Dynamic Deep Neural Network Adversarial Attacks for Edge-based IoT Devices

Mohammed Ayyat, Santosh Kumar Nukavarapu, and Tamer Nadeem
Department of Computer Science
Virginia Commonwealth University
Richmond, VA 23220, USA
{ayyatma, nukavarapuskk, tnadeem}@vcu.edu

***Abstract*—Edge-based IoT devices have experienced phenomenal growth in recent years due to rapidly increasing demand in various emerging applications which typically utilize machine learning (ML) models such as Deep Neural Network (DNN) and demand low latency and low power consumption. To support the edge requirements, ML models have to support faster inference and less computation. Dynamic DNNs (D^{2}NN) have been proposed to support low-latency and power-saving on edge devices by enabling conditional computations and context dependant activation of the network model for inference; saving computational time and edge resources, hence they are becoming popular for edge applications. In this paper, we show that D^{2}NN are vulnerable to our novel adversarial attack, Dynamic DNN Adversarial attacks (DDAS). Unlike conventional adversarial attacks that target classification accuracy, DDAS targets the IoT device resources such as the battery, latency, and so on. We show that our attack is effective under various attack scenarios with a high attack success rate. We also provide a retraining scheme as a countermeasure to DDAS and show its effectiveness.**

***Index Terms*—Adversarial Attack, Classification, IoT**

I. Introduction

With the phenomenal growth of IoT devices at the edge, many applications are being enabled, such as remote health care, augmented reality, and video analytics. These applications demand low latency, privacy, and security, necessitating the employment of supporting features such as on-device computations, privacy-preserving data analytics, and anomaly detection. Therefore, IoT devices typically host on-device machine learning models such as DNNs to support these applications. However, the performance of DNNs when running on edge-based IoT devices is significantly influenced by device resource constraints. As a result, developing techniques to optimally accelerate the inference computations of DNN models in order to achieve real-time applications while conserving energy for edge devices/IoT is very desirable.

Some popular accelerating techniques for model inference include designing lightweight DNNs with fewer layers [1] or pruning an existing DNNs by removing parts of the neural network that have minimal impact on the inference process [2]. While these models showed relatively faster inference and less resource-constrained performance during inference, they are not flexible enough to be further optimized or to be adaptable to the conditions of the environment. For example, these models do not have the ability to terminate the inference early if the model has sufficient confidence to infer the outcome.

Recently, different type of DNNs referred to as Dynamic DNNs (D^{2}NNs) have been proposed [3] to enable low latency and power savings on IoT devices. In contrast to traditional DNNs, dynamic DNNs are capable of performing conditional computations and selectively activating context dependant portions of the network model. This includes the ability to terminate computation and infer early when the earlier layers of the network have sufficient confidence without requiring all subsequent layers to participate in computation [4], or skipping the input space models [5], [6], which conditionally scan only the sufficient spatial or temporal input space for inference and thus reduce the network's unnecessary processing and computation time. These D^{2}NN models, which support characteristics like on-demand computing, hardware adaptability, and fewer resource constraints, are therefore gaining popularity for constructing and developing high-performance IoT/edge applications such as low-latency edge analytics [7] and real-time object detection [5].

Adversarial attacks on machine learning DNN models have increased significantly, and have become a challenge given their realistic and dangerous attack scenarios. In an adversarial attack, the attacker performs perturbations to the input or environmental conditions in order to target the model's accuracy. Given that D^{2}NN models are expected to be heavily exploited and utilized in many edge-based scenarios, understanding the vulnerabilities of these dynamic models to adversarial attacks is very critical. Furthermore, techniques for improving the robustness of the D^{2}NN models to counter these adversarial attacks must also be explored.

In this paper, we present the Dynamic Deep neural network Adversarial Attack (DDAS) for edge-based D^{2}NN models that aim to defeat their objective of early inference or conditional computation. To the best of our knowledge, our work is the first attempt at developing adversarial attacks and mitigation techniques for D^{2}NN models. More specifically, we design and develop ***DDAS-EarlyExit*** attack for D^{2}NN early-exit based models such as BranchyNet [4] using a GAN-based approach. We evaluate DDAS-EarlyExit using various attack metrics under different attack scenarios. Moreover, we implement and evaluate a robust incremental training approach for building resilient D^{2}NN using the adversarial samples generated from our attack model.

This material is based upon work partially supported by the US National Science Foundation under Grants No. CNS-1764185 & OAC-2212424.
*All authors have contributed equally.

978-1-6654-3540-6/22

II. Background

A. BranchyNet

BranchyNet is a dynamic DNN solution that allows input samples to exit the network early by adding side branches to the baseline network branch. This concept is based on the observation that the earlier layers of the network can correctly predict a large portion of the data population. Allowing these data points to perform an early stopping and exit the network early will reduce the network's overall computations, bringing down the average runtime and power consumption. We are using a variation of BranchyNet termed iBranchy [8] that is geared for IoT deployment scenarios. Figure 1 shows an example of an iBranchy architecture exhibiting a model augmented with two extra branches.

The training process of iBranchy is performed by solving a joint optimization problem on the weighted sum of all the classification loss functions associated with the individual exit points. The loss function serves as a method of evaluating how the model is performing given the input data, thus it is used to guide the learning process. Each branch is assigned a weight during training to control its relative importance. These weights are used to direct the model into favoring specific branches. During training, by default the samples pass through all exits even if they would have terminated at an earlier exit. However, the training process can be configured to prevent the samples from passing over to latter exits if they are to be terminated at earlier exits. We have chosen to use the first approach in our implementation.

The inference process starts by passing the sample through the initial block consisting of the internal layers of the network up to the branch of the first exit producing a vector representing the classification likelihood of the sample, the vector is then normalized using $softmax$ function. The cross-entropy[1] value of the normalized probability vector is then computed at the exit point. If the value of the entropy is lower than a predefined threshold, a label is attached to the sample and the inference process terminates. Note that each of the exits is assigned a threshold prior to the inference process that defines its terminating condition. If the sample fails the exit check, it is forwarded to the next block for further processing and iteratively attempts to exit at each of the subsequent exit points.

B. Generative adversarial networks (GAN) overview

The GAN architecture consists of a generator and a discriminator. The generator takes in random noise as input to produce an adversarial sample mimicking the distributions of real samples. The quality of the adversarial sample is measured by the discriminator to decide whether the sample is genuine or not. GAN adversarial training can be thought of as a min-max problem to minimize the generator loss and maximize the discriminator loss during training. In a recent extension of the GAN for adversarial attacks on conventional DNNs, the authors extended the GAN architecture with a feed-forward network representing the target model to assist in generating higher quality perturbations and an adversarial loss to measure the ability of the generator to produce high-quality perturbations [9]. As we describe later, we develop our DDAS-EarlyExit on top of this extension.

[1]The cross entropy is calculated as: $entropy(X) = -\sum_i p(x_i) * log(p(x_i))$; where X represents the normalized probability vector of size i.

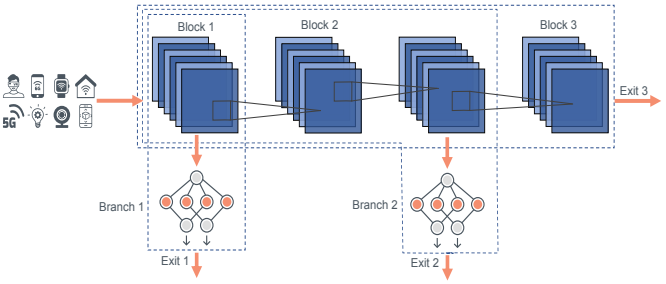
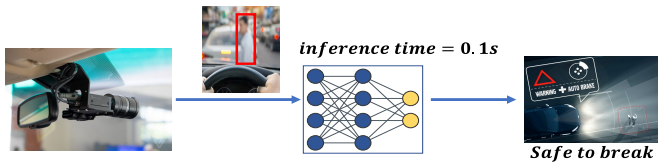


Fig. 1: iBranchy model architecture showcasing a model augmented with 2 extra branches

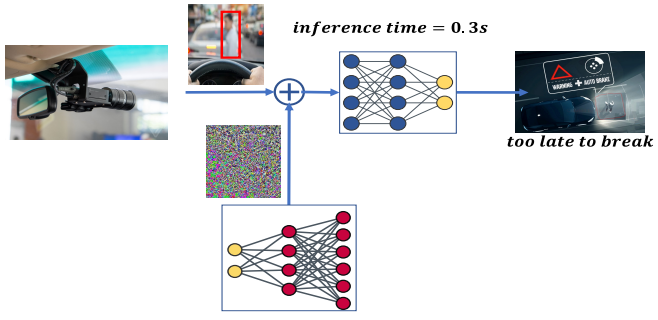
III. DDAS Attack Threat Model

The main objective of DDAS attacks is to compromise the dynamic execution of the D^{2}NN models during the inference process by causing the activation of the longest execution path which in turn causes the exhaustion of the highest computation resources. Consequently, DDAS attacks result in increasing the average inference time and power consumption of the target D^{2}NN model due to the extra computation needed by the model to reach a classification decision.

DDAS attack can be a simple malware installed by the attacker capable of intercepting the data feed sources such as cameras that are used by the target classification model. The malware will monitor and collect the inputs and the corresponding outputs of the model. This data will be used to



(a) Normal scenario



(b) Adversarial scenario

Fig. 2: DDAS attack targeting obstacle avoidance system in an autonomous vehicle slowing pedestrian detection below the threshold needed for safely braking the car

train the DDAS attack to generate adversarial noise specifically targeting the model that will be augmented to the data feed in order to decrease its responsiveness. Autonomous vehicles are an example target for this type of attacks, as they rely on fast input processing to execute critical operations such as obstacle avoidance and traffic mapping. A decreased responsiveness to these functionalities can cause traffic accidents as shown in Figure 2 that showcases a DDAS attack targeting an emergency braking system in an autonomous vehicle.

The impact of DDAS attacks on the classification accuracy needs to be taken into consideration, as it might match with the goals of the attack if a secondary goal of the attacker is to decrease the classification accuracy. However, this might increase the chance of detecting the attack due to performance degradation. Therefore, it might be desirable to design the attack to target specific performance metrics while minimizing the impact on classification accuracy. In our view, designing DDAS to accommodate variable impact on classification accuracy is crucial to cover the requirements of many attack scenarios.

In this study, we focus on attacking multiple early exit dynamic models. For these models, the goal of DDAS attacks is to target the ability of the models to perform early inference and terminate at earlier layers by forcing them to bypass most of the exit points till it reaches the natural endpoint of the model. We refer to attacks that target this specific early exit capability as DDAS-EarlyExit attacks.

IV. DDAS-EarlyExit Design and Implementation

At the most basic level, all DDAS-EarlyExit attacks function by compromising the ability of multi-exit dynamic models to perform early inference thus increasing the computation needed to classify input samples. Depending on different attack scenarios, the attacker may have to balance between aggressively attacking the early exit capability and decreasing classification accuracy, thus they need a method to tune the aggressiveness of the attack. Furthermore, the DDAS-EarlyExit attack should generate adversarial samples that maintain the exterior similarity to the normal samples.

We designed the DDAS-EarlyExit attack model with the following objectives 1) targeting the early exit ability of dynamic models, 2) providing a method to control the impact on classification accuracy, and 3)generating samples similar to the original input. Given these objectives, we chose to implement our attack model as an adversarial learning process using GANs [10], given their ability to generate high-quality synthetic data and preserve the original data's properties.

A. DDAS-EarlyExit Attack Design

Fig. 3 shows the design of our attack model. We utilized a GAN architecture consisting of a generator/discriminator scheme similar to [9] but with a different type of feed-forward network as a target - an iBranchy model. The training process starts by using the generator to generate adversarial noise mimicking the distributions of the real sample while using the discriminator to decide whether the sample belongs to the original distribution or not. The training process is guided through the loss function which is crucial in training the generator and the discriminator.

The choice of the loss function can be different based on the problem. For example, the popular loss function $C\&W$ [11] is chosen for a non-targeted adversarial attack that maximizes the probability of any other class label prediction. Another popular attack class is the targeted adversarial attack in which the loss function is configured to minimize the distance between the predicted class and the targeted class [9]. Unfortunately, both of these functions, and the current existing loss function designs in general, focus on decreasing accuracy which does not match the objectives of our attack. Therefore we opted to perform to design our custom function.

To design our loss function, we need to target a different parameter other than classification confidence, a parameter that reflects how iBranchy performs early exiting. Since iBranchy utilizes cross-entropy to perform the check on whether to perform early exiting on a specific branch. Our loss function will utilize it and work towards maximizing its value. iBranchy performs entropy calculations at each exit individually, producing multiple values for entropy. We can utilize this to target our attack towards a specific branch or a combination of branches.

The attack model is trained to minimize the overall neural network loss L, which is the combined loss of the generator G and discriminator D L_{GAN}, the adversarial loss L_{adv} representing the target network, and the perturbation loss $Lpert$ shown in Equation 1. Each of these terms is multiplied by a constant, and the set of constants α, β, γ are used to control the importance of each term during the learning process.

$$L = \alpha L_{adv} + \beta L_{GAN} + \gamma L_{pert} \tag{1}$$

Starting with the last term; the perturbation loss L_{pert} is calculated as shown in Equation 2. The goal of this term is to bound the magnitude of the perturbation $G(x)$, by adding a soft hinge loss in case the euclidean norm of the perturbations exceeds a certain threshold c. Hence, the objective of this loss is to train the model to choose smaller perturbation values that help maintain the original structure of the input sample. The $\mathbb{E}_x$ operator in the following equation calculates the expected value or mean over variable x.

$$L_{pert} = \mathbb{E}_x max(0, \|G(x)\| - c) \tag{2}$$

The middle term L_{GAN}, which is calculated using Equation 3, represents the minimax game between the discriminator D and the generator G, as D attempts to differentiate the perturbed data $x + G(x)$ from the original data while G attempts to generate perturbations $G(x)$ that can fool D.

$$L_{GAN} = \mathbb{E}_x(logD(x) + log(1 - D(x + G(x)))) \tag{3}$$

The first term, the adversarial loss L_{adv}, shown in Equation 4, represents the model we are trying to attack where $\Im(.)$ is the corresponding normalized probability vector of the input sample at this exit. This loss term is negative in order to train

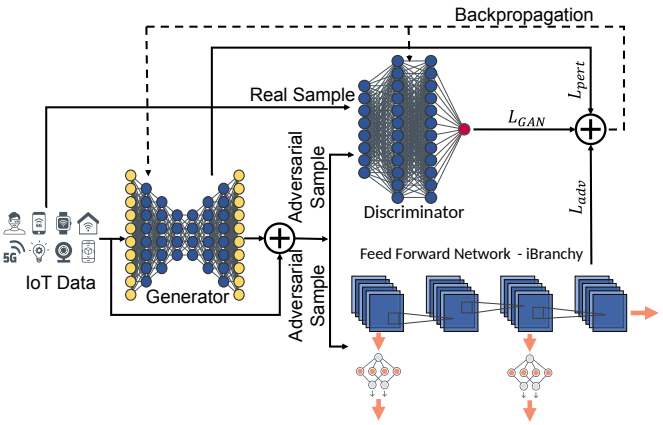


Fig. 3: DDAS-EarlyExit-GAN Attack Model

the attack model to produce perturbed images that maximize the entropy values of the input sample, which causes the perturbed image to have a higher likelihood of failing the entropy check and pushes samples to latter exits in iBranchy fast inference check.

$$L_{adv} = -\mathbb{E}_x entropy(\Im(x + G(x))) \tag{4}$$

This term can be used to attack a singular branch of iBranchy, since entropy is calculated at each individually, we can expand L_{adv} to target a specific branch or multiple branches simultaneously. We accomplish this by multiplying the entropy vector corresponding to all branches by the weight vector w as shown in Equation 5. This weight vector allows us to fine-tune the attack by selecting different weights for branches.

$$L_{adv} = -\mathbb{E}_x \sum_{i=1}^{N} w_i * entropy(\Im_i(x + G(x))) \tag{5}$$

where N is the number of exit points. This equation provides us with the method to adjust the attack aggressiveness and impact on classification accuracy, as by adjusting the weight vector w we can choose to focus on maximizing entropy over all exits producing a highly aggressive attack or we can select the weights to have a more forgiving attack with smaller impact but a higher chance of evading detection, by selecting lower values for certain branches.

B. DDAS-EarlyExit Attack Implementation

As shown in Fig. 3, the DDAS attack consists of three components: The generator, the discriminator, and the feed-forward network. The generator uses an autoencoder architecture with 3 conv layers encoding followed by 4 Resnet blocks and 3 conv layers decoding. The discriminator is a sequential model consisting of 3 conv layers. The feed-forward network is an iBranchy model based on a ResNet32 classifier trained with the CIFAR-10 dataset. Two extra exit points were added to it; one after the first residual block and another after the third block. The design of the exit branches is fairly simplistic with a pooling layer followed by a flattening fully connected layer and then a softmax layer to produce output probabilities to add minimum overhead at run time to simulate edge deployments.

The generator is trained to perturb the clean images which are then tested by the discriminator to differentiate between perturbed and clean images, the entropy of the samples is calculated across all exits of iBranchy and assigned weights according to Equation 5. The combined loss is then calculated and used to train the network using Backpropagation. Calculating the entropies across all exits requires access to the normalized output probability vector at all different exit points. However, access to the probability vector is only needed during the training of the attack model, and after the attack model has been trained, only an input sample is needed to generate an opposite adversarial sample. Therefore, we consider this attack to be a semi-whitebox attack [9]. We briefly discuss our plans to extend it to a blackbox attack in Section VIII.

V. Mitigating DDAS-EarlyExit Attack

The approach we have taken for mitigating the DDAS-EarlyExit attack is an adversarial retraining approach similar to [12]. Adversarial retraining generally works by generating adversarial examples and then mixing them with clean examples and using both adversarial and clean data to retrain the target model. This approach has shown to be relatively effective for traditional misclassification adversarial attacks as [13] shows the error rate on such images was seen to decrease drastically after retraining the network. However, our attack has different goals than misclassification and is measured through different metrics. Therefore, we aimed to implement an iterative robustness retraining scheme and evaluate its effectiveness on our DDAS-EarlyExit. We outline the iterative robustness process we implemented in Algorithm 1.

Algorithm 1: Implementation of robustness training

```
1  for i = 1..robustness_iter do
2      for j = 1..num_of_data_batches do
3          Obtain training samples from M_i training set
4          Use generator to generate adversarial noise
5          Combine adversarial noise to clean sample
6          Pass the adversarial sample by discriminator
7          Calculate loss and Update attack model A_i
8      Calculate accuracy and sample distribution of M_i
9      Calculate robustness of M_i against A_i
10     if M_i passes robustness threshold T then
11         break
12     Use A_i to generate adversarial training samples S'_i from
         clean samples S_i
13     Use mix of S'_i and S_i to retrain M_i to obtain M_{i+1}
```

It starts by training a DDAS-EarlyExit to target a model M by iteratively providing it with batches of training samples (line 3) and training the generator to generate the appropriate adversarial noise (line 4). This noise is added to the clean image (line 5), becoming the adversarial image. This image passes by the discriminator to check whether it can be differentiated from clean samples (line 6). The combined loss is computed and used to update the attack model (line 7). This operation continues until the attack model A is trained. Afterward, we calculate the accuracy and the distribution of

the sample over exit points of M_i under the attack A_i (line 8), which is used to calculate the robustness of the model (line 9). If the model shows adequate robustness we stop the iterative robustness process, otherwise, we continue the process of robustness training (lines 10&11). Finally, we use the attack model to generate adversarial training samples (line 12) and shuffle them with clean samples from the original dataset and use the combined dataset to obtain an updated model M_{i+1} (line 13). This process is repeated for a predetermined number of iterations or until the robustness goals have been met.

VI. Performance Evaluation

We evaluated the performance of our attack using the CIFAR-10 dataset to train our iBranchy model and the DDAS attack model. The dataset was partitioned and both the classification and the attack models were trained using disjointed parts of the overall data. The experiments were run on a local machine using AMD Ryzen 3800X CPU, Nvidia RTX 2080TI GPU, and 64GB RAM. In the following subsections, we discuss the experiment design and our evaluation process.

A. Experiments Design

In our experiments, we manipulated the L_{adv} term defined in Equation 5 by adjusting the vector w that is being multiplied by the entropy calculated at each of the exit points. We designed several scenarios using values of w to test the aggressiveness of the model. In addition, we designed two scenarios to evaluate whether we could adjust w to minimize drops in classification accuracy. All scenarios and their associated weight vectors are listed in Table I. These values were simply chosen to highlight disparate scenarios, however, more sophisticated parameter selection methods such as reinforcement learning can be used to create more specialized attacks. $Base$ scenario refers to the base iBranchy model before any attacks. $C\&W$ is the scenario representing an ordinary adversarial attack that targets classification accuracy rather than early exit capability, using $C\&W$ loss formula [11], this also serves as another base case. L_1, L_2, and L_3 scenarios focus on targeting a specific exit (Exits 1, 2, & 3). L_4 is a scenario that aims to attack Exits 1&2. L_5 aims at attacking all exits. L_6 scenario aims at attacking Exits 1&2, but includes a regularizing term to penalize high entropy at Exit 3 to assist the model in making more accurate predictions. This term utilizes a min-max aspect to the overall loss function by requiring the network to maximize entropy at Exits 1&2 while avoiding high values at Exit 3. Our intuition is that this will allow the network to be more conservative in pushing samples from Exits 1&2 in order to avoid a high penalty at Exit 3 and, consequently, will help to maintain overall accuracy. L_7 is the same as L_6 but with double the regularizing penalty that is expected to make the network more conservative and, as a result, improve overall accuracy.

	(w_0,w_1,w_2)	**Description**
Base	-	Original iBranchy base model
C&W	-	Traditional adversarial attack using $C\&W$
L_1	(1, 0, 0)	L_{adv} focused to target Exit 1
L_2	(0, 1, 0)	L_{adv} focused to target Exit 2
L_3	(0, 0, 1)	L_{adv} focused to target Exit 3
L_4	(1, 1, 0)	L_{adv} focused to target Exits 1&2
L_5	(1, 1, 1)	L_{adv} focused to target all exits
L_6	(1, 1, -1)	L_{adv} focused to target Exits 1&2, ignoring Exit 3
L_7	(1, 1, -2)	Same as L_6 with more emphasis on ignoring Exit 3

TABLE I: Summary of the scenarios we experimented with, and the weight vector associated with each scenario.

Our attack contains several hyper-parameters mentioned in Section IV-A that are used to tweak the learning process. We set the values of the variables $\alpha, \beta, and \gamma$, mentioned in Equation 1, to 10, 2.5, and 1 respectively. The value of the parameter c used to constrain the perturbations in Equation 2 is set to 0.3. These values were chosen through experimenting with the training process to obtain optimal results. We evaluated the performance of DDAS-EarlyExit using three different metrics: overall accuracy of the model after the attack, average inference time of the model, and output exit distribution over the different branches.

B. Evaluation of DDAS-EarlyExit Attack

Starting with the standard adversarial attack using $C\&W$, this attack aims to maximize the misclassification loss. The results in Figure 4 show a high negative impact on accuracy, and figure 6 shows an increasing percentage of samples leaving from the first branch and decreasing from the second and last branches which caused a decrease in inference time by about 21% as seen in Figure 5. This shows that standard adversarial attacks are incapable of achieving our goals of attack as it decreases accuracy and inference time. We attribute this ineffectiveness to the fact that maximizing misclassification confidence does not contribute to entropy maximization and can sometimes even lead to entropy minimization.

Now we analyze the different entropy-based adversarial loss functions. Starting with L_1 scenario, we notice a decrease in the percentage of samples leaving the first branch from 59% to 21%, while the percentage of samples leaving the second and the last branches increased from 27% to 37% and from 13% to 40% respectively. This caused an increase in the inference time by 26%. The impact on the model's accuracy is also not as large as the $C\&W$ based attack, it only dropped down to 54%, compared to the $C\&W$ based attack's reduction to 29%. In the L_2 scenario, we notice a decrease in the exit percentage of the first and second branches from 59% to 47% and from 27% to 15%, while the last branch increased from 13% to 34%. Due to the fact L_2 attack affected fewer samples to exit from the first branch, inference time only increased by 16%. In the L_3 scenario, this attack seems less effective as maximizing the entropy of the samples at the last exit has low impact because that the samples will leave at this point anyway making this attack relatively weak. A common observation in L_2 and L_3 is the drop in Exit 1 despite these scenarios targeting different exits, we attribute this to the training method as samples are fed to further exit even if they terminate at earlier creating a correlative relation between the exits.

The L_4 scenario which combines L_1 and L_2 shows more success, with the distribution across branches changes from

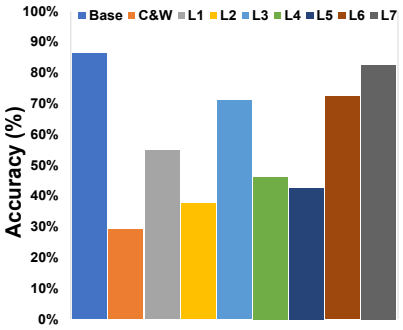


Fig. 4: Classification Accuracy across all experimental scenarios.

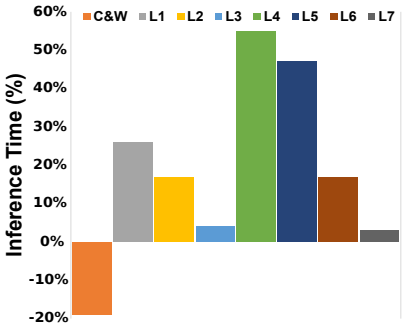


Fig. 5: Change in inference time compared to Base Scenario.

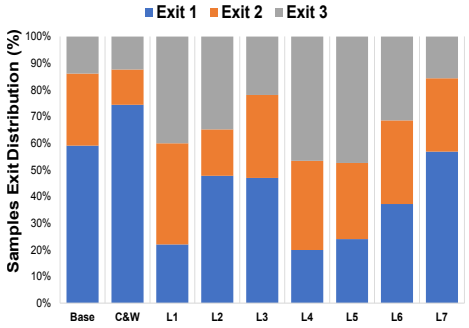
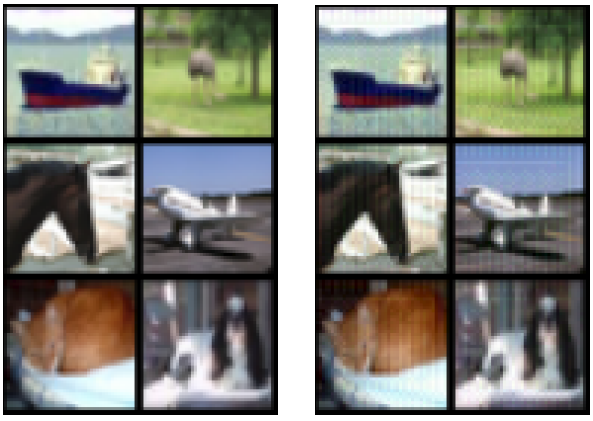


Fig. 6: The distribution of samples over all 3 exits different DDAS attacks



(a) Clean Images (b) With DDAS-EarlyExit

Fig. 7: An illustration of the impact of DDAS-EarlyExit attack using L_4 scenario on the original images.

59%, 27%, 14% to 19%, 34%, 47%, demonstrating a higher increase in the load at the later exits. This continues with L_5, which adds L_3 as well. This attack has slightly lower effect on the sample distribution than L_4 because maximizing the entropy at the last exit is not really beneficiary as the samples leave from that exit anyway. As a result, We conclude that combining the entropies of the early exists as in L_4 scenario can be the effective strategy to increase the attack potency.

Our last two attacks aim at balancing attack aggression and accuracy decline. The L_6 scenario encourages maximum entropy values at both the first and second branches while penalizing high entropy values at the third one. The accuracy drops to just 72% at the cost of a much less aggressive attack as only 32% exit from the last branch. The results validate our intuition that introducing the regularization term provides a way to tune the aggressiveness of the attack and its impact on accuracy. Finally, the L_7 results demonstrate that increasing the regularization term too much causes the attack to become ineffective since the network becomes more conservative and performs relatively similarly to the original base model, cancelling out most of the attack's effects.

Figure 7 depicts a subset of the images used in our experiments, and the same subset of images augmented with adversarial noises generated by our DDAS-EarlyExit attack when using $L4$ scenario. As demonstrated, our attack has no effect on the visual appearance of the images, implying that this attack could go undetected.

C. Evaluation of Iterative Robustness Training

We evaluate the performance of our mitigation approach assuming the most aggressive attack model L_4. Table II shows the performance of our iterative robustness training approach. The different rows of the table II show the performance metrics (accuracy and samples distribution) over the iterations of the robustness training where every two rows represent an iteration consisting of the model before and after the attack.

The results show that with additional retraining iterations, the robustness of the model improved against further attacks. However, after a number of iterations, we observed that the model's robustness converged and became nearly constant. We attribute this to the fact that there is a limit to the amount of information that iterative robustness retraining can contribute to the model, and after a certain number of iterations, there is nothing to gain by performing further retraining of the model as well as the DDAS attack model. This demonstrates that adversarial samples of the DDAS attack are beneficial for retraining and increasing the robustness of the model, but the improvements cease after a certain number of repetitions.

Model	Accuracy	Exit #1	Exit #2	Exit #3
Base Model	88%	59%	27%	13%
Attack on Base	45%	19%	34%	47%
Retrained$_1$ Model	85%	56%	26%	18%
Attack on Retrained$_1$	62%	30%	26%	43%
Retrained$_2$ Model	84%	56%	28%	15%
Attack on Retrained$_2$	68%	35%	28%	37%
Retrained$_3$ Model	83%	54%	25%	21%
Attack on Retrained$_3$	72%	40%	27%	33%
Retrained$_4$ Model	83%	55%	26%	19%
Attack on Retrained$_4$	73%	41%	30%	29%

TABLE II: Performance of iterative robustness over 4 iterations

VII. Related Work

Adversarial attacks on edge-based deep neural networks are currently emerging and they are rapidly growing. In recent work [15], authors classify the different types of possible edge-based adversarial attacks based on the attacker objectives,

model access, attack targets, and defenses. The attacker's goals include disrupting functionality and inferring user, and model privacy. Moreover, a summary of multiple recent edge-based adversarial attack-based works has been described including API attacks [16], side-channel attacks [6], and probing-based attacks [17]. Unlike the existing adversarial attack works, we design, develop, and evaluate adversarial attacks that impact the computation, inference time, and power consumption of the D^{2}NN deployed hosting device/application in this work. Recently, authors in [18] show adversarial attacks can impact the computation of Reinforcement Learning (RL) based and Markov decision models. However, our work differs substantially from RL-based adversarial attacks in that we focus on attacking the different activations/components of the D^{2}NN and not on perturbing the policies used by an RL-trained agent.

Recent works have demonstrated that the design of popular conventional DNN models can be inferred using side-channel attacks by measuring the time [19] and power consumption [20]. We believe that these works give some insights for fingerprinting conventional DNN models, which can help in reverse-engineering the DNN network parameters. However, unlike our work, the authors do not perform any adversarial attacks on the accelerated/edge D^{2}NN models for impacting resource consumption or network latency. However, we note that their work can expand the possibilities of building different types of Black-Box attacks for D^{2}NN models where the confidence measures of the inference are unavailable. Another related work is [25], Hong et el. propose an adversarial attack on multi-exit models . However, unlike their work, we use a GAN-based technique for generating adversarial samples in our approach, given GANs have shown good performance with the generation of adversarial samples for traditional adversarial attacks.

VIII. Conclusion

In this paper, we developed DDAS-EarlyExit attack, a GAN-based attack against dynamic deep learning networks utilizing early exit models such as *iBranchy*. We showed that our attack is capable of significantly impacting the performance of D^{2}NN models in terms of power consumption and inference time for edge-based IoT applications/devices. We explored various attack scenarios that make use of DDAS-EarlyExit and we evaluated different entropy-based adversarial loss functions and highlighted how they differ from misclassification loss, and we showcased how they can be used to design a flexible attack. We also implemented and evaluated an incremental adversarial robustness retraining scheme for building resilient early-exit D^{2}NN models.

As for future works, we would like to extend our semi-white box attack to become a black box attack. While our DDAS-EarlyExit is capable of producing the appropriate adversarial noise after training without requiring any underlying information from the target model, it still needs access to the output probability vector of all exit branches from the target model for its training process. We believe this extension can be accomplished using techniques such as knowledge distillation [23] and surrogacy training [24] to produce a mirrored model to the one we are aiming to attack and target our attacks towards the generated model. Another area we would like to explore is the mitigation techniques against adversarial attacks as well detection mechanisms for these types of attacks.

References

[1] Hinton, G., Vinyals, O. & Dean, J. Distilling the knowledge in a neural network. *ArXiv Preprint ArXiv:1503.02531*. (2015)

[2] Liu, J., Tripathi, S., Kurup, U. & Shah, M. Pruning algorithms to accelerate convolutional neural networks for edge applications: A survey. *ArXiv Preprint ArXiv:2005.04275*. (2020)

[3] Han, Y., Huang, G., Song, S., Yang, L., Wang, H. & Wang, Y. Dynamic neural networks: A survey. *ArXiv Preprint ArXiv:2102.04906*. (2021)

[4] Teerapittayanon, S., McDanel, B. & Kung, H. Branchynet: Fast inference via early exiting from deep neural networks. *2016 23rd International Conference On Pattern Recognition (ICPR)*. pp. 2464-2469 (2016)

[5] Vaudaux-Ruth, G., Chan-Hon-Tong, A. & Achard, C. ActionSpotter: Deep Reinforcement Learning Framework for Temporal Action Spotting in Videos. *2020 25th International Conference On Pattern Recognition*.

[6] Hua, W., Zhou, Y., De Sa, C., Zhang, Z. & Suh, G. Channel gating neural networks. *ArXiv Preprint ArXiv:1805.12549*. (2018)

[7] Pachecom, R. & Couto, R. Inference time optimization using branchynet partitioning. *2020 IEEE Symposium On Computers And Communications (ISCC)*. (2020,7), https://doi.org/10.1109

[8] Nukavarapu, S., Ayyat, M. & Nadeem, T. iBranchy: An Accelerated Edge Inference Platform for IoT Devices. *The Sixth ACM/IEEE Symposium On Edge Computing (SEC)*. pp. 392-396. (2021)

[9] Xiao, C., Li, B., Zhu, J., He, W., Liu, M. & Song, D. Generating Adversarial Examples with Adversarial Networks. (2019)

[10] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. & Bengio, Y. Generative adversarial networks. *Communications Of The ACM*. **63**, 139-144 (2020)

[11] Carlini, N. & Wagner, D. Towards evaluating the robustness of neural networks. *2017 Ieee Symposium On Security And Privacy (sp)*. pp. 39-57

[12] Msika, S., Quintero, A. & Khomh, F. SIGMA: Strengthening IDS with GAN and Metaheuristics Attacks. *ArXiv Preprint ArXiv:1912.09303*

[13] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. & Fergus, R. Intriguing properties of neural networks. (2014)

[14] Krizhevsky, A., Nair, V. & Hinton, G. CIFAR-10 (Canadian Institute for Advanced Research). (0), http://www.cs.toronto.edu/ kriz/cifar.html

[15] Isakov, M., Gadepally, V., Gettings, K. & Kinsy, M. Survey of attacks and defenses on edge-deployed neural networks. *2019 IEEE High Performance Extreme Computing Conference (HPEC)*. pp. 1-8 (2019)

[16] Tramèr, F., Zhang, F., Juels, A., Reiter, M. & Ristenpart, T. Stealing machine learning models via prediction apis. *25th USENIX Security Symposium (USENIX Security 16)*. pp. 601-618 (2016)

[17] Isakov, M., Bu, L., Cheng, H. & Kinsy, M. Preventing neural network model exfiltration in machine learning hardware accelerators. *2018 Asian Hardware Oriented Security And Trust Symposium (AsianHOST)*.

[18] Yekkehkhany, A., Feng, H. & Lavaei, J. Adversarial Attacks on Computation of the Modified Policy Iteration Method.

[19] Won, Y., Chatterjee, S., Jap, D., Bhasin, S. & Basu, A. Time to Leak: Cross-Device Timing Attack On Edge Deep Learning Accelerator. *2021 International Conference On Electronics, Information, And Communication (ICEIC)*. pp. 1-4 (2021)

[20] Zhao, M. & Suh, G. FPGA-based remote power side-channel attacks. *IEEE Symposium On Security And Privacy (SP)*. pp. 229-244 (2018)

[21] He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. *Proceedings Of The IEEE Conference On Computer Vision And Pattern Recognition*. pp. 770-778 (2016)

[22] Goodfellow, I., Shlens, J. & Szegedy, C. Explaining and Harnessing Adversarial Examples. (2015)

[23] Gou, J., Yu, B., Maybank, S. & Tao, D. Knowledge Distillation: A Survey. *International Journal Of Computer Vision*. **129**, 1789-1819 (2021,3), http://dx.doi.org/10.1007/s11263-021-01453-z

[24] Qin, Y., Xiong, Y., Yi, J. & Hsieh, C. Training Meta-Surrogate Model for Transferable Adversarial Attack. (2021)

[25] Hong, S., Kaya, Y., Modoranu, I. & Dumitraş, T. A Panda? No, It's a Sloth: Slowdown Attacks on Adaptive Multi-Exit Neural Network Inference. *ArXiv Preprint ArXiv:2010.02432*. (2020)