accuracy.



(a) (b) (c) (d)

Fig. 3: Base early-exit neural network.



(a) (b) (c) (d)

Fig. 4: ClassyNet using Bag-of-Classes (BoC). Combined set of classes is grey shaded.

D. ClassyNet Visual Example.

To better understand the operation of ClassyNet classification and its distinction from standard early-exit models, we present an example using a synthetic dataset. This dataset comprises five different classes (Red, Yellow, Green, Blue, and Purple) and a neural network with four total exit points. To illustrate the difference in how decision boundaries evolve between the traditional early-exit approach and the ClassyNet approach, we have made two set of figures (Figure 3 and Figure 4) corresponding to the change in decision boundaries over the different exits for both approaches.

The set of figures in Figure 3 demonstrates the evolution of the classification boundaries of the standard early-exit model across the different exits of the network. Figure 3a presents relatively simplistic decision boundaries for all classes. These boundaries start to improve in subsequent exits (Figures 3b and 3c), as the network learns to differentiate and separate between different classes more effectively. By the final exit (Figure 3d), the network achieves satisfactory decision boundaries for all classes. From the figures, We can observe that each exit deals with a complex multi-class decision boundary, making it challenging to achieve high accuracy for specific targeted classes at earlier exits. This observation suggests potential for the BoC approach to simplify early exit classification tasks.

On the other hand, the set of figures in Figure 4 shows the evolution of the classification boundaries using the ClassyNet model with the BoC implementation. In this scenario, the Red class is assigned to the first exit, the $Green$ class is assigned to the second exit, the $Blue$ class is assigned to the third exit, and the remaining classes are assigned to the last exit. These assignments are all incremental.

Figure 4a shows a significantly improved decision boundary (compared to Figure 3a) that focuses on discriminating the assigned class (Red) from all others. This approach simplifies the initial complex multi-class classification problem into a binary classification, leading to better boundary representation and higher accuracy. Similar observations apply to Figure 4b, where the assigned class ($Green$) has a clearer decision boundary compared to Figure 3b. Finally, Figure 4d is identical to Figure 3d, as it considers all four classes at the end.

To incorporate the cost matrix in our example, and to understand how it is populated, we consider four different cases: correct label for a targeted class, incorrect label for a targeted class, correct label for a non-targeted class, and incorrect label for a non-targeted class. We assign a set of weight costs c_1, c_2, c_3, and c_4 for each case, respectively. Assuming that we want to train one of the exits to target samples from two classes, for example $Blue$, and $Green$, and become more capable at classifying samples from these two classes. The weight values can be chosen to direct the training process to favor these two targeted classes. For example, c_1 should always be lower than c_2 to incentivize correct classification. The same applies to c_3 and c_4 is the same. Since we want to encourage more samples from the first two classes to exit at this exit, c_1 and c_2 should also be lower than c_3 and c_4 because the former two weights represent the targeted classes while the latter two are for non-desired classes.

V. Datasets

Effective network security analysis depends heavily on the quality and selection of data. The suitability and practical use of data are crucial factors in conducting successful research. The size of data also affects the performance of machine learning and deep learning models. In this paper, we selected three different datasets - NSL-KDD, CICIDS17, and UNSW-NB15 - to test the effectiveness of our approach. In the following sections, we provide a brief overview of each dataset in terms of its sample size, features, and other relevant characteristics.

A. KDD CUP99 Dataset

The KDD Cup99 dataset [12] is a widely used resource for network intrusion detection. It was developed for the Classifier-Learning Competition in 1999 and is based on DARPA datasets. The KDD Cup99 dataset consists of approximately 4.9 million network connection records, which are categorized as normal or anomalous. Anomalous connections are further classified into four categories: Denial of Service (DoS), User to Root (U2R), Remote to Local (R2L), and Probing attacks. Normal connections represent 97% of the total data, while anomalous connections represent the remaining 3%. Each network connection record contains 41 features, including basic features of network connections such as source IP address, destination IP address, source port number, destination port number, protocol type, and other attributes that are derived from these basic features. These derived features include the number of failed login attempts, the number of compromised user accounts, and the number of root accesses. Of the 41 features, 34 are continuous and 7 are categorical. The dataset is highly imbalanced, with a small proportion of the data corresponding to the different types of attacks.

B. NSL-KDD

The KDD Cup99 dataset carries statistical issues which lead to poor approximation and estimation [12]. Those issues were addressed in the NSL-KDD [22] dataset. The NSL-KDD dataset contains 41 features, including basic features of

network connections such as source IP address, destination IP address, source port number, destination port number, protocol type, and other attributes that are derived from these basic features. These derived features include the number of failed login attempts, the number of compromised user accounts, and the number of root accesses. The dataset contains 23 different attack types, which are classified into four main categories: denial of service (DoS), probing, user-to-root (U2R), and remote-to-local (R2L).

C. UNSW-NB15

The UNSW-NB15 dataset [13] is a relatively newer dataset compared to other datasets like KDD Cup99. It was developed in 2015 by researchers from Australian Centre for Cyber Security. It contains approximately 2.5 million network connection records categorized as normal or anomalous, with 49 features containing both flow and packet-based features. Those features are further divided into four different categories, namely: content, basic, flow, and time. It includes nine different categories of attacks, as well as a set of "unknown" connections. The dataset was designed to provide more realistic and diverse network traffic data, with a focus on capturing traffic from different types of attacks. Its size and diversity make it a valuable resource for researchers in the field of network intrusion detection.

D. CICIDS17

The CICIDS17 [14] dataset is a publicly available benchmark dataset for network intrusion detection, created by researchers at the Canadian Institute for Cybersecurity in 2017. It contains over 18 million network traffic records, captured in a campus network environment, and is designed to represent a variety of realistic attack scenarios. The dataset consists of 80 features, categorized into seven different classes: basic features, content features, time-based features, host-based features, flow-based features, and attack categories. The attack categories are divided into four different classes: denial of service, reconnaissance, network infiltration, and exfiltration. One of the key features of the CICIDS17 dataset is its focus on newer and more sophisticated attack types, including botnet attacks, IoT-based attacks, and web application attacks.

VI. Evaluation

In this section, we will evaluate the performance of ClassyNet versus Baseline neural network and BranchyNet under different testing scenarios. We evaluate the success of the ClassyNet model in assigning different priorities to different classes and causing them to leave from the exits they are assigned to. We also evaluate ClassyNet impact on Accuracy, Inference time on Edge and non-Edge environments. All the results presented in this section were obtained from training and evaluating each scenario, averaged over ten runs.

A. Experiment Design

1) Classification models.: For our experiments, we use a 1d convolutional neural network [23] design consisting of 5 conv1d layers. The first layer is the input layer and it is of varying size to match the size of the incoming traffic. It is followed by the 1d convolutional layers aiming to build abstraction to understand the underlying structure of the network traffic. The last layer is a dense layer corresponding to the number of outputs. We use categorical cross entropy loss in addition to the Adam optimizer to help alleviate overfitting. The learning rate is 0.005.

This model provides the backbone classifier for both the BranchyNet and ClassyNet models. Our augmented models has 3 exits positioned at layers #2, #5, #10 respectively. The implementation for both BranchyNet and ClassyNet was built on top of Intel Labs distiller framework [24]. All our models were trained for 1000 epochs and run multiple times to obtain and the values of our hyper-parameters including the cost matrix were obtained through experimentation.

We developed two models of ClassyNet for the following set of experiments; ***ClassyNet*** and ***ClassyNet+***. ***ClassyNet*** makes use of the *Bag-of-Classes* approach in training the model. While ***ClassyNet+*** uses a combination of both *Bag-of-classes* and *Cost-sensitive loss matrix*. As mentioned in Section V, we evaluated ClassyNet on all 3 dataset. However, due to space limitation, we usually showcase the results from one or two datasets if all three are exhibiting the same behavior.

2) Evaluation platform.: Our local experiments utilized Python 3.7 and PyTorch 1.14, while training and evaluating all models on a workstation equipped with an AMD Ryzen 7900X CPU, two Nvidia RTX 4090 GPUs, and 256GB of memory. While for our Edge experiments, we used Nvidia Jetson TX2 equipped with an ARM Cortex A-57 CPU and 8GB of RAM. It also has a 256-core Nvidia Pascal GPU architecture with 256 NVIDIA CUDA cores. To simulate a lower-end configuration, we disabled the GPU in some experiments.

3) Performance metrics.: In assessing the performance of our ClassyNet in comparison to BranchyNet, we are considering the following aspects:

- ***Exit Efficiency.*** We evaluate how many of samples belonging to targeted classes exited from the assigned exits and use that to assess the exit efficiency of the model. Additionally, we also evaluate how many of these samples exited at subsequent exits to evaluate the impact further down the model.
- ***Latency Time.*** We compute the total computation time for all the testing data, and we use it as an indicator for the latency for this set of experiments. In the upcoming experiments that solely simulate edge environment, we add the appropriate communication latency.
- ***Accuracy.*** We compare the overall classification accuracy of both ClassyNet and BranchyNet over the multiple datasets over all classes.
- ***Edge Deployment*** We test ClassyNet against other model in an edge environment. The objective of the experiments is to evaluate the practicality and efficiency of ClassyNet when we deploy it on edge devices with limited resources. The main resources we are focusing on are the memory

		Normal			DoS (PC2)			Probe (PC1)			R2L			U2R		
NSL-KDD		Exit #1	Exit #2	Exit #3	Exit #1	Exit #2	Exit #3	Exit #1	Exit #2	Exit #3	Exit #1	Exit #2	Exit #3	Exit #1	Exit #2	Exit #3
	Base	0	0	22447	0	0	17383	0	0	3225	0	0	307	0	0	224
	BN	5476	4573	12398	4415	3146	9821	645	728	1852	86	93	127	33	50	141
	CN	0	0	22038	0	12402	5491	1072	1195	957	0	0	310	0	0	172
	CN+	0	0	22724	0	13807	4291	1204	1271	750	0	0	313	0	0	204
		Normal			**DDoS**			**FTP-Patator (PC1)**			**PortScan (PC2)**			**Brute Force**		
CICIDS17	**Base**	0	0	162910	0	0	25891	0	0	1672	0	0	32182	0	0	1098
	BN	67918	32391	61194	10267	5528	9641	614	313	558	12872	6541	11512	483	264	413
	CN	0	0	16391	0	0	25192	857	502	325	0	25423	7271	0	0	912
	CN+	0	0	16573	0	0	26241	945	591	212	0	28172	4376	0	0	1041

TABLE I: The distribution of the classes of NSL-KDD and CICIDS17 datasets over exits using all models. PC1 indicate the first priority class, PC2 indicate the second priority classes. Classes 'Sql-inject' and 'XSS' has been omitted from CICIDS17 due space limitation.

requirements for model deployment and the latency of the inference computation.

B. Experiment Results

1) Results - Exit Efficiency: Table I shows the class distributions of the NSL-KDD dataset under the different models. The experiments show similar properties when used on the CICIDS17 dataset and the UNSW-NB15, which we have to omit due to space limitation. For the base model, no samples exit at either Exit #1 or #2 since it has no early exit points. For BranchyNet, the table shows similar distribution for each class at the different exits with 24% leaving at Exit #1, 17% leaving at Exit #2, and 59% leaving at the last Exit. For the ClassyNet experiments, for the NSL-KDD dataset, we assigned the class Probe as priority class #1 and allowed it to exit starting from Exit #1. We assigned the class DoS as priority class #2 and allowed to exit starting from the second Exit #2. Similarily for the CICIDS dataset, we assigned FTP-Patator as priority class #1 and PortScan as #2. Looking at the ClassyNet distribution, we notice a sizeable improvement in the number of samples of the priorities classes assigned to a specific exit, as the number of samples correctly classified from the priority classes increases by up to 68% in some classes. This clearly shows the viability of our approach in developing class-aware models that significantly increase the number of samples of only specific classes exiting at specific exits. Finally, the ClassyNet+ distribution shows that adding the cost matrix to the training process further increases the percentage of the assigned classes leaving by up to 90% compared to BranchyNet, as they are heavily incentivized to exit from their assigned exits because of the additional penalty associated with leaving from later exits.

2) Results - Latency Time.: To properly evaluate the inference time of the class based classifer, we created multiple synthetic testing data containing different percentages of the targeted classes. Figure 5 shows the latency (inference) time of the four models for different compositions of testing data and a total size of 10k samples using the NSL-KDD dataset. Figure 6 show the results for the UNSW-NB15, which exhibit similar behavior to the NSL-KDD dataset. Due to space limitation, we will focus on discussing NSL-KDD results. We start with synthetic testing data containing 10% of the priority classes, we gradually increase the percentage of samples belonging to these classes till they reach 90% of the total testing data.

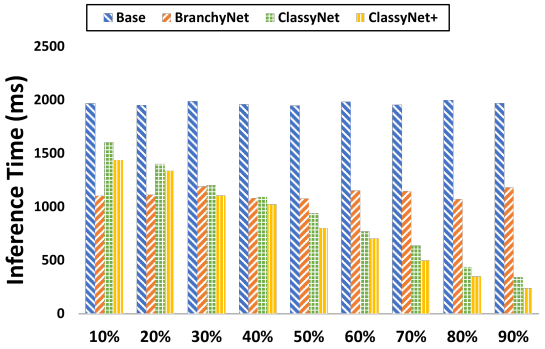


Fig. 5: Total inference time of 10000 samples over different compositions of the NSL-KDD dataset.

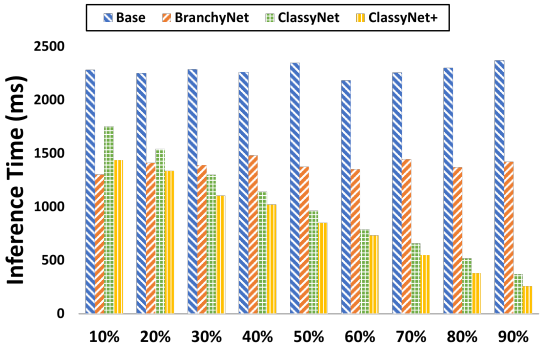


Fig. 6: Total inference time of 10000 samples over different compositions of the UNSW-NB15 dataset.

The figure shows that the latency time of both the *Baseline* and *BranchyNet* models is constant, which is to be expected given that both of these models are independent of any manipulation of testing data. *BranchyNet* does shows a lower latency time than the *Baseline* model. This is to be expected as BranchyNet allows for samples to leave from earlier exits thus lowering the inference time. We can also observe that *BranchyNet* initially outperforms *ClassyNet* in total latency time when the percentage of the targeted classes is small. This is because *ClassyNet* limits the early exits to only specific classes and forces others to later exits, as opposed to the more open exiting criteria of *BranchyNet*. *ClassyNet+* shows more reduction in latency time because of the cost matrix highly penalizing samples leaving from later exits.

As the composition of the data changes, with more samples belonging to the target class, the performance gap between *BranchyNet* and textitClassyNet begins to decrease until

ClassyNet overtakes *BranchyNet*. This happens when around 30% of the samples belong to the targeted classes. This shows that capabilities of ClassyNet, and how class-aware classifier can target classes with a high success rate. For example, when the target class represents 90% of the entire testing samples, the total inference time of *ClassyNet* is about 200ms, which is only 35% of the corresponding time of *BranchyNet* . Additionally, we calculate the total inference time for multiple synthetic dataset of 5000 samples, with all the samples being from priority classes, or randomly drawn from all the different classes. The results are shown in Table II for both NSL-KDD and CICIDS17 dataset. They clearly show that ClassyNet outperforms the Baseline and BranchyNet when dealing with data from targeted classes with ClassyNet as it was able to classify the synthetic set in 3.57x faster than BranchyNet.

	NSL-KDD			CICIDS17		
	PC1	**PC2**	**All Classes**	**PC1**	**PC2**	**All Classes**
Base	1011ms	991ms	1018ms	5268ms	5332ms	5556ms
BN	551ms	574ms	581ms	2885ms	2942ms	3177ms
CN	193ms	301ms	823ms	1046ms	1642ms	4362ms
CN+	154ms	287ms	801ms	837ms	1587ms	4495ms

TABLE II: Total inference time for 5000 samples given that all samples are from the PC1, PC2, or randomly drawn from the dataset. Results are shown for NSL-KDD and CICIDS17 datasets.

3) Results - Accuracy.: Table III summarizes the classification accuracy of Baseline, BranchyNet, ClassyNet, and CLassyNet+ models under all datasets. From the table, we observed that both early-exit models have a decrease of less than 1% in accuracy compared to the Baseline model. However, this decrease is minimal and could be ignored. Moreover, we can observe that ClassyNet has an additional minor reduction in accuracy compared to BranchyNet. However, ClassyNet+ shows a smaller reduction in accuracy compared to ClassyNet, we attribute this to the normalizing factor of the cost matrix. However, all the changes in accuracy among different models are very minor as they all fall within 1% of baseline accuracy. We attribute this slight accuracy reduction to the fact that ClassyNet targets traditionally difficult samples of the target classes and attempts to adjust the model to finalize the inference process on these samples in earlier exits.

	NSL-KDD	**CICIDS2017**	**UNSW-NB15**
Baseline	97.61%	96.89%	94.65%
BranchyNet	97.31%	96.52%	94.18%
ClassyNet	97.13%	96.38%	93.97%
ClassyNet+	97.24%	96.42%	94.11%

TABLE III: Accuracy results for the different models under different architectures and datasets.

4) Results - Edge Deployment.: To test the system under different memory configuration, we implemented a model partitioning scheme. Model partitioning is a common technique with multi-exit models, where a small part of the model is deployed on the edge usually ending with an optional exit and the remainder of the model is deployed on the cloud and gets to process samples that the smaller part could not classify. Additionally, we perform our testing using two configuration of synthetic datasets; Uniform datasets use randomly drawn samples from the full dataset and mimic the distribution of the traffic. Biased datasets are more loaded with samples from high priority classes. The results are shown in Table IV. The first row in the table represent a full deployment on the edge. While the subsequent rows show the memory footprint taken by the partial model left on the edge. The memory sizes correspond to the memory required to deploy the set of layers associated with the various blocks of both BranchyNet and ClassyNet. From the table, we observe that for BranchyNet, the total latency increase with with lower memory sizes for both the Uniform and Biased test cases as more samples need to be sent forward to the cloud for further processing due to the lowering computational capabilities at lower memory sizes. However, for ClassyNet, we observe that this patterns holds true as well. However, for the Biased test, when the majority of the testing samples belong to priority classes, the overall inference remains considerably low compared to BranchyNet.

	BranchyNet		**ClassyNet**		**ClassyNet+**	
Mem	Uniform	Biased	Uniform	Biased	Uniform	Biased
6.8M	544	586	876	174	740	232
1.6M	1900	2486	5836	860	4670	338
0.33M	4730	5116	7752	1410	7186	590
0.15M	6298	6472	8692	1722	8336	904

TABLE IV: Inference Time of different synthetic dataset of 1000 samples under model splitting scenario. Uniform test cases use randomly drawn samples from the dataset. Biased test cases has 80% drawn from priority classes.

VII. Related Work

Network intrusion detection is the process of detecting unauthorized and malicious activities in a computer network. In recent years, neural networks have become popular for network intrusion detection due to their ability to learn complex patterns and classify them with high accuracy. In this related works section, we review the literature on network intrusion detection using neural networks. Several studies have explored the use of neural networks for network intrusion detection. For example, in [25], a neural network-based intrusion detection system was proposed that could classify network traffic into three categories: normal traffic, known attacks, and unknown attacks. The system achieved high detection rates for both known and unknown attacks. Another approach to network intrusion detection using neural networks was proposed in [26]. The authors used a self-organizing map (SOM) neural network to cluster network traffic and identify anomalous behavior. The system achieved high accuracy and was able to detect both known and unknown attacks. In [27], the authors proposed an intrusion detection system based on deep learning using a stacked autoencoder neural network. The system was trained on the UNSW-NB15 dataset, which contains a variety of attack types, and achieved high detection rates for both known and unknown attacks. More recently, [6] proposed a convolutional neural network-based intrusion detection system that achieved high detection rates on the NSL-KDD dataset. [28] proposed a system that combined deep learning and clustering for anomaly-based intrusion detection and achieved high accuracy on the KDD Cup 99 dataset. Additionally, [7]

proposed hybrid intrusion detection systems that combined multiple neural network architectures to achieve high detection rates on various datasets.

All of these papers deal with class-agnostic network intrusion system with no ability to assign different priorities to different threats. Unlike our proposed solution, ClassyNet, which is designed to provide a class-aware early-exit model capable of dealing with some unique challenges in edge environments.

VIII. Conclusion

In this paper, we designed and developed *ClassyNet*, the first dynamic class-aware classification model for edge devices with limited resources that significantly reduces inference latency time in supporting real-time applications by allowing different priorities to be assigned to different classes and enabling classes of higher priority to finish the inference earlier. We detailed the architecture and design details of the proposed ClassyNet, which included two novel additions to early-exit models; *Bag-of-Classes* and *Cost-sensitive loss matrix* to enable class-aware training.

We demonstrated its application in network intrusion detection and showed that this approach can significantly improve the detection of high priority threats while reducing the computational overhead. We compared several performance metrics of ClassyNet vs BranchyNet under different sets of testing data, exit numbers, and testing environments. Furthermore, we compared ClassyNet's performance on edge devices with varied memory capacity limits to that of BranchyNet and two network pruning strategies. According to the results, ClassyNet could achieve up to 4x quicker inference latency time than the nearest model of comparable techniques. We are researching several directions in our future work including developing a mechanism that automatically assigns classes to the ClassyNet's many exits in the most efficient way. In addition, we are looking at how to utilize ClassyNet with imbalanced data. We believe that this work will spark new and exciting research on class-aware classification.

References

[1] Z. Li, W. Yang, S. Peng, and F. Liu, "A survey of convolutional neural networks: Analysis, applications, and prospects," 2020.

[2] M. Alam, M. D. Samad, and et. al., "Survey on deep neural networks in speech and vision systems," 2019.

[3] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning in natural language processing," 2019.

[4] H. B. Pasandi and T. Nadeem, "Towards a learning-based framework for self-driving design of networking protocols," *IEEE Access*, vol. 9, pp. 34 829–34 844, 2021.

[5] S. K. Nukavarapu, M. Ayyat, and T. Nadeem, "Miragenet - towards a gan-based framework for synthetic network traffic generation," in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, Dec 2022, pp. 3089–3095.

[6] X. Zhang, J. Ran, and J. Mi, "An intrusion detection system based on convolutional neural network for imbalanced network traffic," in *2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT)*, 2019, pp. 456–460.

[7] M. S. ElSayed, N.-A. Le-Khac, M. A. Albahar, and A. Jurcut, "A novel hybrid model for intrusion detection systems in sdns based on cnn and a new regularization technique," *Journal of Network and Computer Applications*, vol. 191, p. 103160, 2021.

[8] S. Nukavarapu and T. Nadeem, "iknight–guarding iot infrastructure using generative adversarial networks," *IEEE Access*, vol. 10, pp. 132 656–132 674, 2022.

[9] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, "Dynamic neural networks: A survey," *arXiv preprint arXiv:2102.04906*, 2021.

[10] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Weinberger, "Multi-scale dense networks for resource efficient image classification," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=Hk2aImxAb

[11] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 2464–2469.

[12] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 2009, pp. 1–6.

[13] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, 2015, pp. 1–6.

[14] I. Sharafaldin, A. Habibi Lashkari, and A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," 01 2018, pp. 108–116.

[15] M. Ayyat, S. K. Nukavarapu, and T. Nadeem, "Dynamic deep neural network adversarial attacks for edge-based iot devices," in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, Dec 2022, pp. 61–67.

[16] S. K. Nukavarapu, M. Ayyat, and T. Nadeem, "ibranchy: An accelerated edge inference platform for iot devices," in *The Sixth ACM/IEEE Symposium on Edge Computing*, ser. SEC '21. New York, NY, USA: ACM, 2021. [Online]. Available: https://doi.org/10.1145/3453142.3493517

[17] A. Ganesan, P. Parameshwarappa, A. Peshave, Z. Chen, and T. Oates, "Extending signature-based intrusion detection systems withbayesian abductive reasoning," 2019.

[18] B. Molina-Coronado, U. Mori, A. Mendiburu, and J. Miguel-Alonso, "Survey of network intrusion detection methods from the perspective of the knowledge discovery in databases process," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2451–2479, dec 2020. [Online]. Available: https://doi.org/10.1109\%2Ftnsm.2020.3016246

[19] S. Pal, Y. Gupta, A. Kanade, and S. Shevade, "Stateful detection of model extraction attacks," 2021.

[20] J. Fürnkranz, "Class binarization," in *Encyclopedia of Machine Learning and Data Mining*, C. Sammut and G. I. Webb, Eds. Springer, 2017, pp. 203–204.

[21] C. Zhang, K. C. Tan, H. Li, and G. S. Hong, "A cost-sensitive deep belief network for imbalanced classification," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 30, no. 1, pp. 109–122, 2019.

[22] S. Revathi and A. Malathi, "A detailed analysis on nsl-kdd dataset using various machine learning techniques for intrusion detection," *Int. Journal of Engineering Research and Technology*, vol. 2, 2013.

[23] S. Kiranyaz, O. Avci, and et. al., "1d convolutional neural networks and applications: A survey," 2019.

[24] N. Zmora, G. Jacob, L. Zlotnik, B. Elharar, and G. Novik, "Neural network distiller: A python package for dnn compression research," October 2019. [Online]. Available: https://arxiv.org/abs/1910.12232

[25] O. Linda, T. Vollmer, and M. Manic, "Neural network based intrusion detection system for critical infrastructures," in *2009 International Joint Conference on Neural Networks*, 2009, pp. 1827–1834.

[26] Z. Liu and Y. Lai, "A data mining framework for building intrusion detection models based on ipv6," in *Advances in Information Security and Assurance*, J. H. Park, H.-H. Chen, M. Atiquzzaman, C. Lee, T.-h. Kim, and S.-S. Yeo, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 608–618.

[27] N. Moustafa, J. Slay, and G. Creech, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *Military Communications and Information Systems Conference (MilCIS)*. IEEE, 2015, pp. 1–6.

[28] E. Tufan, C. Tezcan, and C. Acarturk, "Anomaly-based intrusion detection by machine learning: A case study on probing attacks to an institutional network," *IEEE Access*, vol. 9, pp. 50 078–50 092, 2021. [Online]. Available: https://doi.org/10.1109\%2Faccess.2021.3068961