July 2020

Machine Learning for the Internet of Things: Applications, Implementation, and Security

Vishalini Laguduva Ramnath

University of South Florida

Follow this and additional works at: https://scholarcommons.usf.edu/etd

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Scholar Commons Citation


This Dissertation is brought to you for free and open access by the Graduate School at Scholar Commons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact scholarcommons@usf.edu.
Machine Learning for the Internet of Things: Applications, Implementation, and Security

by

Vishalini Laguduva Ramnath

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy
Department of Computer Science and Engineering College of Engineering University of South Florida

Major Professor: Srinivas Katkoori, Ph.D.
Sriram Chellapan, Ph.D.
Robert Karam, Ph.D.
Morris Chang, Ph.D.
Ramachandran Kandethody, Ph.D.

Date of Approval:
June 18, 2020

Keywords: Machine Learning, Internet of Medical Things, Hardware Security, Constrained Platforms

Copyright © 2020, Vishalini Laguduva Ramnath
Dedication

I would like to dedicate this dissertation to my husband, my parents and my grandparents. No words can describe how much I am grateful for your presence in my life. Thank you for your unconditional love and support.
Acknowledgments

There are many people to whom I owe my gratitude for their support during my years as a Ph.D. student.

This dissertation would not have been possible without the guidance and support of many people. I would like to start by expressing my gratitude to my graduate advisor, Dr. Srinivas Katkoori, who was always happy to discuss research ideas and provide guidance, constant encouragement, and precious professional and personal advice. His continuous support made graduate school an incredibly rewarding experience. I want to express my appreciation to Dr. Robert Karam for his advice, and insightful feedback. I also want to thank my committee members, Dr. Sriram Chellapan, Dr. Ramachandram Kandethody, Dr. Morris Chang, and the chairperson of my defense, Dr. Andrew M. Hoff.

I would like to thank several other faculty members from whom I have learned a lot and who have given me valuable academic and career advice. I would like to thank my friends for their significant support during my journey towards this degree. Last but not least, I want to thank the administrative staff of the CSE, including Gabriela Franco, Laura Owczarek, Jessica Pruitt for their help and the entire department for their warm and welcoming manner.
## Table of Contents

List of Tables ................................................................................ iii

List of Figures................................................................................ iv

Abstract ..................................................................................... vi

Chapter 1: Introduction ...................................................................... 1
  1.1 Challenges and Current Trends for the Internet of Things ................. 3
  1.2 Contribution and Outline ........................................................ 7
  1.3 Chapter Summary ............................................................... 9

Chapter 2: Related Work .................................................................... 10
  2.1 Continuous Monitoring through Ubiquitous Sensing ......................... 10
  2.2 Implementing Deep Neural Networks on Constrained Platforms .......... 12
  2.3 Physically Unclonable Functions ................................................. 13
    2.3.1 Strong PUF Architectures .............................................. 14
    2.3.2 PUF-based IoT Security ................................................ 18
    2.3.3 Encryption Protocols for IoT Node Authentication ................. 18
    2.3.4 Machine Learning-based Attacks on PUF Models ............... 19
  2.4 Chapter Summary ............................................................... 20

Chapter 3: Continuous Ubiquitous Activity Monitoring............................... 22
  3.1 Introduction ..................................................................... 22
  3.2 Unsupervised Sleep State Segmentation .......................................... 25
    3.2.1 Local Temporal Feature Representation Learning ..................... 26
    3.2.2 Capturing Long Range Temporal Dependencies ...................... 27
    3.2.3 Sleep State Segmentation ............................................... 28
    3.2.4 Network Architecture and Implementation Details .................... 30
    3.2.5 Implementation on Constrained Platforms .............................. 31
  3.3 Evaluation and Analysis ......................................................... 32
    3.3.1 Quantitative Results.................................................... 33
  3.4 Chapter Summary ............................................................... 36

Chapter 4: Implementation of CNNs on Constrained Platform....................... 37
  4.1 Introduction ..................................................................... 37
  4.2 Convolutional Neural Networks (CNNs) .......................................... 40
    4.2.1 Convolutional Layers ................................................... 41
    4.2.2 Subsampling Layers .................................................... 42
    4.2.3 Network Architectures .................................................. 43
    4.2.4 Ex-situ Training ........................................................ 44
  4.3 CNN Optimizations and Design Trade-Offs ..................................... 45
    4.3.1 Layer-Specific Optimizations............................................ 46
## List of Tables

| Table 2.1 | Comparison of existing ML attacks (best results through simulation) and modeling resistant technique on strong PUFs | 19 |
| Table 3.1 | Performance of different approaches on sleep vs. wake segmentation task | 33 |
| Table 3.2 | Performance of different approaches on wake vs. NREM vs. REM segmentation task | 35 |
| Table 3.3 | Performance of the model on different platforms | 36 |
| Table 4.1 | Impact of receptive field size on latency, energy consumption, and memory requirements for a $28 \times 28$ input image | 46 |
| Table 4.2 | Impact of dense layer capacity on latency, power consumption, and memory | 50 |
| Table 4.3 | Performance comparison of different architectures with design level optimizations on the MNIST dataset | 57 |
| Table 4.4 | Performance comparison of different architectures with design level optimizations on the CIFAR-10 dataset | 57 |
| Table 4.5 | Performance comparison of LeNet on different platforms | 60 |
| Table 5.1 | Brute force attack classification performance and cloning accuracy | 71 |
| Table 5.2 | Discriminator performance for different PUF architectures | 84 |
| Table 5.3 | Cloning error and time for different PUF models | 88 |
| Table 5.4 | Cloning rates and times for different PUF architectures under different encryption protocols | 89 |
List of Figures

| Figure 1.1 | A typical IoMT framework is collection of ubiquitous sensor nodes and data processing servers that collect and process highly sensitive data. | 2 |
| Figure 2.1 | The Taxonomy of PUF. | 15 |
| Figure 2.2 | Architecture of an Arbiter PUF (Reproduced from [32]). | 16 |
| Figure 2.3 | Architecture of a 2-XOR Arbiter PUF (Reproduced from [38]). | 17 |
| Figure 3.1 | An IoMT framework for sleep continuous monitoring. | 23 |
| Figure 3.2 | The proposed sleep segmentation framework to reduce the noise in the segmentation output and enforce coherence. | 27 |
| Figure 3.3 | The proposed autoencoder network architecture to capture the local temporal patterns in the input data. | 30 |
| Figure 3.4 | Accuracy comparisons across individual patients for both tasks, sorted by accuracy. | 34 |
| Figure 4.1 | A typical CNN architecture for the object recognition task. | 40 |
| Figure 4.2 | Convolutional operations on an input $10 \times 10$ image given a kernel of size $3 \times 3$ with (a) valid padding or (b) same padding. | 48 |
| Figure 4.3 | CNN architecture on the PYNQ development board. | 53 |
| Figure 4.4 | The physical setup for conducting the experiments. | 54 |
| Figure 4.5 | Comparison of power consumption for four optimized (red) and unoptimized (blue) CNN architectures on MNIST. | 58 |
| Figure 4.6 | Comparison of power consumption for four optimized (red) and unoptimized (blue) CNN architectures on CIFAR-10. | 59 |
| Figure 5.1 | A typical IoT architecture with both the enrollment phase and the authentication phase of a PUF-based IoT node authentication scheme. | 64 |
Figure 5.2  The proposed attack model on oblivious PUFs architecture. ................. 70

Figure 5.3  Illustration of (a) an autoencoder, (b) a variational autoencoder, and (c) the proposed multi-headed variational autoencoder. ................................. 76

Figure 5.4  ML-based discriminator model to ascertain a PUF integrity. ................. 84

Figure 5.5  Effect of challenge obfuscation on cloning performance. ...................... 90

Figure 5.6  Comparison of cloning and discriminator accuracy for cloning models under different PUF architectures. ......................................................... 92
Abstract

Artificial intelligence and ubiquitous sensor systems have seen tremendous advances in recent times, resulting in groundbreaking impact across domains such as healthcare, entertainment, and transportation through a collective ecosystem called the Internet of Things. The advent of 5G and improved wireless networks will further accelerate the research and development of tools in deep learning, sensor systems, and computing platforms by providing improved network latency and bandwidth. While tremendous progress has been made in the Internet of Things, current work has largely focused on building robust applications that leverage the data collected through ubiquitous sensor nodes to provide actionable rules and patterns. Such frameworks do not inherently take into account the issues that come with scale such as privacy, security of the data, and the ability to provide a completely immersive experience. This is particularly significant since, due to the somewhat limited scope of computing resources, the IoT edge nodes themselves do not process the observed data. Instead, they transmit the collected data to more powerful servers for processing. This information transmission can place strain on the network while introducing security concerns such as eavesdropping and man-in-the-middle attacks.

In this dissertation, we address these concerns by using machine learning as a disruptive tool by developing privacy-aware, lightweight algorithms while evaluating the feasibility of hardware security primitives such as physical unclonable functions (PUFs) for IoT node security. To be specific, we develop unsupervised algorithms for continuous activity monitoring from ubiquitous
sensors without *any labeled data*, which forms the first step to a decentralized learning paradigm. The proposed framework is inherently privacy preserving by limiting the amount of data transmitted through the network while providing real-time feedback. Second, we analyze the properties of different deep learning approaches with respect to power consumption, memory footprint, and latency and provide design-time optimizations to enable implementation on compute-constrained platforms. Finally, we evaluate the feasibility of using PUF-based authentication for IoT edge nodes by exploring their susceptibility to machine learning attacks. We show that strong PUF architectures are susceptible to a non-invasive machine learning-based cloning attack. We also propose a probabilistic, discriminator model to bolster the security of the PUF-based authentication protocol by identifying possible instances of cloning attacks and bolstering the PUF-based authentication.

Combined, these approaches offer a way forward for the development of an IoT framework for continuous activity monitoring that can scale to millions of nodes while ensuring the privacy and security of the observed data. We show that the proposed activity monitoring algorithm can effectively recognize and segment activities from streaming data without any labeled data on constrained platforms with close-to-real-time latency. Our design-time improvements for deep learning algorithms can yield up to 11× reduction in power consumption compared to another FPGA platform with 96× more memory capacity, while maintaining the state-of-the-art classification accuracy. Through extensive experiments, we show that strong PUF architectures can be successfully cloned, including those encrypted using two different encryption protocols in DES and AES and with varying degrees of obfuscation. The proposed discriminator can distinguish cloned PUF devices and authentic PUFs with an average accuracy of 96.01% and can be used for rapidly authenticating millions of IoT nodes remotely from the cloud server.
Chapter 1: Introduction

The Internet of Things (IoT) ecosystem has grown exponentially with the convergence of various technologies such as deep learning, sensor systems, and advances in computing platforms such as Field Programmable Gate Arrays (FPGAs) and Graphical Processing Units (GPUs). The advent of 5G technology and the promise of higher bandwidth is expected to increase the highly connected nature of today’s IoT ecosystem. These advances have led to the development of ubiquitous sensor nodes to enable the collection of physiological data for smart healthcare. Collectively termed the Internet of Medical Things (IoMT), smart healthcare has emerged as a viable option to improve quality of life in different ways such as healthy living, remote access to healthcare for rural areas, and home-based healthcare to name a few.

A typical IoMT framework is illustrated in Figure 1.1. It can be seen that the data collected across the sensor nodes are transmitted through a myriad of servers, routers and the network gateway before it is processed offline and the actionable rules and applications can be obtained. Applications of IoT devices range from wearable computing devices, bio-implantable devices to monitor vital bodily functions for direct human interaction, as well as for “smart” devices that we interact with on a day-to-day basis. With such a highly pervasive nature of “smart” devices, the nature of data being collected and processed can be increasingly private and require safeguards to ensure the integrity and security of the data [1, 2].
Due to the somewhat limited scope of computing resources, the IoT edge nodes themselves do not process such information. Instead, they are used as data collection agents that transmit the collected data to more powerful edge servers for information processing. This information transmission is often done through wireless networks. This introduces some security concerns such as eavesdropping and man-in-the-middle attacks, which are quite common security concerns and hence require robust security protocols for ensuring the integrity of the transmitted data. While we have explored the infrastructure aspects of the Internet of Medical Things ecosystem, there have been relatively fewer works on actual smart algorithms that can leverage the wireless edge and distributed learning platform for intelligent health diagnostics and monitoring.

Figure 1.1: A typical IoMT framework is collection of ubiquitous sensor nodes and data processing servers that collect and process highly sensitive data.

Current work in the IoMT framework has largely focused on building robust applications that leverage the data collected through ubiquitous sensor nodes to provide actionable rules and patterns. They make the following underlying assumptions: (1) the data nodes are secure and
are not susceptible to any intrusions, (2) the focus is on obtaining higher accuracy at the cost of higher computational requirements, and (3) the data nodes are not computationally constrained. These considerations have resulted in highly specialized, resource-intensive applications that are not cognizant of the communication, computation, and architectural considerations of the IoMT framework. Ideally, applications must trade off the accuracy gains based on the context and resource availability for more efficient utilization of the various data and compute nodes to better adapt workload distribution in edge environments.

1.1 Challenges and Current Trends for the Internet of Things

One of the major limitations in current machine learning applications in the IoMT framework is that they are not resource-aware and are not inherently privacy preserving. The data is collected at the edge nodes, transmitted through the network (wired or wireless) and processed offline on large, computationally intensive servers. One significant step towards ensuring data security and privacy is to reduce the amount of data transmitted via the wireless spectrum. This approach requires that the data must be processed at the edge nodes themselves while ensuring that only the novel data is transmitted to the computing server. Most successful applications are based on deep learning approaches, which require large amounts of labeled data and highly computationally intensive platforms. However, the compute-constrained nature of devices at the edge limits the extent to which the data can be processed effectively, especially given the computational complexity of modern machine learning approaches.

The critical aspect of running deep learning architectures on edge devices is to go beyond the development of hardware accelerators. It requires a deeper understanding of the various design
aspects of neural networks and their impact on SWAP (Size, Weight, Area, and Power). Current approaches to enabling deep learning on resource-constrained hardware platforms have focused primarily on reducing latency and power. They make strong assumptions about the availability of resources such as processing power, memory, and sizes. This provides a set of challenges: (1) deep learning models require significant training resources such as compute power and labeled training data, (2) enabling inference and training of deep learning models on constrained platforms requires a trade-off between accuracy and SWAP, which might not be an option for mission-critical applications such as real-time healthcare analytics, and (3) the data distribution of the collected/observed data can be dynamic and may not conform to the training data distribution and hence can harm the performance of deep learning models. Neural networks are typically implemented on relatively unconstrained devices, such as a workstation Graphics Processing Units (GPUs), so the design and optimization techniques do not scale well to low power and resource-constrained applications in the Internet of Things (IoT). While efforts have shifted for various acceleration frameworks on constrained platforms, obtaining the energy-efficient implementations without a significant drop in accuracy remains an art than science. The use of alternate computing platforms and efficient training methodologies can help alleviate this dependence on computation resources.

Current machine learning applications for IoMT have largely been tackled through supervised learning [3, 4]. These approaches, while working well, require large amounts of expert-annotated, labeled data which can be very expensive to obtain. Additionally, the collection of such an amount of data involves the capture and transmission of private, sensitive information such as motion characteristics, location, heart rate, etc., and can cause serious security issues if there is any breach in security. With advances in machine learning and hardware security, the integrity of data
can be compromised and the continuous collection of highly sensitive information can be liable to issues. Decentralized learning [5] has emerged as a viable alternative, where each model on the edge devices is sent to a remote server. The remote server then finds a common model through various model averaging methods and the on-device model on the edge nodes are simultaneously updated with the new, common model. This approach requires fast and efficient learning algorithms that can work at scale and on-device, including on computationally constrained edge nodes. However, this does pose some security concerns: (1) it has been shown that knowing the model parameters and weights can enable white-box attacks, (2) a man-in-the-middle attack can perturb the weights in a subtle manner that can completely change the model’s performance characteristics, and (3) combining models across various compute architectures can result in muddled precision characteristics for compressed models, which can, in turn, degrade their performance. Additionally, the annotation of such large-scale data can be difficult to obtain and is not conducive for scaling to millions of users. The other alternative is the use of self-supervised and unsupervised approaches to learn robust representations from streaming data without the need to store and transmit the data. With the robust representations, the model can then be fine-tuned with smaller, annotated data. Combining the representation learning capabilities of self-supervised networks and the model aggregation capabilities of decentralized learning offer a way forward for processing large amounts of data at the edge, while preserving the privacy of the person using the IoMT devices and ensuring the integrity of the machine learning framework.

Ensuring the integrity and security of the collected data [1, 2] requires robust security protocols for ensuring the integrity of the transmitted data. Security protocols, such as node authentication, have to be sufficiently lightweight, yet highly secure to ensure that these protocols can
be performed on power-constrained IoT nodes. Authentication protocols can vary from being very simple, such as physical storage of a secret key on silicon devices, to complex cryptography-based algorithms that can require significant power and area requirements on the device. It has, however, been shown that the most straightforward authentication that of physically storing the secret key on the node device can be bypassed through physical and side-channel attacks [6]. Recovering the secret key through such physical attacks can compromise the entire IoT network and hence compromise the integrity and anonymity of the transmitted data. With the need for lightweight, yet secure authentication protocols increasing with the rapidly growing use of IoT nodes, physically unclonable functions (PUFs) [7] have emerged as a viable option for IoT node security [8].

While highly sophisticated and secure, PUF models are susceptible to cloning using complex mathematical models and cryptanalysis. Previous works have shown that they can be cloned using machine learning models. However, they require that the IoMT node is physically available, and inside knowledge about the node is known a priori. Another assumption in current works is that the protocols for PUF-based authentication are sent through the communication channel in plain text, i.e., no encryption masks the direct relationship between challenge and response characteristics of the PUF within the data node. Given that most, if not all, communication in the wireless channel is encrypted through some hashing or encryption technique, this is a very strong assumption to make. Hence, there is a strong need to evaluate the viability of using PUF-based authentication protocols for node security in the IoMT framework and formulating a suitable defense mechanism for detecting and authenticating compromised edge nodes.
1.2 Contribution and Outline

In this dissertation, we introduce an inherently privacy preserving IoMT framework for continuous human activity monitoring while taking into account factors such as resource constraints on edge devices. We provide a mechanism to implement the proposed algorithm on constrained platforms using design-level optimizations. We also evaluate the use of physically unclonable functions in an authentication protocol for ensuring the security of the edge nodes.

In Chapter 2, we review extant literature on the use of machine learning in the Internet of Medical Things (IoMT). We explore the various approaches to continuous activity monitoring using multimodal data such as accelerometer (motion) and photoplethysmography (heart rate) data from consumer-grade devices. We then review existing approaches for implementing complicated, deep learning approaches on edge devices which are computationally constrained. Next, we explore the use of PUFs as an authentication protocol in the IoMT framework and review their susceptibility to machine learning-based cloning attacks. Next, we propose a machine-learning based countermeasure to identify compromised nodes while enhancing the authentication of edge nodes from a central server without the need for physical review.

In Chapter 3, we introduce an inherently privacy preserving activity monitoring algorithm from streaming data through self-supervised representation learning. Using a probabilistic segmentation algorithm, we segment the continuous stream of motion and photoplethysmography heart rate data into its constituent activity states, without any labeled data. The continual learning approach can be used in decentralized learning frameworks to enhance the security and privacy of data. We show that predictive learning-based self-supervised representation learning can be used to learn robust features for unsupervised sleep state segmentation. We also show that the proposed
framework can be implemented on constrained platforms and offer a way forward for real-time, unsupervised sleep state segmentation on constrained platforms without trading off accuracy for latency by leveraging current advances in hardware accelerators for deep learning.

In Chapter 4, we analyze the properties of different deep learning approaches with respect to power consumption, memory footprint, and latency and describe optimized implementations of CNNs on a low power heterogeneous computing platform. We show significant improvement in power consumption with minimal effect on latency while maintaining the state-of-the-art classification accuracy. Our design-time optimizations enable the implementation of large deep learning architectures with limited available memory and allow us to use significantly less memory than previously reported implementations while increasing throughput on smaller platforms with a strict area, power, and memory constraints. We show that modifying certain network design parameters such as filter size, number of fully connected layers, and subsampling techniques have a considerable impact on the overall performance and efficiency, enabling informed trade-offs and optimization. We demonstrate up to $11\times$ reduction in power consumption compared to another FPGA platform with $96\times$ more memory capacity, while maintaining the state-of-the-art classification accuracy.

In Chapter 5, we evaluate the feasibility of using PUF-based authentication for IoMT edge nodes by exploring its susceptibility to machine learning cloning attacks. We show that a brute force cloning attack on strong PUFs is increasingly complex and hence not trivial for feasible cloning. We propose a non-invasive, architecture-independent cloning attack on strong PUFs and show that the proposed approach can successfully clone the PUF model even if the challenge-response pair is encrypted or obfuscated. We also propose a probabilistic, discriminator model to bolster the security of the CRP protocol by identifying possible instances of cloning attacks. Through extensive
experiments, we show that the proposed framework can successfully clone different PUF architectures, including those encrypted using two different encryption protocols in DES and AES and with varying degrees of obfuscation. The discriminator can distinguish cloned PUF devices and authentic PUFs with an average accuracy of 96.01% and can be used for rapidly authenticating millions of IoT nodes remotely from the cloud server.

Finally, in Chapter 6 we identify key challenges for the Internet of Medical Things and discuss the way to push the frontiers of IoMT frameworks using machine learning as a disruptive technique and framework to manage the growing complexity and scale of IoMT frameworks.

1.3 Chapter Summary

In this chapter, we introduced the basic structure of an IoT framework and discussed the predominant approaches taken in designing current IoT systems. We identified privacy, security, and the need for resource-aware algorithms as key areas where machine learning could be used to enhance scalability of the IoT framework and outlined our contributions in these areas.
Chapter 2: Related Work

In this chapter, we discuss prior work in designing activity monitoring systems using ubiquitous sensors. We also review current efforts in the design and implementation of deep neural networks on constrained platforms. We conclude with a review of the basics of physically unclonable functions and their use in the Internet of Things as an authentication mechanism.

2.1 Continuous Monitoring through Ubiquitous Sensing

The problem of sleep state recognition and segmentation has been largely tackled through supervised learning \[3, 4\]. These approaches, while working well, require large amounts of expert-annotated, labeled data which can be very expensive to obtain. Additionally, the collection of such an amount of data involves the capture and transmission of private, sensitive information such as motion characteristics, location, heart rate, etc., and can cause serious security issues if there is any breach in security. With advances in machine learning and hardware security, the integrity of data can be compromised and the continuous collection of highly sensitive information can be liable to issues.

Doherty and Smeaton \[9\] proposed an algorithm for segmenting lifelog data into events. Lifelogging is the process of using the Microsoft SenseCam architecture to collect a sequence of images every second along with other sensory data such as accelerometer readings, thermometer readings, infrared sensor readings, and light sensors. This allowed for multimodal feature extraction
from the experience of the user collecting the data. Doherty and Smeaton proposed a method for segmenting the lifelog data into events based on the MPEG-7 encoded visual content. Each image from the SenseCam data is processed to extract a feature set consisting of color layout, color structure, scalable color, and edge histogram. Using these MPEG-7 descriptors and other visual features, they segment the data into events by performing three steps: (1) compute similarity between adjacent blocks of images, (2) determine threshold value by learning from the values of similarity which correspond to different events, and (3) remove successive event boundaries to form a cohesive event. The similarity measure is computed by using the MinMax normalization method on the feature vector consisting on the extracted visual features as well as other sensor data. An event boundary likelihood is calculated by comparing the similarity measure with the adjacent images using a peak scoring technique. Different distance measuring techniques such as Euclidean, Manahattan, and Histogram intersection were tested out with the Histogram Intersection providing the best results.

Hsu et al. [10] were among the first to use consumer device-based data for developing a continuous monitoring system from multiple sensors such as accelerometer, gyroscope, gravity sensor, compass data as well as light sensor data. First, the user’s motion data is analyzed and classified into six (6) categories based on a motion feature vector computed using mean, variance, skew, kurtosis, and FFT (Fast Fourier Transform). The context of the user’s action is classified using a host of SVM classifiers. Based on the user’s context, the importance of a temporal window is computed and the video is used to record the plausible event for further event detection. This usage of sensor data is one of its kind of application in event detection and allows for fast, real-time monitoring of events.
Some of the first works for activity monitoring used a combination of handcrafted features rather than automated feature extraction for representing the data. Neural networks were used primarily as a classification mechanism [11, 12]. Using larger, deeper networks for automated feature extraction and classification was explored later [13] with greater success, at the cost of computational complexity. Advances in Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have resulted in a multitude of frameworks that have tackled the problem of continuous activity monitoring, albeit in a post-processing manner on servers with more computational power than a typical IoT edge node.

We refer the reader to [14] for a more detailed review on using deep learning networks for continuous activity monitoring using consumer-grade ubiquitous sensing devices.

2.2 Implementing Deep Neural Networks on Constrained Platforms

While progress has been made on successfully implementing CNNs and other machine learning algorithms on FPGAs [15, 16, 17, 18, 19, 20, 21, 22], the effort has been largely focused on creating platforms and frameworks that allow seamless transition from the traditional CPU/GPU-based design to the FPGA platform. There have been several different approaches to address the problem of execution of CNNs and other machine learning algorithms traditional FPGA platforms such as Xilinx Stratix and Virtex7 as well as Altera Arria FPGAs. One approach is to design computationally less intensive variations of CNNs such as Binary Neural Networks (BNN) [23]. Some have proposed the use of vectorization to provide a unified representation for all operations to optimize the computation process [22] as well as providing quantizing model parameters [24]. Some have taken an optimization-based approach such as reconfigurable designs for streaming datapaths for
the various layers in a CNN [15], novel optimizations for convolution operations [20], and optimizing loops in the convolution operations [21]. Such works have largely focused on the implementation and optimization of convolutional neural networks on traditional embedded platforms whose hardware specifications are on the scale of CPUs and as well as several mid-level GPUs, having working memory from 1GB DDR3 RAM to even 48GB of RAM. These approaches perform the training of the network "ex-situ", without being aware of the target platform and its computational constraints.

On the other hand, there is a family of approaches that target the training process of a specific network while being aware of any future computational constraints. Such approaches are primarily based on quantization and offer a way to “compress” or reduce the model size without compromising the accuracy of the network. Some common approaches are to reduce the parameters through pruning [25], knowledge distillation [26, 27], and parameter quantization [28]. The notion of quantization-aware training [29, 30, 31] has also gained momentum, where the model’s parameters are tuned with mixed-precision weights. This training process allows the model to learn compact features and associated parameters, which could otherwise be lost during post-training quantization or pruning.

2.3 Physically Unclonable Functions

Physically Unclonable Functions [7, 32], or physical random functions, are an embodied version of physical functions that maps an external stimulus (the challenge) to a random, but a repeatable response. The physical function is characterized by the inherent randomness introduced during the manufacturing process and is nearly impossible to replicate given a polynomial amount of resources. A PUF model’s characteristics are best expressed through the collection of challenge-
response pairs (CRPs) and hence form the basis of most, if not all, PUF-based security protocols. PUFs can be categorized into two types based on the number of valid CRPs, namely weak PUFs and strong PUFs [33]. A PUF is said to be a \textit{weak} PUF if it has a fixed, small set of CRPs that are valid and are assumed to be access restricted. \textit{Strong} PUFs, on the other hand, leverage large amounts of the inherent unpredictability and hence possess a large number of CRPs. They are also considered to have an unprotected physical interface and are more commonly used in security applications. We refer the reader to [33] for an extensive review of weak and strong PUF models.

There have been numerous PUF models introduced and evaluated over the years. The taxonomy of PUF is illustrated in Figure 2.1. Broadly, they can be divided into two major groups - the time-delay based models and the memory-based models. \textit{Time delay-based models} include ring oscillator PUFs and Arbiter PUFs or APUF and its variations such as feed-forward arbiter PUFs. Such PUF models can generate real-time, chip-specific signatures without the need for expensive memory for key storage and thus, have been particularly conducive to device authentication, intellectual property, and data privacy preservation to name a few. \textit{Memory-based PUF} models, on the other hand, exploit the variations between matched silicon devices of memory elements to characterize the inherent random function. Some common bistable memory elements that are exploited for the PUF functions are SRAM, latches, and flip-flops. Again, we refer the reader to [34] for a more detailed review of PUF architectures.

2.3.1 Strong PUF Architectures

A strong PUF can support a large number of complex CRPs with physical access to the PUF for a query such that an attacker cannot generate correct response given finite resources and time [33, 34, 35]. While a weak PUF has only a few CRPs, which makes it difficult for the attack and
prediction techniques. They are mostly used in secret key storage, hence not known to the public. In this work, we consider strong PUF. The number of CRPs of strong PUFs can grow exponentially depending on the number of module blocks available for generating responses for a large number of corresponding challenges. Error due to noise in the response of PUF can be minimized using helper data [36, 37]. There are different types of strong PUF, some of which are Arbiter PUF, XOR Arbiter PUF, Lightweight PUF, and FeedForward PUF.

The Arbiter PUF [32] is a standard PUF configuration consisting of $n$ stage sequences of multiplexers. Two signals are sent simultaneously through these $n$ stages as shown in Figure 2.2. The path is determined by the external bits for $n$ stages. The last stage, consisting of an arbitrary element of a latch, determines whether the upper or the lower signal arrived first and its respective output which is either one or zero. Here the external bit is considered to be the challenge and the output is the response $Y$.

The XOR Arbiter PUF [38], is composed of $n$ individual Arbiter PUF with $k$ stages. The same challenge is applied to both the Arbiter PUF and the responses are XORed together to produce
Figure 2.2: Architecture of an Arbiter PUF (Reproduced from [32]).

A single output response. A 2-stage XOR Arbiter PUF is shown in Figure 2.3. Depending on the \( n \) stages there can be different XOR Arbiter PUF denoted as \( n \)-XOR Arbiter PUF.

The Lightweight PUF [39] is similar to the XOR Arbiter PUF, there are \( n \) stages of Arbiter PUF in parallel with \( k \) stages, each stage producing the output. These outputs are XORed to produce the multi bit output. Also, the external bits or challenges can be applied \( n \) stages of the Arbiter PUF.

The inherent CRPs of strong PUFs can grow exponentially depending on the number of module blocks available for response generation with large possibilities of corresponding challenges. Error induced by noise on the response of PUF can be minimized using helper data [36, 37]. For completeness, we assume such an error-correction mechanism incorporating temperature, voltage, and aging variations are already present in the PUF to be cloned. A strong PUF does not contain a read-out protection scheme assuming an attacker has to enumerate a large number of CRPs. Hence, it makes an invasive attack infeasible while impelling attacker to apply ML-based techniques to be
The linear additive behavior of Arbiter PUF (APUF) has made it an ideal target for ML attack. Hence, higher non-linearity in a given PUF architecture can improve the uniqueness and randomness with increased defense against modeling attack. Other approaches to ML resistant PUFs have been randomized challenges [40], obfuscation [41, 42], and sub-string based challenges [43]. Randomized challenges [40] to PUF and obfuscating PUF responses [41] have also been proposed. Rostami et al. [43] presented a prover-verifier framework for successful authentication based on a subset of response substring. Vijayakumar et al. [44] proposed to utilize bagging and boosting ML algorithms to improve the accuracy of classifiers given sufficient entropy of cascading PUFs.

The majority of works describing ML resistant PUFs employ clearly defined architecture and adequately large CRPs for the training process. The randomness and uniqueness, instead,
deteriorate substantially when CRPs that do not belong to original CRPs for a particular PUF is used as the case we are tackling in this work.

2.3.2 PUF-based IoT Security

Physical Unclonable Functions (PUFs) have, increasingly, been proposed as the basis for node security in the IoT framework [45, 46, 47, 48, 49, 50]. PUF-based IoT node security has primarily been implemented in two ways - CRP-based authentication and PUF-based key generation [38]. In the latter, a PUF’s response is typically used to create secret keys for use in traditional cryptography. The PUF’s response to a given challenge (processed through an error-correcting circuit) is typically hashed to generate the secret keys. The former approach i.e., CRP-based authentication, is more widely used, especially with strong PUF models, to create robust authentication protocols. The resulting authentication protocol involves the evaluation of the identity of a PUF model by a central authentication server by applying a set of pre-defined external challenges and validating the resulting response. The CRPs are collected in an enrollment phase before deployment, and the resulting database forms the basis of authentication during deployment.

2.3.3 Encryption Protocols for IoT Node Authentication

With the use of CRPs for IoT node authentication, the need for encryption protocols has risen due to the need for added security from eavesdropping protocols. The use of encryption protocols in IoT node communication and authentication has seen staggering rise [51, 52, 53, 54, 55]. In summary, the encryption protocols used are the Data Encryption Standard (DES) [56] and the Advanced Encryption Standard (AES) [57]. While there has been successful cryptanalysis of DES, it still takes an extraordinary amount of compute and access to data to achieve it, whereas there has
Table 2.1: Comparison of existing ML attacks (best results through simulation) and modeling resistant technique on strong PUFs

<table>
<thead>
<tr>
<th>Strong PUFs</th>
<th>ML Attack Algorithms</th>
<th>ML resistance technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>APUF</td>
<td>LR [59], SVM [60], ANN [60],</td>
<td>Challenges Randomization [40], Multiple Input Signature</td>
</tr>
<tr>
<td></td>
<td>Reverse Fuzzy Extractor [61]</td>
<td>Register [62], Response Obfuscation [41]</td>
</tr>
<tr>
<td>Feed-Forward APUF</td>
<td>ES [59], Multi-layer Perceptrons [63]</td>
<td>Gradient Boosting [44]</td>
</tr>
<tr>
<td>XOR APUF</td>
<td>LR [59]</td>
<td>Function Composition [44], Response Obfuscation [41]</td>
</tr>
<tr>
<td>Lightweight PUF</td>
<td>LR [59]</td>
<td></td>
</tr>
<tr>
<td>Ring Oscillator PUF</td>
<td>ES [64]</td>
<td></td>
</tr>
<tr>
<td>Current-based PUF</td>
<td>ES [64]</td>
<td>SVM [65]</td>
</tr>
</tbody>
</table>

not been a successful attack on the 128-bit AES encryption protocol. While encryption protocols have been used extensively in IoT node communication, it requires some semblance of computation to get working. Hence, there have been other protocols proposed to overcome such computation power such as obfuscated CRP [42] and substring matching [58], to name a few. In this work, we consider the encryption protocols AES and DES as the encryption mechanisms used for encrypting the CRPs in the IoT framework.

2.3.4 Machine Learning-based Attacks on PUF Models

The widespread introduction of PUF models into IoT node authentication has seen an increase in approaches that attempt to test their effectiveness through attacking or cloning the PUF model. Cloning a PUF model typically involves the fitting of a complex mathematical function to capture the correlation between the input challenge and the corresponding PUF response. There have been several approaches, including leveraging machine learning models and physical modeling. Given the growing popularity of PUF-based authentication, there have been numerous attempts to test the effectiveness of the approach, primarily through mathematical modeling of the PUF’s characteristic function. Rührmair et al. [59] proposed an ML-based attack on strong PUFs based on a predictive model. The authors were able to clone the functionality of the underlying PUF.
given the PUF model by evaluating model parameters using Logistic Regression (LR) with RProp and Evolution Strategies (ES). Though the method was quite successful in cloning, the attacker needs to know the underlying PUF architecture and the corresponding signature function. While it is reasonable to assume that CRPs can be obtained by eavesdropping or other interfaces [33], it is not always possible to ascertain the underlying PUF model without physical access to the PUF. Although the presented attacks work better under a given PUF size and architectural complexity, an attacker should have the idea of underlying PUF architecture to make the generated clone samples match the statistics of the real CRPs. The other type of approach [66, 67] involves physical access to the PUF model beyond just knowledge about PUF architecture and model. They typically involve the use of machine learning approaches to model the PUF response by exploiting the physical characteristics obtained through side-channel approaches. Recently, efforts have shifted to a combined ML and side-channel (timing and power) to present an improved hybrid attack surface [6, 68]. A mathematical model-free ML attack using PAC (Probably Approximately Correct) learning framework has been proposed in [69]. The authors presented that an influential bit, if present in stable PUF response, can predict the future response corresponding to a challenge with low probability. Table 2.1 gives an overview of the existing ML attacks and modeling PUFs that are resistant to the ML attacks.

2.4 Chapter Summary

In this chapter, we reviewed current work in designing and implementing various components in the Internet of Things framework such as applications, edge node implementation, and security. Specifically, we first summarized existing IoT and ubiquitous sensing approaches to continuous activity monitoring. We then explored the various mechanisms used to implement and accelerate
deep neural networks on resource constrained platforms. We also introduced the basic mechanisms of physically unclonable functions (PUFs) and the various architectures. We finally discussed existing work on integrating PUFs into the authentication protocols in IoT frameworks and analyzed their susceptibility to various cloning attacks, including machine learning-based ones.
Chapter 3: Continuous Ubiquitous Activity Monitoring

In this chapter, we propose an unsupervised, continuous ubiquitous activity monitoring framework based on self-supervised representation learning and a probabilistic segmentation algorithm. We demonstrate that the framework can be used to perform unsupervised sleep state segmentation. We discuss how the framework can be extended to other tasks such as monitoring activities of daily living from accelerometer data.

3.1 Introduction

Sleep is a vital process in the maintenance of health and well-being. The lack of sleep, both in duration and quality, is a prevalent issue and affects over 50 million people in the United States alone [70]. The effects of poor or deprived sleep can have a major impact on the daily, healthy functioning of the human body and can lead to serious issues such as obesity, diabetes, and heart disease to name a few [71]. However, given the importance of sleep, the major way to test and measure the quality of sleep is polysomnogram (PSG), an expensive and invasive test that can require tremendous amount of resources such as a sleep lab, physician, and equipment to monitor physiological signs and are available only for an assessment of a selected few nights. Long term continuous monitoring is not quite an option. However, the emergence of the Internet of Things (IoT) paradigm has enabled the deployment of low, inexpensive consumer devices that can collect non-invasive information such as movement, heart rate, etc. These devices offer an alternative to the PSG test and hence can enable long-term, continuous monitoring of sleep for maintaining a healthy
lifestyle in modern times. However, there are some significant challenges to continuous sleep state monitoring. First, the time series data can have both local and global patterns that need to be captured in learned representations. Second, the amount of data to be analyzed can be enormous since the data (accelerometer, heart rate, etc.) is sampled at very high frequencies and can result in hundreds of readings per minute. Finally, each of these data samples need to be labeled by experts in the field, which can be very expensive and hard to obtain.

However, in traditional IoT or rather IoMT (Internet of Medical Things) frameworks, the sensors act as data collection nodes which transmit the collected data through a myriad of servers, routers, and the network gateway before it is processed offline and the actionable rules and applications can be obtained. This information transmission is often done through wireless networks. This introduces some security concerns such as eavesdropping and man-in-the-middle attacks, which are quite common security concerns and hence require robust security protocols for ensuring the integrity of the transmitted data.
The problem of sleep state recognition and segmentation has been largely tackled through supervised learning [3, 4]. These approaches, while working well, require large amounts of expert-annotated, labeled data which can be very expensive to obtain. Additionally, the collection of such an amount of data involves the capture and transmission of private, sensitive information such as motion characteristics, location, heart rate, etc., and can cause serious security issues if there is any breach in security. With advances in machine learning and hardware security, the integrity of data can be compromised and the continuous collection of highly sensitive information can be liable to issues. Decentralized learning [5] has emerged as a viable alternative, but requires fast and efficient learning algorithms that can work at scale and on-device, including on computationally constrained edge nodes. Additionally, the annotation of such large-scale data can be difficult to obtain and is not conducive for scaling to millions of users. Hence, the need for self-supervised or unsupervised approaches to sleep state segmentation is increasing.

In this work, we tackle the problem of unsupervised sleep state segmentation in streaming data through self-supervised representation learning. We propose a two-pronged approach to the problem. First, we propose the use of continuous predictive learning to learn robust temporal embedding of streaming data through self-supervision. Second, we propose an unsupervised, probabilistic algorithm to segment the continuous stream of motion and photoplethysmography heart rate data into its constituent sleep states, without any labeled data.

There have been several approaches to sleep state recognition and segmentation using motion and photoplethysmography data. The common approach [3, 4] uses supervised machine learning algorithms that use contextual, motion-based, and photoplethysmography features to classify pre-segmented sleep epochs into the different sleep states. Such approaches consider the task of sleep
state classification that assume that the time series data is already pre-segmented into epochs and hence the task is constrained to that of recognition and does not involve the segmentation of long, time series data. We, on the other hand, tackle the problem of sleep state segmentation and hence, have to both temporally localize and recognize the sleep state.

Our contributions in short are: (1) we are among the first to tackle the problem of sleep state segmentation in streaming data, (2) we propose an unsupervised segmentation framework that can segment long, untrimmed sequences of motion and heart-rate data without any labels or supervision, (3) we show that the continual learning approach can be used in decentralized learning frameworks to enhance the security and privacy of data, and (4) we show that the proposed framework is sufficiently lightweight and can be implemented on constrained platforms without trading off accuracy for latency by leveraging current advances in hardware accelerators for deep learning.

The rest of this chapter is organized as follows. We first introduce the proposed unsupervised sleep state segmentation framework. We then provide quantitative and qualitative analysis of our approach on real-life accelerometer data. We then conclude with discussion on the approach and future steps to extend the proposed framework.

### 3.2 Unsupervised Sleep State Segmentation

In this section, we introduce the proposed unsupervised sleep state segmentation framework. We present a feature representation learning approach for capturing temporal representations. We then introduce an unsupervised segmentation algorithm and conclude with discussion on the implementation and challenges on constrained platforms.
3.2.1 Local Temporal Feature Representation Learning

Time series data, especially acceleration and photoplethysmography heart rate data, can be very long temporal sequences. Hence it is essential to learn robust representations of the local temporal structure of the data. In this work, we refer to the data contained in one second to be a representative of the local temporal structure. Since motion (acceleration) and photoplethysmography data can be highly stochastic, the one second interval allows us to capture the underlying, sequence-invariant patterns within the raw input data. We learn the local representations using an autoencoder framework [72], a neural network architecture trained to learn abstract, concise representations through an encoding-decoding mechanism. To leverage advances in convolutional neural network (CNN) [73] acceleration on constrained platforms [15, 19, 74], we train a convolutional autoencoder network with two networks (an encoder and a decoder) working in tandem to learn an encoded representation called a latent space. The encoder learns to compress the input data into an abstract, encoded representation that captures underlying pattern in the input data. The decoder network is trained to reconstruct the original input from this encoded representation. The autoencoder network is trained to minimize the reconstruction loss, which is framed as the $L2$ difference between the real and the reconstructed data. The encoder networks consists of three convolution blocks consisting of two convolution layers each. The convolution blocks interspersed with max pooling and batch normalization [75] layers to reduce the dimension and improve training stability. Since the time series data does not contain a spatial dimension, we use 1-D convolutions to capture sequence-invariant representations of the input sequence.
3.2.2 Capturing Long Range Temporal Dependencies

While the representations learned using the autoencoder framework capture short-term dependencies, they are not enough to capture the temporal dependencies in the input photoplethysmography data, which can extend to several hours of data. It is also not feasible to increase the input window size for the encoder network without increasing the number of parameters in the encoder and decoder networks, which can increase non-linearly with increase in the input dimension. Hence we capture the long-term temporal dependencies through a predictive learning framework [76] using an Long Short-Term Memory (LSTM) based predictor [77]. We train a predictor network to continuously predict the features of the input data at the next time step. The decoder network is then trained (or rather fine-tuned) to reconstruct the predicted data. This is illustrated in Figure 3.2. The hidden state of the predictor network is used as the temporal embedding for time $t$, since it captures the past temporal dependencies and highly influences the predictions of the LSTM network. Note that this is different from Aakur et al. [76], who train the network to predict the future feature space as opposed to the actual data. The reconstruction error (framed as the sum squared error between the prediction and observed data) is used to train the encoder, decoder, and predictor.
networks. Hence the training objective is given by

$$\arg\min_{\theta_e, \theta_d, \theta_p} \sum_{i=1}^{n} \|x_{t+1} - \hat{x}_{t+1}\|_2^2$$

(3.1)

where $\theta_e$, $\theta_d$, and $\theta_p$ represent the parameters of the encoder, decoder, and predictor networks respectively; $x_{t+1}$ and $\hat{x}_{t+1}$ are the observed and predicted inputs at time $t + 1$, respectively.

The reason we use predictive learning while many other embedding methodologies are available is because the predictive learning approach offers three major advantages: (1) it allows for training on real-time, streaming data and does not require us to store large amounts of data, which can lead to security and privacy issues, (2) the prediction stack can be made to be lightweight and contain fewer parameters for optimization, which is essential for real-time processing in constrained platforms available in a typical IoT framework, and (3) the predictive learning approach is naturally conducive to decentralized learning [5] where each instance of the networks can be trained separately and combined later for improved performance.

3.2.3 Sleep State Segmentation

The final step in our framework is sleep state segmentation in streaming data which is used to segment the time series data into the segment based on the sleep state. We use the temporal embeddings constructed through the network described in Section 3.2.2 and provide an initial, coarse segmentation by assigning each time step (one second intervals) to $k$ clusters. Each cluster represents each sleep state such as awake, non-Rapid Eye Movement (NREM), Rapid Eye Movement (REM), etc. We define this to be a coarse segmentation because it does not take into account the temporal coherence, the valid state transitions, and uncertainty of the classification into account.
We perform this classification by modeling a Gaussian Mixture Model (GMM) [78] with $k$ mixtures. We optimize the GMM using the Expectation-Maximization [79] and the resulting classification is a $k$-dimensional vector with probability of the input belonging to each of the $k$ classes. This is different from $k$-means clustering, which provides a hard assignment and does not return any probabilities and allows us to capture any uncertainty in classification.

The use of coarse segmentation would be a naïve approach which can result in non-homogeneous clusters of sleep states. To alleviate this, we introduce a temporal smoothing function that allows us to present temporally coherent, homogeneous segmentation of the input sequence. Based on the Viterbi algorithm [80], the smoothing process is a probabilistic temporal linking algorithm which takes the coarse segmentation as input and outputs the final segmentation. We denote the $i$-th cluster from time $t$ by $d_t$. We compute the temporal affinity score between two time steps $d_t$ and $j_{t+1}$ as

$$S_c(d_t, d_{t+1}) = (1 - \beta)E_c(d_t) + \beta E_c(d_{t+1}) + \psi_{d_t, d_{t+1}}$$  \hspace{1cm} (3.2)$$

where $E_c(\cdot)$ is the class confidence score of the given time step, $\beta$ is a temporal memory factor, and $\psi_{d_t, d_{t+1}}$ is the probability of switching the labels between $d_t$ and $d_{t+1}$. We model $\psi(\cdot)$ as the prediction error defined in Equation 3.1 between the time steps $t$ and $t + 1$. The time steps with maximum total temporal affinity are combined to form a new cluster and their probability scores are averaged. The smoothing algorithm is applied $n = 15$ times or until less than 5% of the time steps change their labels.
3.2.4 Network Architecture and Implementation Details

The overall structure of the proposed temporal autoencoder is illustrated in Figure 3.3. We leverage the advances from the VGG-Net architecture [73] and use small filters (1-D filters with receptive field of 3 and height of 1) to convolve across time. We divide the network into blocks of convolutional layers interspersed with max pooling layers. The network architecture is based closely on the VGG-11 architecture, which we find to be extremely friendly to computationally constrained platforms (Chapter 4). We progressively reduce the dimensionality of the input through temporal pooling, which is implemented as the traditional pooling operation with a height of 1. The decoder network is the mirror image of the encoder network. We pre-train the encoder and decoder network to reconstruct the input signal, sampled every second which is approximately 45 accelerometer readings. We also introduce some noise randomly in the input to the encoder in order to help make the network invariant to any temporal perturbations due to reading errors and calibration. This allows the network to learn robust representations that allow for more consistent predictions. The predictor network is an LSTM network with a hidden size of 128. We do not use any attention mechanisms since the goal is an auto-regressive prediction. We find that adding attention mechanisms harm the temporal prediction ability of the LSTM network. This effect could,
arguably, be attributed to the loss function which will heavily penalize any deviations from the input, whereas the attention mechanism will force the network to focus on the most discriminative parts.

3.2.5 Implementation on Constrained Platforms

While the segmentation ability of the approach is essential, we must also allow for effective implementation of the network on constrained platforms, which is typical of most computational nodes in an IoT framework. To account for the computational constraints, we make some essential design choices and implementation tweaks in our framework. The first, and most important, one is the use of 2-D convolutional kernels in the implementation of the network. While the data is one-dimensional and 1-D convolution operations are more efficient, we find that existing work on hardware accelerators for deep learning have primarily concentrated on 2D CNNs. Hence, we approximate the 1-D convolution operation by making the height of the convolution kernel as 1.

Second, we find that the use of pooling layers has a considerable effect on the energy consumption on the PYNQ platform [81], when the pooling operation requires padding for execution. Hence, we ensure that the dimensions before the pooling layer are such that it does not require any padding for execution. Finally, we observed a direct correlation between the network latency and the layer capacity of fully connected layers. Hence, we approximate the computation of the fully connected layers by manipulating (e.g., reshaping) the learned parameters of the fully connected layer (the weight matrix $W$) into convolutional filters. This allowed us to reduce the memory footprint and latency by factors of 2 and 1.75, respectively. We perform the same approximation with the inner mechanics of the LSTM layer.
3.3 Evaluation and Analysis

In this section, we present our experimental evaluation of the proposed framework. We perform our experiments on the data collected by Walch et al. [3]. The data contains accelerometer and photoplethysmography data collected from 39 participants using an Apple Watch for an eight hour sleep period. The subjects were placed under 7- to 14-day ambulatory recording period. The data includes motion data in the form of accelerometer readings and heart rate obtained by photoplethysmography built into the Apple Watch. We use the accelerometer (in units of g) and heart rate data to segment the data into its constituent sleep states. We sample the accelerometer every second, which results in around 50 readings per second. For efficient computation and to maintain equality for all sequences, we sample 48 readings in each time instant. For time instances with less than 48 readings, we interpolate within the readings to obtain the set number of readings per second.

Evaluation is done using Monte Carlo cross-validation as described in [3]. We report the accuracy as the evaluation metric per time step (one second interval). This is different from the evaluation protocols in [3], where the evaluation is done per sleep epoch, which typically lasts 30 seconds and requires highly expert annotations. Our setting is a more realistic evaluation setting for continuous sleep state segmentation on consumer grade devices, which do not have such expert annotations. We evaluate against several supervised baselines such as logistic regression, random forest, neural networks as well as unsupervised baselines such as $k$-means. It should be noted that we use the same, self-supervised features as input to each of the algorithms, whose parameters are the only training objective in the fine-tuning stage.
Table 3.1: Performance of different approaches on sleep vs. wake segmentation task.

<table>
<thead>
<tr>
<th>Supervision</th>
<th>Approach</th>
<th>Accuracy (%)</th>
<th>Wake Correct (%)</th>
<th>Sleep Correct (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>LR</td>
<td>0.71</td>
<td>0.42</td>
<td>0.95</td>
</tr>
<tr>
<td></td>
<td>RF</td>
<td>0.67</td>
<td>0.38</td>
<td>0.92</td>
</tr>
<tr>
<td></td>
<td>NN</td>
<td>0.72</td>
<td>0.41</td>
<td>0.94</td>
</tr>
<tr>
<td>None</td>
<td>k-means</td>
<td>0.59</td>
<td>0.3</td>
<td>0.89</td>
</tr>
<tr>
<td></td>
<td>GMM</td>
<td>0.54</td>
<td>0.23</td>
<td>0.87</td>
</tr>
<tr>
<td></td>
<td>GMM + TS</td>
<td>0.63</td>
<td>0.31</td>
<td>0.91</td>
</tr>
</tbody>
</table>

Note: Performance reported across all subjects in the data. Best accuracy for GMM+TS and LR are 81.8% and 85.6%, respectively. The recall rates are 80.5% and 83.7%, respectively.

3.3.1 Quantitative Results

We evaluate our approach and various baseline algorithms on two different tasks. The first task is to segment the streaming data into two categories - sleep and wake. Here the goal is to correctly identify segments of time when the person is awake and when the person is sleeping. We summarize the results in Table 3.1. We categorize the approaches into two categories based on how much supervision (i.e., labeled data) is required for the approach. If the approach requires labeled data then it is termed as requiring “full supervision” and “no supervision” otherwise. It can be seen that using our temporal embedding allows the fully supervised baselines to perform very well and narrows the gap between the fully supervised and unsupervised approaches. It can be seen that the use of temporal smoothing greatly improves the performance of the GMM-based approach. Our best performance is 96.2% accuracy for patient ID 0 and our best performance per evaluation set (10 patients) is 81.8%, which is state-of-the-art on the sleep segmentation task, especially on computationally constrained platforms. The recall rate is 83.7%. We also evaluate the approaches on the much harder task of segmenting into three classes - wake, NREM sleep, and REM sleep. We summarize the results in Table 3.2. This is a much harder task since the features
for REM and NREM sleep are very closely related and can cause large amounts of uncertainty in the classification task. We obtain our best accuracy (78.9\%) on patient ID 0, with our best performance per evaluation set (10 patients) as 63.2\%, which is state-of-the-art on the wake vs. REM vs. NREM sleep segmentation task. The recall rates are 76.9\% and 70.3\% for wake vs. sleep and wake vs. REM vs. NREM segmentation tasks respectively. We report the accuracy per patient in Figure 3.4, sorted by the accuracy. Also, the data for this task is highly imbalanced and require careful fine-tuning of the training process to obtain good results. We ensure that there is a balanced set of samples for all classes at each training epoch. It can be seen that the representations learned through predictive learning provides good results and further narrows the performance gap between supervised and unsupervised methods.

The difference between supervised and unsupervised models to be noted is that there is a significant gap between the supervised approaches (\sim 9\% on the wake vs. sleep task and \sim 6\% on the wake vs. NREM vs. REM task). This, however, is a trade-off between the amount of expensive,
Table 3.2: Performance of different approaches on sleep vs. NREM vs. REM segmentation task. TS refers to temporal smoothing.

<table>
<thead>
<tr>
<th>Supervision</th>
<th>Approach</th>
<th>Wake Correct (%)</th>
<th>NREM Correct (%)</th>
<th>REM Correct (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>LR</td>
<td>0.57</td>
<td>0.51</td>
<td>0.53</td>
</tr>
<tr>
<td></td>
<td>RF</td>
<td>0.53</td>
<td>0.40</td>
<td>0.44</td>
</tr>
<tr>
<td></td>
<td>NN</td>
<td>0.54</td>
<td>0.39</td>
<td>0.5</td>
</tr>
<tr>
<td>None</td>
<td>k-means</td>
<td>0.49</td>
<td>0.51</td>
<td>0.37</td>
</tr>
<tr>
<td></td>
<td>GMM + TS</td>
<td>0.50</td>
<td>0.42</td>
<td>0.33</td>
</tr>
<tr>
<td></td>
<td>GMM</td>
<td><strong>0.51</strong></td>
<td><strong>0.44</strong></td>
<td><strong>0.43</strong></td>
</tr>
</tbody>
</table>

Note: Performance reported across all subjects in the data. Best accuracy for GMM+TS and LR are 63.2% and 67.3%, respectively. The recall rates are 70.3% and 76.9%, respectively.

expert-annotated data required to train these models. However, the use of self-supervised temporal embedding can significantly reduce this performance gap. The output of the unsupervised models can be used to pre-train supervised models for reducing the over-reliance on labeled training data and is an active area of research.

We also evaluated the power consumption and latency of the proposed approach on a highly computationally constrained platform, namely, the PYNQ board (see Section 4.5.2 for details of the board). The results are reported in Table 3.3. We observe an average inference latency of 1.75 seconds per time step of one second and power consumption of 3.27W. However, breaking down the latency and power consumption per module in the pipeline, we observe that the latency for the encoder network, predictor network, and the GMM module are 407 ms, 616 ms and 727 ms, respectively. Better hardware acceleration modules can further improve the latency and move towards real-time segmentation.
Table 3.3: Performance of the model on different platforms.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Hardware Platform</th>
<th>Latency (s)</th>
<th>Power (W)</th>
<th>Efficiency $J \times s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>Intel Xeon E5-2680 v3 CPU</td>
<td>1.24</td>
<td>38.9</td>
<td>48.2</td>
</tr>
<tr>
<td>Tensorflow</td>
<td>NVIDIA 1080 GPU DDR5 8GB</td>
<td>0.47</td>
<td>235</td>
<td>110.5</td>
</tr>
<tr>
<td>This Work</td>
<td>PYNQ FPGA 512 MB DDR3</td>
<td>1.75</td>
<td>3.27</td>
<td>5.7</td>
</tr>
</tbody>
</table>

3.4 Chapter Summary

In this chapter, we presented an unsupervised sleep state segmentation algorithm that uses a self-supervised temporal feature learning paradigm that requires no labeled data. The predictive learning framework is naturally conducive towards decentralized, federated learning that can help preserve the privacy and reduce the load on the network. We show that the unsupervised segmentation approach performs competitively with fully supervised baselines and even outperforms some supervised baselines. We aim to improve the feature learning capabilities of the predictive learning framework to further reduce the gap between the fully supervised and unsupervised baselines and integrate into a decentralized learning framework using multiple IoT edge nodes.
Chapter 4: Implementation of CNNs on Constrained Platform

In this chapter, we analyze the properties of different deep neural network layers and provide design-time optimizations for implementing the architecture proposed in Chapter 3 on constrained platforms such as the PYNQ board. We leverage the unique properties of this platform, and through accurate design choices and optimizations, implement highly compact and efficient versions of four common CNN architectures that can be extended to various deep learning models such as LSTMs [77], in addition to CNNs.

4.1 Introduction

Reconfigurable computing architectures such as Field Programmable Gate Arrays (FPGAs) and FPGA computing systems have emerged as a feasible platform for on-chip learning in IoT edge devices. These devices often have low power consumption but have limited parallel processing capabilities compared to more traditional ML platforms, such as workstation Graphics Processing Units (GPUs) or many-core processors. Nevertheless, few studies in the literature have sought to design and optimize complex ML tasks on constrained platforms, and as such, most studies report very high performance and inference accuracy, at the cost of high power consumption, significant memory requirements, and large device footprints [15, 19, 74, 82, 83, 84], leading to relatively low energy and memory efficiency. Instead, we propose design optimizations for four Convolutional

\footnote{This chapter was published in International Conference on VLSI Design 2020 (VLSID) (Accepted and to appear).}
Neural Network (CNN) architectures, and present results on a small, low power heterogeneous computing platform that is representative of systems that may be used in constrained IoT devices.

CNNs are compute- and resource-intensive, containing many trainable parameters often in the range of tens to hundreds of millions and perform several complex classes of tasks on a given input, such as two-dimensional convolutions and subsampling. Given the scale of the computational and memory requirements, and the desire to implement on-chip deep neural networks for faster system response times, reduced network bandwidth requirements, and privacy benefits, it is critical to investigate network architectures and optimizations in implementation that is amenable to low-power, constrained environments, without sacrificing prediction or classification accuracy.

Field Programmable Gate Arrays (FPGAs) are amenable to the implementation of various ML algorithms and can yield power and performance improvements over general-purpose processors. However, mapping software implementations to FPGAs using High-Level Synthesis (HLS) is a significant challenge. Obtaining the maximum power and performance benefits often requires additional direction that automated tools alone fail to provide. The PYNQ environment from Xilinx [81] provides an alternative to the traditional FPGA work-flow, instead of leveraging a heterogeneous processing environment for Python scripting. Several recent works explored implementing learning and big data analytics algorithms on the platform, including Spark [85], edge detection [86, 87], video processing [88], and recurrent neural networks [89]. In this work, we leverage the unique properties of this platform, and through accurate design choices and optimizations, implement highly compact and efficient versions of four common CNN architectures: LeNet, AlexNET, VGG-11, and VGG-16. Implementations are validated using standard datasets, MNIST [90] and CIFAR-10 [91],
and a precision power supply with precision measurement capabilities is used to evaluate energy efficiency accurately.

In short, we make the following novel contributions:

1. It analyzes the properties of different CNN layer types with respect to power consumption, memory footprint, and latency.

2. It describes four optimized implementations of CNNs on a low power heterogeneous computing platform, and shows significant improvement in power consumption with minimal effect on latency, and maintaining the state-of-the-art classification accuracy.

3. It presents design-time optimizations that enable the implementation of a CNN architecture with a depth greater than five convolutional layers using no more than 512 MB of available memory. Such optimizations allow designers to use significantly less memory than previously reported implementations, increasing throughput on smaller platforms with a strict area, power, and memory constraints.

To the best of our knowledge, this is the first such example of CNN implementations on a low power, resource-constrained platform with performance comparable to high power devices with significantly more resources. As such, the techniques discussed in this work can be applied to designs for learning and classification in constrained environments such as mobile, automotive, IoT, or medical devices. The rest of the chapter is organized as follows: Section 4.2 provides a brief introduction to the computational model of convolutional neural networks. Section 4.3 discusses the design choices, trade-offs, and optimizations used to implement four CNN architectures on the PYNQ FPGA board. Section 4.4 describes the network-specific optimization that we used to
implement four CNN architectures. Section 4.5 presents the datasets used, the evaluation platform, and power measurements methodology. Section 4.6 details the experimental setup and provides a comprehensive evaluation of network performance, resource utilization, and power consumption. We conclude in Section 4.7 with future directions for the research.

4.2 Convolutional Neural Networks (CNNs)

In this section, we provide a brief introduction to the structure of Convolutional Neural Networks (CNNs) and describe the challenges in designing and optimizing networks for low power architectures with limited resources. We begin with a discussion on convolutional layers followed by an analysis of subsampling techniques used to reduce the dimensionality of the features, a discussion on the use of dense layers, and end with a brief description of the training process. An example of the typical CNN data flow is shown in Figure 4.1. Given an input image, activation or feature maps are constructed through the convolution of the input and the convolutional kernels. The resulting feature maps are sent through a subsampling layer or “pooling” layer to reduce its dimensionality. The process is repeated until the features reach a suitable dimensionality, after which the non-linearity in the high dimensional features are captured through fully connected “dense” layers and are used for the classification.
4.2.1 Convolutional Layers

Convolutional layers form the core functionality of CNNs and are one of the more computationally intensive processes in the network. The convolutional layer has a set of parameters representing a set of learnable filter kernels called its receptive field. Each filter is designed to be small in the spatial dimension, but with considerably greater depth as it extends through the entire input image. Each filter is convolved across the spatial dimensions (height and width) of the input image to produce an activation map that is descriptive of the filter’s responses. As the layers are trained, the network tends to learn filters sensitive to lower-level visual features such as an edge of some orientation at the initial levels to highly complex features at higher layers of the network. Such activation maps represent the visual features captured from the image at a particular level. Formally, the convolutional operation at a given layer $\ell$ and a kernel of size $k \times k$ is given by

$$x^\ell_{ij} = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} W_{m,n} y^{\ell-1}_{(i+m),(j+n)}$$

(4.1)

where $x^\ell_{ij}$ is the output at point $(i,j)$ on the activation map at layer $\ell$ and $y^{\ell-1}$ is the output from layer $\ell - 1$.

For activation function each convolutional layer computes linear regularities in the input through its convolutional operation on the image space. However, it is essential to introduce nonlinearity into the features that are learned from the convolutional layer. This nonlinearity is typically accomplished through the use of an activation function. Common activation functions include $tanh$, $sigmoid$, and Rectified Linear Unit (ReLU). ReLU activations have proven to be more effective in enabling faster training times in networks as well as helping alleviate the vanishing gradient prob-
lem, a condition in which training is increased due to slow propagation of the gradient through the layers [92].

The ReLU layer applies the function \( f(x) = \max(0, x) \) to all values in its input volume. Activation layers increase the nonlinearity of a network without compromising the receptive field of the convolutional layer. We use ReLU activations for the convolutional layer and the sigmoid function for the fully connected layers in our experiments. We use these settings to closely follow the original work to study the impact of the other design choices on accuracy, memory, and latency in a more controlled setting.

4.2.2 Subsampling Layers

Subsampling, or pooling, layers are used to reduce the dimensionality of the activation maps produced by the convolutional layers. This yields a reduction in the total number of parameters and number of computations in the successive layers while helping to control over-fitting the parameters to the specific dataset.

The subsampling is performed across depths of the input and resizes the spatial dimension using a specified operation across the receptive field. The common operations performed are MAX, MEAN, and L2-Norm with a stride of 2 and receptive field 2 × 2. We use max pooling kernels of size 2 × 2 with stride 2 in our experiments.

Dense or fully connected layers are neural networks where there exist fully pairwise connections between neurons in successive layers, but no connections among neurons within the same layer. Fully connected layers can be used to define a family of functions parameterized by the weights of the neurons in the network. Fully connected layers can capture and express the statistical properties
present in the input data, including highly nonlinear functions. The capacity of a fully connected network (multiple dense layers connected in succession) is determined by the number of layers and the number of neurons in each layer. A high capacity equates to the ability to express complex functions [93]. Activation functions, discussed in Section 4.2.1, are also used in fully connected layers, with the sigmoid function being more common. Mathematically, the operations performed by the fully connected layer is given by Equation 4.2,

$$X_j^\ell = \sigma\left(\sum_i W_{ij}^{\ell-1} X_i^{\ell-1}\right)$$

(4.2)

where $X_j^\ell$ is the output of the current layer given the input $X_j^{\ell-1}$ and the weights for the given layer $W_{ij}^{\ell-1}$. $\sigma$ represents the activation function.

4.2.3 Network Architectures

We designed, optimized and evaluated four commonly used CNN architectures in increasing levels of complexity: LeNet [94], AlexNet [95], VGG-11 [73], and VGG-16 [73]. Each of these networks follows a similar design paradigm: a combination of convolutional layers followed by subsampling layers, ending with multiple dense layers. In this section, we describe the architecture of each model.

The LeNet [94] architecture is comprised of a convolutional layer with six $5 \times 5$ filters and a max-pooling layer with stride 2, followed by another convolutional layer with sixteen $5 \times 5$ Filters and a max-pooling layer with stride 2. Three fully connected layers follow, with sigmoid activation for the first two and softmax for the last.
The AlexNet [95] proposed by Kirchevsky et al. was the first successful implementation of a deep CNN with a total of eight layers. Five convolutional layers interleaved with three subsampling layers, one after the first two convolutional layers and another after the last three convolutional layers. The size of the kernels varies from $11 \times 11$, $7 \times 7$, and $3 \times 3$, with the number of filters ranging from 96 to 1024. Like LeNet, AlexNet also contains three fully connected layers with sigmoid activation for the first two and softmax for last.

The VGGNet [73] provided an analysis of the effect of depth vs. the accuracy of a CNN using small convolutional kernels of small receptive fields ($3 \times 3$). We implemented and tested two versions, with 11 and 16 layers each, referred to as VGG-11 and VGG-16, which have shown to achieve state-of-the-art results considering the number of parameters and power consumption. The networks use a combination of two convolutional layers, and three sets of convolutional layers, called convolutional layer blocks are interleaved with max-pooling layers for reducing the spatial dimensionality.

4.2.4 Ex-situ Training

The process and the equations in this section describe the forward computation or the inference process. The process of identifying parameters which can realize the desired functionality and classification accuracy indicates the learning or training phase. Network training is accomplished through an error minimization process called backpropagation. In each iteration, the error is computed as the difference between the target value and the predicted value, then is propagated back through the network to improve accuracy in subsequent iterations. Backpropagation requires approximately twice the number of computations of the forward process for both error propagation...
and parameter updates. This allows designers to use similar computation kernel designs in both the forward and backward processes.

A standard process used to optimize backpropagation is called gradient descent, where the gradient of the error is propagated backward to adjust the weights such that further inferences possess lower error. A subset of the training data, called a mini-batch, is chosen for a training step and the parameters of the network are updated based on its average error. Additionally, another hyperparameter in this is called the learning rate, which allows us to control the extent to which the errors are propagated and hence controls the extent to which the weights are adjusted. In our experiments, this process was done on one GPU (NVIDIA Titan X) as we are primarily concerned with the power consumption and latency of the inference (forward) operations, which is representative of a deployed mobile or IoT application.

4.3 CNN Optimizations and Design Trade-Offs

Every layer type in the CNN architecture has a specific purpose and has its impact on the accuracy, area, latency, and memory footprint. Furthermore, every layer type has several design-time options which allow inevitable trade-offs in network design. We have conducted a study which seeks to determine what the relationship is between these design choices and its practical impact on the resulting CNN in hardware. This design space exploration can help engineers to balance inference accuracy and latency under power and memory constraints.

When designing networks for mobile or IoT systems, these parameters must be balanced to achieve acceptable inference accuracy and latency under the power and memory constraints. In this section, we describe and analyze the various design-time options provided by the multiple layers of a
Table 4.1: Impact of receptive field size on latency, energy consumption, and memory requirements for a $28 \times 28$ input image.

<table>
<thead>
<tr>
<th>Receptive Field Size</th>
<th>Latency (ms)</th>
<th>Pow. (W)</th>
<th>Mem. (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$11 \times 11$ (96)</td>
<td>224.10</td>
<td>2.253</td>
<td>0.137</td>
</tr>
<tr>
<td>$5 \times 5$ (6)</td>
<td>21.50</td>
<td>1.932</td>
<td>0.101</td>
</tr>
<tr>
<td>$3 \times 3$ (64)</td>
<td>31.74</td>
<td>1.857</td>
<td>0.074</td>
</tr>
</tbody>
</table>

CNN, both empirically as well as qualitatively, then describe specific optimizations for the network architectures used in this study.

4.3.1 Layer-Specific Optimizations

From Equation 4.1, it is apparent that convolutional layers contribute the most towards the overall computational workload. It is essential that these operations are optimized as they form the core of the network, and as such, their use cannot be minimized without affecting network performance. Convolutional layers offer two major design-time options: padding and receptive field size. Each of these options offer a trade-off between power consumption, latency, and accuracy.

The receptive field of a convolutional layer defines the spatial extent to which feature correlations are learned in the layer. For example, in a layer with $5 \times 5$ receptive fields, the correlation among $5 \times 5$ “patches” in the image are learned, and features with correlations across larger “patches” are not adequately captured.

The receptive field is usually a design choice based on the application. However, this choice of receptive field affects the performance of the network in terms of accuracy, latency, and power consumption. To further explore this phenomenon in detail, we empirically analyze the effect of filter size on latency, power consumption, and memory footprint in the PYNQ environment (Table 4.1).
We use a random $28 \times 28$ image from the MNIST [90] dataset and used convolutional layers with different receptive fields.

The first column refers to the filter size and number of filters, while the remaining columns indicate the latency, power consumption, and memory requirements. It can be seen that as the receptive field increases, the power consumption and latency also increase. $11 \times 11$ filters have as much as $3.71 \times$ higher latency per filter compared to the $3 \times 3$ filters. The choice of receptive field size has a less pronounced effect on power consumption, with the difference ranging from 4% ($5 \times 5 \rightarrow 3 \times 3$) to 14% ($11 \times 11 \rightarrow 5 \times 5$) per filter. However, as the input depth increases, larger filter sizes consume more power per filter. Hence the network architecture design i.e., the number of filters per layer plays a significant role in the number of parameters and the number of computations per image to be performed during the inference stage.

The choice of padding in convolutions has both immediate and long-term effects on the inference process. There are two commonly used padding styles in convolutional layers, which we refer to as valid and same. From a network performance perspective, the primary difference is the change in the spatial dimensions of the output. Valid padding (Figure 4.2(a)) performs convolutions at only valid locations, resulting in an output that is smaller than the input. Conversely, the same padding ensures that the output of the layer has the same dimensions as the input, regardless of the input dimension and the kernel size. This is illustrated in Figure 4.2(b). The input is typically padded with zeros to ensure that the output dimension ($10 \times 10$) is the same as the input. It is to be noted that this padding operation is an additional computation step and hence can have effects on both the power and latency, but does not affect the memory footprint since there are no stored or learned parameters.
Figure 4.2: Convolutional operations on an input $10 \times 10$ image given a kernel of size $3 \times 3$ with (a) valid padding or (b) same padding.

In addition to preserving the spatial dimension, same padding has been shown to improve accuracy since the features at the border could be lost too quickly. However, this trades off accuracy with power consumption and latency. In our studies, using the same padding consumes $1.78 \times$ power relative to valid padding, with latency increased by $2 \times$. In constrained environments, repeated convolution operations using the same padding option can increase the power consumption inference latency.
4.3.2 Subsampling Layer

The subsampling layer does not have any learnable parameters and performs pooling operations across the spatial dimension of the input. There are two static parameters to a pooling layer, namely the stride and spatial extent. However, it can consume more power as the number of operations rises linearly with respect to its input dimensions, both spatial as well as depth. This becomes particularly more acute as the kernel size is kept small, since it increases the number of computations to be performed. Given that the most commonly used pooling operations are of size $2 \times 2$ with stride $2$, the number of computations can be very high. For example, given an input of dimension $28 \times 28 \times 32$, which is the input to the first pooling layer in VGG-16, the number of pooling operations is $6,272$. The number of pooling operations increases almost non-linearly with the depth of the network and number of filters in its preceding layer.

Subsampling layers have two modes of operations depending on the spatial dimensions of the input. If the input dimensions are evenly divisible by the pooling kernel, then padding is not required. If not, then padding is needed before subsampling. A more optimized approach (in terms of memory) would involve the conversion of the input matrix into 1-D vectors, performing the pooling and transforming back to the 2-D form.

From our experiments, we observe that the second approach increases the latency by over $2.5 \times$ relative to the first approach while consuming $1.2 \times$ more power. In a large network with multiple subsampling layers, ensuring that the input is divisible by the subsampling kernel size is highly advantageous. By controlling the padding operation in the convolutional layers, it is possible to ensure that the convolved output is the correct size input for the pooling kernel, and thus the first pooling technique can be applied. For example, in our implementation of LeNet on the MNIST
Table 4.2: Impact of dense layer capacity on latency, power consumption, and memory.

<table>
<thead>
<tr>
<th>Layer Capacity</th>
<th>Latency (ms)</th>
<th>Pow. (W)</th>
<th>Mem. (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – 100</td>
<td>0.97</td>
<td>2.10</td>
<td>0.041</td>
</tr>
<tr>
<td>101 – 500</td>
<td>1.60</td>
<td>2.17</td>
<td>0.189</td>
</tr>
<tr>
<td>500 – 1024</td>
<td>91.48</td>
<td>2.25</td>
<td>4.08</td>
</tr>
<tr>
<td>1024 – 2048</td>
<td>138.60</td>
<td>2.35</td>
<td>8.04</td>
</tr>
</tbody>
</table>

dataset, we design the second convolutional layer to have *valid* padding which ensures that the input to the subsequent (subsampling) layer with kernel $2 \times 2$ and stride 2 is evenly divisible. In Section 4.4, we describe such design considerations for each network.

4.3.3 Fully Connected Layer

The fully connected layers, which form the most memory-intensive portion of the CNN, do not contribute significantly to the power consumption, do contribute to network latency and memory requirements. In Table 4.2, we analyze the latency, memory requirement, and power consumption of fully connected layers. We observe that the effect on power consumption as a function of the *layer capacity* is less significant, whereas the contributions to network latency and memory footprint are more pronounced. Thus, using the minimum layer capacity required for the network to learn the nonlinearity in the data for the target inference accuracy can reduce latency and memory requirements, while having a less significant effect on the power consumption.

4.4 Network-Specific Optimizations

In this section, we describe the design choices that helped optimize the architectures for execution in the PYNQ hardware environment.
We designed, optimized, and evaluated four commonly used CNN architectures in increasing levels of complexity: LeNet [94], AlexNet [95], VGG-11 [73], and VGG-16 [73]. Each of these networks follow a similar design paradigm: a combination of convolutional layers followed by subsampling layers, ending with multiple dense layers. In this section, we describe the architecture of each model followed by a discussion of the design choices that helped optimize them for execution in the PYNQ hardware environment.

For LeNet on the MNIST dataset, the second of the convolutional layers utilizes valid padding to ensure that optimal max pooling operations are used. A minimum capacity of 120 and 84 neurons in the two densely connected layers were used as it provided competitive performance to state-of-the-art accuracy while reducing memory footprint. The proposed network was implemented and tested with the CIFAR-10 dataset for comparison and validation.

The design optimizations for AlexNet on the MNIST dataset, the first and second convolutional layers use valid padding to ensure that optimal max pooling operations were used in the subsequent pooling layers. The minimum capacity of 512 and 256 neurons in the two densely connected layers were used as it provided the competitive performance to other published works but allowed us to reduce the memory footprint. The proposed network was also implemented and tested with the CIFAR-10 dataset for comparison and validation.

The design optimizations for VGGNet on the MNIST dataset, two convolutional layers (2 and 3) and three convolutional layers (2, 3, and 10) were set to use valid padding for the VGG-11 model and VGG-16 models, respectively, to ensure that optimal max pooling operations were used in the subsequent pooling layers. A minimum capacity of 1024 and 256 neurons in the two densely connected layers was used as it provided performance comparable to other published works and
allowed us to reduce the memory footprint. Again, the network was implemented and tested with the CIFAR-10 dataset for comparison and validation.

4.5 Experimental Evaluation

In this section, we describe the datasets used, the evaluation platform, and power measurement methodology.

4.5.1 Datasets for Training and Evaluation

We used two publicly available datasets in our experiments, called MNIST [90] and CIFAR-10 [91]. The Modified National Institute of Standards and Technology (MNIST) dataset is comprised of 70,000 $28 \times 28$ handwritten digit (0-9) images. Because the image dimensions are not a power of two, the network architecture must consider the choice of padding for optimal use of computational resources while accounting for latency and power consumption (Section 4.3). For object recognition, the CIFAR-10 dataset is a common benchmark. It is comprised of 10 classes of images totaling 60,000 $32 \times 32$ color images, with each class having approximately 6,000 images.

4.5.2 Evaluation Platform

All Programmable System on Chips (APSoCs) is a class of Zynq devices made by Xilinx which contain a single integrated circuit (IC) with a multi-core processor and an FPGA. The PYNQ-Z1 is an open-source project that allows system designers to use the Python programming language and libraries for embedded system development. The PYNQ-Z1 board contains the ZYNQ XC7Z020-1CLG400C chip, which includes a 650MHz dual-core ARM A9 processor, 512MB DDR3 memory, and an Artix-7 family FPGA with 13,300 logic slices. As seen in Figure 4.3, the discussed
network architectures are implemented on the PYNQ board. Although an Application Specific Integrated Circuit (ASIC) would provide several benefits, including higher performance, lower area, and lower power consumption, its lack of reconfigurability and complexity of the design life cycle make a reconfigurable and programmable platform especially attractive for the rapidly evolving domains of computer vision and machine learning. High performance embedded applications with hardware executed in parallel, video processing at higher frame rate, control with low latency, and signal processing in real time can now be created using PYNQ. In simplifying and developing APSoC design PYNQ can exploit elements like - a high-level programming language (Python) and overlays of FPGA with Python libraries to name a few.

4.5.3 Power Measurement

The experimental setup included a logging computer with an Intel Core i7-7700 processor, 16GB of memory, and Windows 10-64bit OS, as well as a Keithley 2280S-32-6 power supply, a precision measurement tool for recording the instantaneous current drawn by the PYNQ development board.
Figure 4.4: The physical setup for conducting the experiments.
board. The experimental setup is shown in Figure 4.4. The computer was connected to and controlled the behavior of the 2280S via the Keithley KickStart software (v1.9.8). The PYNQ board was powered with a 5V DC supply, and the 2280S sampled the current at 100Hz. This was sufficient for capturing current pulses during both full network evaluation, as well as when measuring instantaneous power from individual layers. We tested networks with both MNIST and CIFAR-10 datasets, using a subset of 100 images from each dataset, which had not previously been used for training or validation. The same sequence of 100 images was used for each trial, and a total of 10 trials were conducted per network, per dataset. The power consumption during each trial was logged for further analysis. Power was computed using the formula $P = V \times I$, where $I$ is the current measured during execution.

Although the *Kill-a-Watt* AC power meter is commonly used in the literature for power measurements of neural network on FPGAs, we found the increased sampling frequency, recording capabilities, and higher precision measurement enabled a more in-depth analysis and understanding of the relationship between the size and function of each network layer, and the corresponding effect on power consumption. Furthermore, powering directly from a 5V supply allows us to emulate powering the board from a battery, as would be the case for mobile or IoT edge applications, rather than lossy AC adapters or commodity desktop power supplies.

### 4.6 Results

This section provides a quantitative analysis of the performance and efficiency of the four CNN architectures implemented with the optimizations described in Section 4.3. We show that
each of the optimizations has a considerable impact on power consumption, memory footprint, and latency.

4.6.1 Throughput Analysis

The throughput of the network is given as billions (Giga) of operations per second or GOPs. Table 4.3 and Table 4.4 show the performance of the networks on the MNIST and CIFAR-10 dataset with respect to inference latency, GOPs, and average power, a measure of energy efficiency, and accuracy. As the number of parameters in the network grows, the optimizations in Section 4.3 become increasingly important. This enables us to fit large networks on resource-constrained platform while minimizing power consumption, and maintaining accuracy. The measured throughput for LeNet for MNIST dataset is 0.42 GOPs, which is competitive with other larger hardware platforms, e.g., at 0.54 GOPs on an Altera Stratix V 5SGSD8 FPGA (48GB DDR3 DRAM) [15] and 0.48 GOPs on a Zynq-7000 XC7Z020 FPGA (1GB DDR3 DRAM) [19].

As seen in Table 4.4 the GOPs for AlexNet for CIFAR-10 dataset is $4 \times$ higher than that for MNIST dataset, despite the increase in input dimensions from $28 \times 28 \times 1$ to $32 \times 32 \times 3$. This is attributed to the fact that the input’s spatial dimensions allow us to design the network such that there are no convolutional layers that require padding. For example, consider AlexNet, whose throughput increased from 0.54 to 2.17. For the CIFAR-10 dataset, the spatial dimension of the input ($32 \times 32$) required *same* padding as opposed to the MNIST implementation, where the first two convolution layers of dimensions $9 \times 9 \times 96$ and $7 \times 7 \times 256$ had *same* padding, thereby reducing the number of floating-point operations performed, by a factor of 10. This allowed for a 301.58% increase in throughput.
Table 4.3: Performance comparison of different architectures with design level optimizations on the MNIST dataset

<table>
<thead>
<tr>
<th>Network</th>
<th>Latency (s)</th>
<th>Throughput (GOPs)</th>
<th>Avg. Power (W)</th>
<th>Efficiency (J × s)</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeNet</td>
<td>$9.10 \times 10^{-3}$</td>
<td>$4.20 \times 10^{-1}$</td>
<td>2.85</td>
<td>2.85</td>
<td>97.06</td>
</tr>
<tr>
<td>AlexNet</td>
<td>$4.06 \times 10^{-1}$</td>
<td>$5.40 \times 10^{-1}$</td>
<td>2.97</td>
<td>3.13</td>
<td>99.16</td>
</tr>
<tr>
<td>VGG-11</td>
<td>$7.60 \times 10^{-2}$</td>
<td>$3.38 \times 10^{2}$</td>
<td>2.85</td>
<td>2.86</td>
<td>99.40</td>
</tr>
<tr>
<td>VGG-16</td>
<td>$5.31 \times 10^{-1}$</td>
<td>$5.79 \times 10^{1}$</td>
<td>3.13</td>
<td>3.41</td>
<td>99.26</td>
</tr>
</tbody>
</table>

Table 4.4: Performance comparison of different architectures with design level optimizations on the CIFAR-10 dataset

<table>
<thead>
<tr>
<th>Network</th>
<th>Latency (s)</th>
<th>Throughput (GOPs)</th>
<th>Avg. Power (W)</th>
<th>Efficiency (J × s)</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeNet</td>
<td>$9.78 \times 10^{-3}$</td>
<td>$3.90 \times 10^{-1}$</td>
<td>2.85</td>
<td>2.85</td>
<td>72.01</td>
</tr>
<tr>
<td>AlexNet</td>
<td>$6.14 \times 10^{-1}$</td>
<td>$2.17$</td>
<td>2.91</td>
<td>3.29</td>
<td>79.45</td>
</tr>
<tr>
<td>VGG-11</td>
<td>$1.46 \times 10^{-1}$</td>
<td>$1.75 \times 10^{2}$</td>
<td>2.93</td>
<td>2.95</td>
<td>81.21</td>
</tr>
<tr>
<td>VGG-16</td>
<td>$4.97 \times 10^{-1}$</td>
<td>$6.17 \times 10^{1}$</td>
<td>2.92</td>
<td>3.17</td>
<td>81.85</td>
</tr>
</tbody>
</table>

The performance with optimizations is also competitive with CPU implementations of Caffe running on an Intel Xeon E5-2680 v3 CPU (12 cores, 2.50GHz) with an NVIDIA Tesla K20X GPU (2,688 CUDA cores). This is indicated in the latency of larger networks such as AlexNet. In [15], the authors report the inference latency for AlexNet as 1029.9 ms on the Stratix V FPGA using F-CNN, and 7552.1 ms on a CPU implementation of Caffe for a batch size of 96. This network consisted of 48 $11 \times 11$ filters, followed by 128 $5 \times 5$ filters, and 640 $3 \times 3$ filters for a total of 945,856 parameters. A comparable implementation, using our optimizations, yields an inference time of 980.2 ms, which is an improvement of 5% over the implementation of F-CNN, and over 6.7× performance improvement compared to the CPU. Implementing the full AlexNet ($96 \ 11 \times 11$ filters followed by $256 \ 5 \times 5$ filters and a total of $2,560 \ 3 \times 3$ filters, for a total of 15.98 million parameters), with the same batch size of 96, results in an inference latency of 7448.7 ms. Note that, in all cases, the number of parameters reported excluding those in the fully connected layers.
Figure 4.5: Comparison of power consumption for four optimized (red) and unoptimized (blue) CNN architectures on MNIST.

4.6.2 Power Consumption

The inference power consumption for each network was analyzed using the measurement set up described in Section 4.5.3. Figure 4.5 visualizes the power consumed for LeNet, AlexNet, VGG-11, and VGG-16 when evaluating on the MNIST dataset, while Figure 4.6 shows the power consumed for the CIFAR-10 dataset. Power results are shown for a random subset of 100 images from each dataset. As the number of operations increases, the power consumption is also expected to increase. However, this is not always observed. For example, VGG-16 has more parameters than AlexNet, but the average power consumption is similar (Figures 4.5 (b) and (d)). The code-level optimizations, along with the design choices, allow us to implement architectures with significantly higher complexity and accuracy while preserving energy efficiency. We also analyze the power consumption of the proposed optimizations using Energy Delay Product (EDP). The energy effi-
An important point to note is that the power consumption across the architectures does not increase dramatically when faced with higher input dimension images. For example, consider the power consumption of the architectures on the MNIST dataset in Figure 4.5 and on the CIFAR-10 dataset 4.6. It can be seen that the average power consumption is similar despite the change in the input dimension - from an input image of dimension $28 \times 28 \times 1$ to $32 \times 32 \times 3$. Design optimizations result in lower power consumption for VGG-16 and AlexNet for CIFAR-10 than on MNIST. This can arguably be attributed to the lack of same padding operations in the convolutional layers on the CIFAR-10 dataset.
Table 4.5: Performance comparison of LeNet on different platforms

<table>
<thead>
<tr>
<th>Approach</th>
<th>Hardware Platform</th>
<th>Latency (ms)</th>
<th>Power (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F-CNN[15]</td>
<td>Stratix V FPGA 48GB DDR3</td>
<td>7.06</td>
<td>27.3</td>
</tr>
<tr>
<td>Caffe[15]</td>
<td>Intel Xeon E5-2680 v3 CPU</td>
<td>48.23</td>
<td>-</td>
</tr>
<tr>
<td>Caffe[15]</td>
<td>NVIDIA Tesla K20X GPU 6GB GDDR5</td>
<td>8.01</td>
<td>235</td>
</tr>
<tr>
<td>fpgaConvNet[19]</td>
<td>Zynq-7000 XC7Z020 FPGA 2GB DDR3</td>
<td>7.91</td>
<td>1.75</td>
</tr>
<tr>
<td>This Work</td>
<td>PYNQ FPGA 512 MB DDR3</td>
<td>9.10</td>
<td>2.85</td>
</tr>
</tbody>
</table>

1 Power consumption is calculated from GOPs and GOPs/W

We also compare our implementation with other hardware-specific implementations specified by Zhao et al. [15] and Venieris et al. [18, 19]. The LeNet architecture with the MNIST dataset is typical to these platforms, so this is used as a baseline (Table 4.5). Here, the heterogeneous platform on PYNQ combined with software and design-time optimizations can compete and/or outperform hardware-specific optimizations on traditional FPGA and embedded platforms. It is interesting to note that the latency does not suffer significantly with the low power consumption to GPUs, which has a power consumption of 278W and 23W compared to CPUs and other FPGA implementations.

4.7 Chapter Summary

In this chapter, we presented a detailed analysis of power, accuracy, and performance of four typical CNN implementation on FPGAs, and compared with implementations on more area/power-intensive platforms. Compared to typical GPU platforms used for CNN implementation and research, a low-power FPGA, paired with highly optimized network architectures, consumed 11× less power and maintaining the state-of-the-art classification accuracy. Compared to more substantial,
multi-core/multi-threaded CPU, the PYNQ FPGA platform with our optimized CNN code consumed 10× less power with approximately 10% performance overhead and no loss in classification accuracy. The analysis of various CNN layers can guide the design and implementation of more optimized networks in the future and can make the local implementation of deep learning classifiers feasible in highly constrained applications.
Chapter 5: Security Considerations in The Internet Of Things

In this chapter, we assess the feasibility of using PUF-based architectures as the core component in the authentication protocol. We first explore existing work on the use of PUFs and the underlying security considerations. We then propose three machine learning based attack models that attempt to clone the PUF architecture under a variety of scenarios such as unencrypted, encrypted, and obfuscated CRPs to simulate real world conditions. We find that the use of unsupervised, generative pretraining helps capture the underlying relationship between the challenge and response, even with the introduction of noise through encryption and obfuscation. Finally, we introduce a machine learning-based defense called a discriminator to help distinguish between the cloned and original PUF to help bolster the authentication process.

5.1 Introduction

The Internet of Things (IoT) ecosystem has grown exponentially with the convergence of various technologies such as deep learning, sensor systems, and advances in computing platforms. The advent of 5G technology and the promise of higher bandwidth is expected to increase the highly connected nature of today’s IoT ecosystem. The massive collection of ubiquitous and pervasive devices in the IoT ecosystem has been deployed across a variety of environments to collect and process massive amounts of data. Applications of IoT devices range from wearable computing

A part of this chapter was published in IEEE Computer Society Annual Symposium on VLSI (ISVLSI) [96]. Permission is included in APPENDIX A.
A part of this chapter was published in IFIP International Internet of Things (IOT) Conference [97]. Permission is included in APPENDIX A.
devices, bio-implantable devices to monitor vital bodily functions for direct human interaction, as well as for “smart” devices that we interact with on a day-to-day basis. With such a highly pervasive nature of “smart” devices, the nature of data being collected and processed can be increasingly private and require safeguards to ensure the integrity and security of the data [1, 2].

With such highly private data, IoT nodes need to be adequately authenticated before collecting and processing such data. The authentication protocol can be as simple as storing the secret key on physical, silicon-based devices and can also be complex cryptography-based protocols. Choosing the authentication protocol has the following set of challenges that must be addressed: (1) IoT devices are typically resource-constrained, thus requiring high energy efficient security protocols, (2) their highly distributed nature can provide easy physical access to the node, and (3) the highly connected nature of IoT framework requires fast and secure security protocols. Traditional approaches to cryptography, while effective, have not proven to be sufficiently lightweight and fast for IoT device authentication. For example, authentication protocols that require storing the secret key on each node device, while an effective strategy, can be bypassed through physical and side-channel attacks on the node device [6] and compromise the integrity of the IoT network and associated data. Recent efforts have shifted to leveraging the inherent randomness induced in silicon devices during the manufacturing process as the secret key, opposed to the traditional binary key stored in silicon devices, which can be susceptible to physical attacks. Such approaches, called Physically Unclonable Functions (PUFs), have helped provide a higher level of security against direct physical attacks. This alleviates the need for costly physical protection measures. PUFs have become increasingly popular and have been used for IoT device authentication [45, 46, 47, 48, 49] and other security tasks [98, 99]. PUFs have been used for both augmenting existing cryptography-
based authentication protocols as well as novel protocols that exploit the lightweight-nature and usability of PUF architectures.

Figure 5.1: A typical IoT architecture with both the enrollment phase and the authentication phase of a PUF-based IoT node authentication scheme.

Silicon-based PUF devices [98] are easily fabricated, physical structures that leverage the stochastic nature of the manufacturing process to create physically unclonable, unique identifiers for each manufactured unit. This typically results in a one-way function. Given an electronic stimulus, the response of a PUF device is an unpredictable, repeatable function. This response identifies each device with a unique signature. This is primarily attributed to the interaction of the external stimulus and the physical structure of the PUF. This interaction is termed as the Challenge-Response Pair (CRP), where the challenge is the external stimulus, and the PUF’s reaction is termed as the response. The unpredictable nature of the PUF, which can be highly sensitive to noise, error
correction circuits [100] are used to reduce the uncertainty in the PUF’s response to make it more reliable. A PUF with sufficiently large challenge-response pairs are called strong PUFs and are typically chosen for most practical security applications.

The use of PUFs as the basis for IoT node authentication has gained momentum in recent times [45, 46, 47, 48, 49]. Using PUFs for IoT security protocols typically involves an initial enrollment phase and an authentication protocol during the actual data exchange. Figure 5.1 illustrates the typical architecture of an IoT network and the generic enrollment protocol. The inner figure shows the enrollment phase and the authentication phase of a PUF-based IoT node authentication scheme. The Pre-Process block represents an optional encryption and/or obfuscation process. A typical IoT network consists of remote, resource-constrained data nodes \((N_1, N_2, N_3, \ldots, N_k)\) connected to static server nodes \((S_1, S_2, S_3, \ldots, S_n)\) that transfer the acquired data to the cloud using routers \((R_1, R_2, R_3, \ldots, R_m)\). The data is transmitted from the routers to the cloud using a network gateway. IoT edge nodes can range from simple sensors to complex systems with a processor, memory, communication, etc. Strong PUFs implemented in complex IoT nodes are subject to attacks, which is the focus of this work. When a data node is added to the IoT network, the enrollment phase is executed to create a CRP database for the PUF within the data node. This database of CRPs is used in the authentication phase when two nodes corresponding to the same server node want to communicate. The shared server node authenticates both data nodes, generates security key pairs, and helps secure key sharing. While practical, the enrollment phase can be used by a malicious attacker to eavesdrop and clone the set of CRPs, which can be used to bypass the PUF-based authentication and compromise the security of the data nodes. There have been advances
that have now been proposed that the extraction of CRPs is then destroyed, i.e., fuse the extraction wires, thereby eradicating the possibility of cloning via this method.

Following the protocols established in [101], extant IoT networks using PUF authentication [45, 46, 47, 48, 49] make the following underlying security assumptions: (1) cloning a PUF architecture, either physically or mathematically is a difficult problem, especially if the underlying architecture is unknown, (2) an adversary has unrestricted physical access to the communication channel, (3) the challenge-response characteristics of the PUF within the data IoT node is an implicit property and is not accessible to an adversary, and (4) the attacker can obtain access to the database of CRPs through malicious software attacks, though explicit knowledge of the secret keys. Given these security assumptions, the goal of the adversary becomes straightforward. In essence, it must be able to spoof the server nodes into accepting a malicious node on behalf of the original data nodes without actual possession of the node in question. Any physical intrusions can compromise the integrity of the PUF and hence render the attack futile. The underlying stochastic nature of PUFs and the above constraints lend itself to a solid security protocol that can be hard to breach. However, advances in machine learning have led to a vast majority of non-invasive attacks on PUF-based security. Machine learning-based approaches can be characterized by the application of a learned mathematical model on a collected subset of valid CRPs. The curation of such data is typically assumed to be an eavesdropping protocol, which is not an unreasonable assumption. Prior works, especially the pioneering work of Rühmair et al. [59], have shown great success in cloning PUFs, gaining cloning accuracy of up to 99.99%. Such success does come with a caveat - the underlying architecture must be known a priori, either through invasive physical intrusions or explicit architecture knowledge.
Today’s IoT nodes are designed such that they are tamper-proof \cite{102,103} which makes it difficult or impossible for micro-probing. Even if the attacker is successful in micro-probing, given the myriad of PUF architectures in literature, extracting information on the underlying PUF architecture is extremely difficult. Hence, earlier ML-based PUF attacks with the assumption of knowing underlying architecture are either not practical or extremely difficult to stage. Additionally, these methods assume that the challenge is available to the attacker in plain-text, i.e., there is no encryption applied to the challenge. Given that most communication through a wireless channel is encrypted, these are very strong assumptions to make, especially in the context of node security in an IoT framework. In this work, we present, for the first time, an ML-based attack that does not require PUF architecture information. We also offer a countermeasure for this attack that can be effectively used to evaluate an IoT node’s trust level remotely.

To overcome such limitations, we focus on an architecture-independent attack, that assumes no prior knowledge of the PUF architecture in the system. We show that observed CRPs are sufficient to improve the cloning accuracy of a strong PUF irrespective of the underlying architecture. The attack can simulate PUF-based data node without knowing underlying PUF architecture. To evaluate the effectiveness of our approach, we compare against a brute force attack model (Section 5.2) that leverages the current advances in PUF-architecture cloning. We leverage architecture-specific cloning \cite{59} through a cascaded framework of (1) PUF architecture identification, (2) employ architecture-specific cloning models, and (3) evaluate the prediction accuracy of the model by combining the architecture classification accuracy and the cloning accuracy in a harmonic mean.
Inspired from the pioneering work of Goodfellow et al. [104] on Generative Adversarial Networks (GANs), we propose a machine learning-based defense, a discriminator, to identify the possibility of cloning using any ML-based attack non-invasive attack. Countermeasures [41, 43] to ML-based cloning have focused on creating complex, cloning resistant PUF architecture. As we enter into a more realizable IoT ecosystem, complex PUF architectures may not be suitable for lightweight IoT systems. Hence, we propose a lightweight, probabilistic identification of cloning through machine learning. To the best of the authors’ knowledge, this is the first such framework for the non-invasive attack of PUF-based IoT network authentication schemes and a proposed mechanism to differentiate original PUFs from cloned ones. In short, this work makes the following novel contributions:

- propose a non-invasive, architecture-independent cloning attack on strong PUFs,
- show that a brute force attack on strong PUFs to identify the PUF architecture for cloning is increasingly complex and hence not trivial for feasible cloning,
- we show that the proposed approach can successfully clone the PUF model even if the challenge-response pair is encrypted or obfuscated, and
- propose a probabilistic, discriminator model to bolster the security of the CRP protocol by identifying possible instances of cloning attacks.

In summary, we present one of the first frameworks to clone PUF-based authentication in an IoT setting, without any physical access to the device and any prior knowledge of the underlying PUF architecture. We also show that the approach can be extended, through unsupervised
noisy, pretraining, to handle two (2) standard encryption protocols and three (3) common PUF architectures, which form some of the more common node authentication setups in practice.

The rest of this chapter is organized as follows. We describe and evaluate a baseline, brute-force approach in Section 5.2, followed by a description of the proposed attack and discriminator approach in Section 5.3. In Section 5.4 we describe the usage of denoising autoencoder for robust feature representation. Section 5.5 introduces the proposed approach for learning a discriminative latent subspace that can be used for machine learning-based cryptanalysis of the security protocols in a typical IoT ecosystem. Section 5.6 presents a mathematical model that is able to discriminate between an original and a cloned PUF called the discriminator model. We then present our empirical evaluation and qualitative evaluation of strong PUFs in the proposed approach with different settings in Section 5.7 and Section 5.8 respectively. Finally, in Section 5.9 we conclude with a discussion on the proposed approach.

5.2 Brute Force Attack on Strong PUFs

In this section, we describe the brute force attack that we formulate for evaluating the effectiveness of the proposed solution. We begin with the motivation for a brute force attack and basic assumptions that are required for applying the cloning models. We then continue with discussion on the mathematical models employed for identifying the underlying PUF architecture and the subsequent results. We conclude with advantages and limitations of the proposed approach.

The success of the proposed models by Rühmair et al. [59] allows us to successfully clone strong PUF models with a prediction accuracy of 99.9%. However, to use it in a non-invasive manner, we would first need to identify the underlying PUF architecture, as the approaches in
5.2.1 Identifying PUF Architectures

Given the set of challenges $\tilde{C}$, we can observe the set of valid responses $R_{c_i}$ for each PUF architecture $c_i \in C_{puf}$, where $C_{puf}$ is the set of all known PUF architectures described. Hence, the
Table 5.1: Brute force attack classification performance and cloning accuracy.

<table>
<thead>
<tr>
<th>PUF Model</th>
<th>PUF Classification Rate (%)</th>
<th>Cloning Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>APUF</td>
<td>81.49%</td>
<td>77.42%</td>
</tr>
<tr>
<td>3 XOR APUF</td>
<td>76.53%</td>
<td>72.71%</td>
</tr>
<tr>
<td>4 XOR APUF</td>
<td>65.01%</td>
<td>61.76%</td>
</tr>
<tr>
<td>5 XOR APUF</td>
<td>63.57%</td>
<td>60.39%</td>
</tr>
<tr>
<td>6 XOR APUF</td>
<td>61.31%</td>
<td>58.25%</td>
</tr>
<tr>
<td>LW 3 XOR APUF</td>
<td>76.91%</td>
<td>73.05%</td>
</tr>
<tr>
<td>LW 4 XOR APUF</td>
<td>65.37%</td>
<td>62.10%</td>
</tr>
<tr>
<td>LW 5 XOR APUF</td>
<td>59.32%</td>
<td>56.33%</td>
</tr>
</tbody>
</table>

The objective of the classification is to learn a function \( f_c \) which maximizes the probability

\[
\arg\max_{\tilde{C}_i \in \mathcal{C}} P(c_i | \tilde{C}_i, R_{c_i}) \tag{5.1}
\]

where the objective is to find the PUF architecture \( c_i \) given the challenge \( \tilde{C}_i \), and the subsequent response \( R_{c_i} \). We use the following machine learning models as the basis for the function \( f_c(\cdot) \): logistic regression, artificial neural network, and random forests.

5.2.2 Empirical Evaluation

We evaluate the performance of the proposed brute force attack to identify the architecture of eight (8) common strong PUF architectures. We use a fixed number of randomly sampled 100 CRPs for evaluation for each PUF architecture for a total of 800 CRPs. We report average results from 5 different runs, with the test set sampled each time randomly. We curate a collection of 100,000 CRPs for training the classification model.
As can be seen from Table 5.1, identifying the PUF architecture from an observed set of CRPs is not a trivial task. Even with 100% cloning accuracy for a given PUF architecture, identifying the said architecture requires a large set of CRPs for training a model. The maximum performance that we were able to obtain was using the logistic regression model, which took 100 iterations to converge, resulting in the maximum classification rate for Arbiter PUF architecture. There was a large confusion among different design variations of each PUF type. The prediction rate for XOR PUFs decreased as the complexity of the architecture increased.

It can be seen that identifying the PUF architecture requires significant training resources - 100,000 CRPs while recognizing the arbiter PUF with an average accuracy of 81.49%. The classifier performed worst on the lightweight PUFs, yielding a maximum identification accuracy for the 3 bit XOR lightweight PUF. The identification rate also affected the cloning prediction rate of the brute force approach, as each misclassified PUF architecture affecting the cloning quality. While the average cloning accuracy can be as high as 77.42% (for the Arbiter PUF), the numbers can be misleading in practice. The performance of the two-stage attack model is rather low, considering the possible gap between the intra-Hamming and inter-Hamming distances of PUF CRPs, this prediction rate cannot be considered to be successful cloning of the PUF architecture.

5.3 Architecture Independent PUF Modeling

In this section, we describe our proposed approach for a PUF-independent attack model on various PUF architectures by exploiting the CRP authentication protocol. We begin with a discussion on the use of machine learning models to capture the underlying correlation between challenge-response pairs to model the randomness unique to a given PUF architecture. We then
introduce a noisy autoencoder-based pretraining of the neural network model for handling noise and obfuscation-based techniques for more robust feature learning. We then follow with a discussion on defending against such attacks using complementary machine learning models.

5.3.1 Attack Model

Each PUF is made unique through a digital signature characterized by its response to a given challenge. This signature is representative of the randomness encoded in its state due to manufacturing variations and other physical disorders. To compromise the integrity of the CRP protocol, one has to model this randomness to generate a response representative of the PUF’s signature. There are two approaches to this problem: a model-based solution and a model-agnostic solution. The model-based solution, explored in [59], attempted to capture this randomness through modeling the characteristics of a PUF using domain knowledge (PUF architecture) and characteristics (delay model, thermal response characteristics, etc.). Thus, the attack consists of a regression of the model’s parameters.

We, however, consider an architecture-independent approach to the solution by disregarding the need for a characteristic equation for the PUF. We postulate that the challenge and subsequent response of any given PUF is representative of its characteristic function. Thus, modeling the dependency between the various features of a given challenge along with the target response allows us to capture the randomness of a given PUF architecture. To this end, we use several approaches to capture the dependency between the challenge and response pairs of various PUF architectures. Since the underlying dependency is not known to be linear or non-linear, we explore several different machine learning models that characterize the dependence with a linear decision boundary (logistic regression) or with a non-linear decision boundary (random forest and artificial neural networks).
The attack model consists of learning the optimal function that maps the given $n$-bit challenge $C = c_1, c_2, \ldots, c_n$ to an appropriate output response $R \in \{-1, 1\}$ with a probability $p(R|C)$. The objective of the attack model is to learn the function $f : C \rightarrow R$ such that the difference between the generated and actual response of the PUF is minimized. Hence the best attack model is characterized by the search for the optimal function $f$ given by

$$\arg\min_{(C_s, R_s)} E[(\hat{f}(C) - f(C))^2]$$

where $\hat{f}(C)$ is the characteristic function of the given PUF architecture and $(C_s, R_s)$ represents the space of all known challenge-response pairs obtained through the eavesdropping protocol. We search for the optimal function $f(C)$ through the characteristic equation of the different machine learning models defined above. For example, in a logistic regression model, $f$ is defined as

$$f = \arg\max(\sigma(R \times d(\vec{w}, C)))$$

where $\vec{w}$ is a learned vector that represents the decision boundary ($d$) for the logistic regression model and $\sigma$ is the logistic function.

### 5.4 Denoising Autoencoders for Robust Feature Learning

While the attack model presented in Section 5.3.1 can handle clear-text challenges, the encryption protocols such as AES and DES can inject noise into the relationship between the challenge and the response, hence obscuring the characteristic function of the PUF architecture. To account for this, one must either (1) break the encryption through traditional cryptanalysis or (2) learn robust representations that can decouple the noise from essential information within the input.
challenge. Since the computational resource for pursuing the former can be expensive, we take the latter approach and attempt to learn robust representations through unsupervised pretraining using a denoising autoencoder. In this approach, we train a neural network (multilayer perceptron, MLP) as our attack model.

A traditional autoencoder is an unsupervised neural network, whose objective is to learn a compressed representation of the input data through a cascaded encoding-decoding operation. The network architecture comprises of two neural networks, an encoder network, and a decoder network, working together to learn an encoded representation or latent space. The role of the encoder is to compress the input data into a lower-dimensional representation that captures the underlying pattern of the data by learning to ignore as much of the spurious patterns or noise as possible. This compressed representation represents the bottleneck layer of the network. The role of the decoder is to learn to reconstruct the original input from this compressed representation. This process is represented in Figure 5.3(a), where it can be seen that the latent space has a lower dimensionality compared to the larger dimensional input and output. The input and output of the autoencoder framework have the same dimensions. The training objective for an autoencoder network is to minimize the reconstruction loss, which is typically an $L_2$ loss or binary cross-entropy.

While autoencoders learn useful features (the latent space) that can be used for downstream classification tasks, the addition of noise or perturbations in the input can drastically change the representations unless added during training. To account for noise injected through encryption, we train the autoencoder as a denoising autoencoder. The idea is to train the autoencoder to reconstruct the input from a corrupted or randomly perturbed version of the input. This training strategy is applied to force the hidden layer to discover more robust features and prevent it from merely learning
Figure 5.3: Illustration of (a) an autoencoder, (b) a variational autoencoder, and (c) the proposed multi-headed variational autoencoder.
the identity function. We construct the denoising autoencoder by adding a stochastic corruption step to the input. While the input can be perturbed in many ways, we want our representations to handle the inherent noise applied due to the wireless channel, obfuscation, and encryption. Hence, in our implementation, we apply the following perturbations: (1) randomly mask part of the input by making them zero, (2) add random white noise to the input, and (3) add a hashing function to the CRP to simulate the encryption techniques. At every training iteration, one of the above perturbations is applied to the input, and the output of the decoder network is compared to the original input.

5.4.1 Implementation and Training Details

Due to the complex nature of the proposed network, we present the implementation details for understanding. The encoder network is a four-layer network of fully connected layers. Between each subsequent layer is a dropout layer [105], which helps prevent overfitting. Each dropout layer has a dropout probability of 50%. The number of neurons in each layer is reduced by $0.5 \times$ to reduce the dimensionality of the processed data. This follows the standard protocol in autoencoders to induce the bottleneck at the end of the encoding network. The decoding network is a mirror of the encoding network, with the number of neurons increasing to match the output dimensions. We train the network for ten epochs at a learning rate or $1e^{-4}$ using the standard Gradient Descent optimizer.

5.5 Variational Autoencoders for Latent Space Modeling

In this section, we introduce the proposed approach for learning a discriminative latent subspace that can be used for machine learning-based cryptanalysis of the security protocols in a
typical IoT ecosystem. We begin with a brief introduction to variational autoencoders, which form the backbone of the proposed approach. We then introduce the proposed approach with a multi-headed decoder, which helps learn a more robust subspace for better modeling of the encryption protocols. Finally, we expand on the strategy employed in the optimization process for end-to-end training of the proposed network.

5.5.1 Variational Autoencoders

Encryption techniques such as AES and DES, to name a few, secure the transmitted data by injecting noise into the data through various techniques including, but not limited to hashing and block cipher. By doing so, the actual data within the transmitted information is hidden from prying influences. Hence, any attempt to break the security of the encryption must either (1) know the encryption techniques and the hidden cipher to recover the original data, or (2) model the underlying data distribution effectively to learn a model for manipulating the information stream. While there have been existing work in crypt-analysis for the former approach, the latter has not been explored extensively. Modeling the internal structure of the data distribution offers three significant advantages: (1) knowing the underlying distribution allows us to reduce the dimensionality of the data by ignoring the noise in the transmission, (2) allows for the possibility of learning a generative model that clones the source of the data distribution, which in our case is the PUF within the IoT data node, and (3) learning a generative model allows the attacker to probe the PUF with genuine, or rather, valid challenges to further extract the PUF characteristics. To achieve the above, we employ the use of an unsupervised neural network called autoencoders, or more specifically, variational autoencoders.
While autoencoders are incredibly useful in learning a compressed representation of a (potentially) noisy input data, there is no way to restrict, or rather, predict the latent space representation of a given input in a deterministic manner. This poses two critical concerns. First, while very useful for compression, the latent space learned in a traditional autoencoder is scattered. This leads to better reconstructions of the input image but is not conducive to generate new samples that match the valid distribution. Second, a deterministic latent space allows for better probing of the PUF model through generating legitimate challenges. It also allows us to model the PUF characteristics in a model agnostic manner. To overcome these limitations, we employ the use of a variational autoencoder. A modification on the traditional autoencoder network paradigm, a variational autoencoder aims to restrict the latent space into a more deterministic manner by introducing an additional optimization constraint. Figure 5.3(b) illustrates the typical architecture of a variational autoencoder. As can be seen, the bottleneck layer is not passed through to the decoder network directly. Rather, it is used to generate a normal distribution $N(\mu, \sigma)$ (i.e., mean $\mu$ and standard deviation $\sigma$). The latent space is then sampled from this distribution to ensure that the bottleneck layer follows a given set of distribution and hence is deterministic. The training objective then becomes the reconstruction loss and the KL divergence loss to ensure that the distribution follows the standard normal distribution $N(0, 1)$. This additional loss ensures that the parameters $\mu$ and $\sigma$ do not regress such that the latent space of the encoder network is preserved. The objective function is given by

$$\mathcal{L}(\theta, \phi, X) = E_{Z \sim q_{\phi}(Z|X)}(\log P_{\theta}(X|Z)) - D_{KL}(q_{\phi}(Z|X)||P_{\theta}(Z))$$  \hspace{1cm} (5.4)
where $X$ is the input to be modelled (the encrypted challenge in our case), $Z$ is the hidden variables (the latent space) from which to generate new challenges, $P_\theta(X|Z)$ is the generative process done by the decoder and $q_\phi(Z|X)$ represents the encoding process. $\theta$ and $\phi$ represent the parameters of the decoding and encoding processes, respectively.

5.5.2 Multi-headed Decoding for Robust Latent Subspace Modeling

The use of a variational autoencoder helps in providing a deterministic latent space by forcing the encoder representations to follow a normal distribution. Given that the only task of the encoder is to learn representations that can be reconstructed, there can be a tendency to overfit to the sample distribution due to the single-task learning paradigm. To overcome this inhibition, we propose the use of a multi-headed decoder network to introduce a form of multi-task learning. This provides a form of inductive transfer and allows us to form better representations for modeling the PUF characteristics. In addition to the traditional reconstruction head, we introduce a second decoder which acts as a brute-force decrypting mechanism. We assume that a minimal amount of CRPs is available to the attacker in both plain-text and encrypted forms. Given the multitude of possible eavesdropping mechanisms, this is not an unreasonable assumption. The proposed architecture is shown in Figure 5.3(c), where it can be seen that a joint representation, learning by the encoder, is used as the latent space for both reconstructing the original challenge as well as the decrypted challenge. This allows the model to learn a latent space representation that captures the inherent structure of a valid CRP while learning to ignore the noise induced by the encryption protocols. In Section 5.7, we can see that the use of the second decoder network as a brute-force decryption method offers better modeling of the underlying PUF architecture.
Formally, the objective of the proposed network differs from the traditional variational autoencoder (Equation 5.4). First, there is another generative process to uncover the plain-text challenge represented by $d_{\psi}(\tilde{X}|Z)$, where $\tilde{X}$ represents the plain-text challenge. Second, the generation of the decrypted challenge must also be dependent on the encoded representation $Z$. This results in the updated objective function given by

$$
L(\theta, \phi, \psi, X, \tilde{X}) = E_{z \sim q_{\phi}(Z|X)}(\log P_{\theta}(X|Z) + \log P_{\theta}(\tilde{X}|Z)) - D_{KL}(q_{\phi}(Z|X)||P_{\theta}(Z))
$$

where $\tilde{X}$ is the clear text challenge, $X$ is the input to be modelled (the encrypted challenge in our case), $Z$ is the hidden variables (the latent space) from which to generate new challenges, $P_{\theta}(X|Z)$ is the auto-generative process done by the first decoder, $d_{\theta}(\tilde{X}|Z)$ is the decrypted generative process done by the second decoder and $q_{\phi}(Z|X)$ represents the encoding process. $\theta$, $\psi$ and $\phi$ represent the parameters of the two decoding processes and the lone encoding process, respectively.

The addition of the second decoder network introduces the notion of multi-task learning (MTT). The use of multi-task learning is crucial in many aspects, especially considering that the number of CRPs available are often very low, ranging from the low hundreds to a thousand. Since the encoder network is shared among the two decoders, this reduces the possibility of the network to overfit to the training set of the CRPs and helps generalize to unknown CRPs. In addition to preventing overfitting, the hard parameter sharing paradigm offers other benefits such as attention focusing, implicit data augmentation, reducing representation bias, and regularization, to name a few.
5.5.3 Implementation Details and Training Strategy

Since the proposed architecture has a complex structure, we detail the implementation
details and the training strategy for the approach here. The encoder consists of four (4) densely
connected layers, with each layer interspersed with a dropout layer. Each dropout layer has a
dropout probability of 50%. We reduce the dimensionality of the input by $0.5 \times$ at each fully
connected (dense) layer. This follows the standard protocol in autoencoders to induce the bottleneck
at the end of the encoding network. Each of the two decoders (reconstruction and decryption) consist
of two fully connected layers that increase the dimensionality back to the original dimension and
decrypted challenge dimensions, respectively. We also have a series of two (2) fully connected layers
that take the latent space as input and produces the PUF response as output. This is the only part
of the network that is trained in a supervised manner, i.e., using labels and target dimensions. The
encoder and two decoders are trained in an unsupervised manner.

Since the training data is limited, most neural networks tend to overfit to the smaller amounts
of data and do not generalize well to the other, unobserved challenge-response pairs. To overcome
this, we propose the following training regimen. For ten epochs, we first train the network end-to-
end only with the reconstruction decoder as active i.e., it is trained first as a traditional variational
autoencoder. For the next ten epochs, we then train the decryption decoder for ten epochs while
freezing the weights of the reconstruction decoder. This represents the unsupervised training portion
of the proposed training regimen. We then begin the supervised training process. In this part of
the training, we freeze the layers of the decoding structures and take the latent space produced by
the encoder network and feed it to a series of fully connected layers and model the PUF response
to the input challenge. The neural network’s target is the PUF response. We train for a total of 100 epochs, with the unsupervised and supervised portions interspersed together.

5.6 Machine Learning-based Countermeasure

The modeling of the internal randomness of a given PUF architecture puts the integrity of the CRP-based authentication into question. Hence, it becomes critical that we are able to differentiate between the original PUF and an adversarial attack, such as the ones described in Section 5.2 and Section 5.3. To this end, we introduce a mathematical model that is able to discriminate between an original and a cloned PUF called the discriminator model, as illustrated in Figure 5.4. The discriminator decides whether each instance of the response belongs to the actual PUF or a malicious attacker. As seen in Figure 5.4, the discriminator model takes in the response of the original PUF along with the response of the PUF cloned with several ML attacks as the input to predict whether the PUF is an original or a cloned and returns the probabilities. The cloned part of the response is shown in red. The output of this discriminator is a single scalar value $D(C)$, indicative of an adversarial attack. The value $D(C)$ is a probability function that maps a given response ($R$) to the distribution belonging to either the original PUF ($\hat{f}(C)$) or an attacker ($f(C)$) for a given $n$-bit challenge $C$. Hence, the optimal discriminator model is given by

$$D^*(C, R) = \frac{p(\hat{f}(C))}{p(f(C)) + p(\hat{f}(C))}$$

(5.6)

where $D^*(C, R)$ is a mathematical model that maps the response $R$ for a given challenge ($C$) into the probability space of either the original PUF ($\hat{f}(.)$) or the attack model ($f(.)$). We report the performance of the discriminator model on different PUF architectures in Table 5.2. Again, we ex-
Figure 5.4: ML-based discriminator model to ascertain a PUF’s integrity

explore the use of well-known machine learning models as the basis for our discriminator mathematical model.

Table 5.2: Discriminator performance for different PUF architectures.

<table>
<thead>
<tr>
<th>PUF Model</th>
<th>Discriminator Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>APUF</td>
<td>94.43</td>
</tr>
<tr>
<td>3 XOR APUF</td>
<td>98.81</td>
</tr>
<tr>
<td>4 XOR APUF</td>
<td>95.99</td>
</tr>
<tr>
<td>5 XOR APUF</td>
<td>96.15</td>
</tr>
<tr>
<td>6 XOR APUF</td>
<td>100</td>
</tr>
<tr>
<td>LW 3 XOR APUF</td>
<td>91.66</td>
</tr>
<tr>
<td>LW 4 XOR APUF</td>
<td>94.78</td>
</tr>
<tr>
<td>LW 5 XOR APUF</td>
<td>96.31</td>
</tr>
<tr>
<td>Average</td>
<td>96.01</td>
</tr>
</tbody>
</table>

The search space for the optimal discriminator is similarly characterized by the optimization function defined in Equation 5.2. However, the search is represented by the discriminator to distinguish between the original PUF’s response and a cloning attack.
5.6.1 Search for Optimal Attack-Discriminator Model

The search space for the optimal attack model and discriminator model is defined by the optimizer functions defined in Equation 5.2 and its subsequent adaptation for the discriminator, respectively. We employ a simple grid search algorithm to find the optimal attack model \( f(.) \) from a given set of possible models \( F \). The attack models space, \( F \) comprises of all transformation functions that satisfy the condition \( f : C \rightarrow R \). We restrict the search space to the given three machine learning models: Logistic Regression (LR), Random Forest (RF), and Neural Network (NN). We also ensure that the optimal discriminator is chosen from a set of discriminative function \( G(.) \in G_s \), where \( G_s \) is the collection of all discriminative functions that optimize the probability function defined in Equation 5.6. Again, we restrict the search space to the three aforementioned models. While the grid search suffers from the curse of dimensionality and does not scale to large search spaces of \( F \) and \( G_s \), limiting the number of plausible functions allows us to exhaustively search for the optimal discriminator for a given attack model and a target PUF. Additionally, the grid search is a reasonable approach given that it can be embarrassingly parallel.

5.7 Evaluation and Analysis

In this section, we quantitatively evaluate and analyze the performance of the three machine learning-based models proposed in Section 5.2 and Section 5.3. We begin with a discussion on the experimental setup and metrics. We then evaluate the proposed approaches in three different settings: (1) unencrypted authentication protocol, (2) encrypted authentication protocol, and (3) authentication using obfuscated challenges. We conclude with an evaluation of the machine learning-based countermeasure, proposed in Section 5.6, for each of the proposed approaches.
5.7.1 Experimental Setup

We follow the same experimental setup by [59] and report the upper bound of the attacker’s ability to successfully clone a given PUF architecture as its accuracy in a supervised setting. We report all results as the average of ten (10) experimental runs. For evaluating under the unencrypted setting, we consider three strong PUF architectures (Arbiter, XOR, and Lightweight), while each of them contains three stages (64, 128, and 256) and the number of XOR is limited to (3, 4, and 5) for both XOR and Lightweight PUFs. This gives us a total of 24 different strong PUF architectures for validating the efficacy of the proposed cloning models. For evaluating under the encrypted setting, we consider two (2) conventional encryption techniques - the Data Encryption Standard (DES) and the Advanced Encryption Standard (AES). We use the 128-bit versions of both encryption methods. We consider two strong PUF architectures in a 64-stage Arbiter PUF and XOR PUFs as well as two (2) variations of the XOR PUF - 3-XOR and 4-XOR PUFs to evaluate the ability of the proposed approach to generalize to more complex architectures. We present the average results of the experiments conducted on a limited CRP regime of less than 250 CRP pairs for both training and testing. Although DES is susceptible to crypt-analysis, it is a non-trivial task. 128-bit AES is resistant to brute force attacks, given that there can exist as much as $3.4 \times 10^{38}$ key combinations. Such characteristics make the task of cloning an encrypted PUF a challenging problem. For evaluating under the obfuscated challenge setting, we use a simplified version of the OB-PUF proposed in [42]. We use the arbiter 64-stage PUF as the base PUF architecture. We randomly perturb the $n\%$ of the challenge and evaluate the ability of the cloning model to reconstruct and generate the cloned response.
5.7.2 Unencrypted PUF-based Authentication

We evaluate the ability of the proposed approaches in the unencrypted PUF-based authentication setting. This is the commonly used setting in machine learning-based cloning attacks, such as [59] on PUF architectures. We summarize the cloning results in Table 5.3, from the optimization process described in Section 5.2. Results for each machine learning model can be seen in Figure 5.6. Each bar represents the average accuracy the PUF architectures. Along X-axis, X(Y) refers to machine learning model X is used for tasks Y - cloning model (CM) or discriminator model (DM). Our approach does not require physical access or prior knowledge on the PUF architecture. The average cloning accuracy of our approach can be as high as 93.50%. It is to be noted that while [59] achieve cloning accuracy of 99.9%, they do require that the underlying architecture is known, and physical access is available. The cloning accuracy of the proposed model drops as the complexity of the PUF architecture grows, with the lightweight 5-XOR Arbiter PUF being the hardest to clone. This could arguably be attributed to the randomness introduced by the complex PUF architectures. From Table 5.3, we can see that, on average, a strong PUF can be cloned with a cloning error of 10.83% irrespective of its underlying architecture of the PUF. The aging of the PUF [106] affects the delay characteristic, which produces a different pattern of the responses compared to the compromised node. It can be seen that the cloning time is reasonable, particularly given the complexity and stochastic nature of the considered PUFs.

5.7.3 Encrypted PUF-based Authentication

In this setting, we evaluate the cloning ability of machine learning models when the challenge is encrypted, which is the standard practice in most practical IoT systems. We summarize the
Table 5.3: Cloning error and time for different PUF models

<table>
<thead>
<tr>
<th>PUF Model</th>
<th>Cloning Error (%)</th>
<th>Cloning Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>APUF</td>
<td>6.50%</td>
<td>0.00001 min</td>
</tr>
<tr>
<td>3 XOR APUF</td>
<td>8.20%</td>
<td>1.18083 min</td>
</tr>
<tr>
<td>4 XOR APUF</td>
<td>10.70%</td>
<td>1.63333 min</td>
</tr>
<tr>
<td>5 XOR APUF</td>
<td>9.00%</td>
<td>62.8010 min</td>
</tr>
<tr>
<td>6 XOR APUF</td>
<td>10.70%</td>
<td>240.040 min</td>
</tr>
<tr>
<td>LW 3 XOR APUF</td>
<td>12.00%</td>
<td>0.02650 min</td>
</tr>
<tr>
<td>LW 4 XOR APUF</td>
<td>12.50%</td>
<td>30.9667 min</td>
</tr>
<tr>
<td>LW 5 XOR APUF</td>
<td>17.00%</td>
<td>180.025 min</td>
</tr>
<tr>
<td>Average</td>
<td>10.83%</td>
<td>64.5759 min</td>
</tr>
</tbody>
</table>

cloning rates and times for different PUF architectures under different encryption protocols in Table 5.4. We observe that adding the encryption protocols cause significant problems to standard machine learning-based cloning models.

5.7.4 Obfuscated PUF-based Authentication

We also evaluate the performance of the cloning models when the challenge is obfuscated, as postulated in obfuscated PUF architectures such as [41, 42]. We consider a simpler version of these approaches for our experiments. We use the 64-stage Arbiter PUF as the base PUF model. We randomly perturb or obfuscate the plain text challenge to an arbitrary constant. This results in an obfuscated challenge, which is then presented to the cloning model to generate a response. We present results in Figure 5.5. It can be seen that traditional machine learning-based cloning models such as logistic regression (LR), random forests (RF), and neural networks (MLP) are drastically affected by increasing amounts of obfuscation. The autoencoder models, on the other hand, can maintain their performance to reasonable levels, with the denoising autoencoder performing a little better at higher obfuscation levels.
Table 5.4: Cloning Rates and times for different PUF architectures under different encryption protocols

<table>
<thead>
<tr>
<th>Approach</th>
<th>PUF Model</th>
<th>DES</th>
<th>AES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Acc.</td>
<td>Time (s)</td>
</tr>
<tr>
<td>LR</td>
<td>64-Stage Arbiter</td>
<td>46.9</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td>3-XOR</td>
<td>60.9</td>
<td>26.2</td>
</tr>
<tr>
<td></td>
<td>4-XOR</td>
<td>43.8</td>
<td>53.9</td>
</tr>
<tr>
<td>RF</td>
<td>64-Stage Arbiter</td>
<td>51.6</td>
<td>0.001</td>
</tr>
<tr>
<td></td>
<td>3-XOR</td>
<td>59.4</td>
<td>0.31</td>
</tr>
<tr>
<td></td>
<td>4-XOR</td>
<td>42.2</td>
<td>1.8</td>
</tr>
<tr>
<td>MLP</td>
<td>64-Stage Arbiter</td>
<td>56.1</td>
<td>35.8</td>
</tr>
<tr>
<td></td>
<td>3-XOR</td>
<td>51.1</td>
<td>70.8</td>
</tr>
<tr>
<td></td>
<td>4-XOR</td>
<td>50.1</td>
<td>98.7</td>
</tr>
<tr>
<td>Standard AE</td>
<td>64-Stage Arbiter</td>
<td>58.7</td>
<td>43.1</td>
</tr>
<tr>
<td></td>
<td>3-XOR</td>
<td>54.5</td>
<td>44.0</td>
</tr>
<tr>
<td></td>
<td>4-XOR</td>
<td>51.9</td>
<td>46.7</td>
</tr>
<tr>
<td>Denoising AE</td>
<td>64-Stage Arbiter</td>
<td>65.6</td>
<td>45.6</td>
</tr>
<tr>
<td></td>
<td>3-XOR</td>
<td>58.1</td>
<td>43.6</td>
</tr>
<tr>
<td></td>
<td>4-XOR</td>
<td>57.3</td>
<td>42.6</td>
</tr>
<tr>
<td>Standard VAE</td>
<td>64-Stage Arbiter</td>
<td>69.4</td>
<td>84.7</td>
</tr>
<tr>
<td></td>
<td>3-XOR</td>
<td>61.5</td>
<td>87.0</td>
</tr>
<tr>
<td></td>
<td>4-XOR</td>
<td>55.5</td>
<td>86.7</td>
</tr>
<tr>
<td>Our VAE</td>
<td>64-Stage Arbiter</td>
<td>75.6</td>
<td>98.6</td>
</tr>
<tr>
<td></td>
<td>3-XOR</td>
<td>64.8</td>
<td>83.6</td>
</tr>
<tr>
<td></td>
<td>4-XOR</td>
<td>60.3</td>
<td>82.6</td>
</tr>
</tbody>
</table>
Figure 5.5: Effect of challenge obfuscation on cloning performance. Accuracy is shown in comparison with varying amounts of challenge obfuscation.

It should be noted that the performance is tested only on a limited evaluation set of 200 CRPs. More complicated obfuscation techniques such as those proposed in [41, 42] and less training would further degrade the performance of machine learning-based cloning attacks.

5.7.5 Discriminator Performance

Given the competitive performance of the machine learning models for cloning PUF architectures under different conditions, it becomes imperative that we are able to distinguish between a cloned PUF and the original PUF. We evaluate the ability of the proposed discriminator model (Section 5.6) to identify a cloned PUF. We present the results in Figure 5.6. It can be seen that it is possible to identify a cloned PUF with a high degree of confidence from its response to the presented challenge. We are able to identify cloned PUFs with up to 95% accuracy (Figures 5.6 (a, e, and h))
for some PUF architectures such as lightweight XOR PUFs and Arbiter PUFs. Other architectures such as 4-XOR and 5-XOR PUFs are harder to clone and harder to discriminate between cloned and original PUFs.

5.8 Qualitative Evaluation of Strong PUFs

Following experimental setup by [59], we report the upper bound of the attacker’s ability to successfully clone a given PUF architecture as its accuracy in a supervised setting. To evaluate the ability of the proposed approach to cloning a given PUF successfully, we consider two strong PUF architectures in a 64-stage Arbiter PUF and XOR PUFs. We consider two (2) variations of the XOR PUF - 3-XOR and 4-XOR PUFs to evaluate the ability of the proposed approach to generalize to more complex architectures. We also consider two (2) conventional encryption techniques - the Data Encryption Standard (DES) and the Advanced Encryption Standard (AES). We use the 128-bit versions of both encryption methods. This gives us a total of six (6) different strong PUF architectures for validating the efficacy of the proposed method. We present the average results of the experiments conducted over ten (10) trials and on a limited CRP regime of less than 250 CRP pairs for both training and testing. Although DES is susceptible to crypt-analysis, it is a non-trivial task. 128-bit AES is resistant to brute force attacks, given that there can exist as much as $3.4 \times 10^{38}$ key combinations. Such characteristics make the task of cloning an encrypted PUF a challenging problem.

*Arbiter PUFs* are often considered by many to be strongly predictable and hence more susceptible to machine learning-based attacks. However, with the added security of an encryption protocol, the predictability of an arbiter PUF model can be considered to lower significantly. We
Figure 5.6: Comparison of cloning and discriminator accuracy for cloning models under different PUF architectures.
can corroborate this in our experiments with a 64-stage arbiter PUF. We present these results in Table 5.4. It can be seen that the brute force attacks do not perform well on this task, although some, such as logistic regression, have shown up to 99.9% accuracy in cases when the challenge is not encrypted. Additionally, the addition of even a relatively weak encryption scheme such as 128-bit DES significantly degrades the performance of machine learning models. On the other hand, our proposed approach can clone the Arbiter PUF model with significantly higher accuracy. There is a significant difference in performance between the proposed approach and the brute force models, even considering the similarly structured MLP approach, which differs from the proposed approach only in that the unsupervised training regime is not conducted on it during the training phase.

**XOR PUFs** offer a significantly higher challenge to the cloning problem compare to the arbiter PUFs. As the number of stages grows, the predictability of the PUF architecture reduces. This makes the XOR PUF more suitable for nodes requiring additional security. The addition of encryption protocols such as DES and AES makes it even more challenging to clone a given PUF architecture. From Table 5.4, we can see that as the number of stages increases, the ability of the machine learning models to clone the PUF device reduces drastically. It is important to note that in the literature [59, 99], the maximum number of XORs used is 6. We experiment up to 4 XOR PUFs in this work. We also find that in XOR PUFs, the role of the decryption head is significantly higher than in arbiter PUFs. This could arguably be attributed to the fact that each of the XOR nodes in the PUF architecture adds to the non-linearity of the PUF characteristics, thereby reducing its predictability and hence providing added security against machine learning attacks.
5.8.1 Ablative Studies

We also perform ablation studies to evaluate the impact of each of the components that are part of the proposed framework: (1) decryption decoder head, (2) the reconstruction decoder head, and (3) the use of variational autoencoders for unsupervised pre-training of the encoder network. It can be seen that each decoder head adds significant improvements over the base model. The performance improvement due to the addition of the decryption decoder can be as high as 5.7%. Additionally, the mere use of neural networks is not sufficient to guarantee successful cloning of a PUF architecture, especially with the employment of encryption schemes. We can see that the use of the objective functions described in Equation 5.4 and Equation 5.5 and the unsupervised pre-training regimen add significant performance gains over the vanilla neural networks (MLP). We observe as much as 20.6% improvement in cloning accuracy for arbiter PUFs.

5.9 Chapter Summary

In this chapter, we presented and evaluated three different machine learning approaches to attack PUF-based edge node authentication through cloning the underlying PUF model. We are, to the best of our knowledge, the first to address the problem of encrypted and obfuscated CRPs. We showed that a priori knowledge and physical access to the PUF architecture is not necessary to clone the PUF model. Additionally, the use of autoencoder-based pre-training allowed us to handle additional challenges such as encryption and simple obfuscation. We showed that machine learning models could be powerful enough to clone PUF models in different settings successfully. We also introduce a novel discriminator model to identify cloned and original PUFs with a high degree of confidence. Extensive experiments show that the proposed approach can generalize even
with a limited number of CRPs and can show significantly higher cloning accuracy compared to brute force machine learning models. In the future, we aim to show that the proposed approach can recover CRPs that are transmitted with complex obfuscation techniques and handle noise induced through channels and aging of PUF devices.
Chapter 6: Conclusion and Future Work

The Internet of Things and, more recently, the Internet of Medical Things (IoMT) have seen tremendous progress, transforming various industries such as healthcare, entertainment and transportation. These advances have had an incredible impact on the workforce by increasing productivity and improving the quality of life. Despite such advances, there appears to be some inhibitions in scaling to millions of users such as security, privacy and immersive, real-time experience.

In this dissertation, we successfully proposed the use of machine learning as the disruptive mechanism for enabling the development of real-time, immersive algorithms for improving the quality of life while ensuring the privacy and security of the user and data. We postulate the use of unsupervised, computation-aware algorithms can give rise to decentralized learning systems which can provide a more secure information processing framework. We also find that machine learning can be used to find vulnerabilities in existing IoT node authentication and bolster them through probabilistic screening of compromised IoT data nodes.

Specifically, in Chapter 3, we introduced an unsupervised sleep state segmentation algorithm that uses a self-supervised temporal feature learning paradigm that requires no labeled data. The predictive learning framework is naturally conducive towards decentralized, federated learning that can help preserve the privacy and reduce the load on the network. We show that the unsupervised segmentation approach performs competitively with fully supervised baselines and even outperforms some supervised baselines. In addition to strong performances on sleep state segmentation, we show
that the framework can be implemented on constrained platforms without any loss in accuracy and latency to move towards providing real-time feedback.

In Chapter 4, we analyzed the properties of different deep neural network layers and provide design-time optimizations for implementing the architecture proposed in Chapter 3 on constrained platforms such as the PYNQ board. We leverage the unique properties of this platform, and through accurate design choices and optimizations, implement highly compact and efficient versions of common CNN architectures that can be extended to various deep learning models such as LSTMs [77], in addition to CNNs. Using our design-time improvements, a low-power FPGA, paired with highly optimized network architectures, consumed $11 \times$ less power and maintaining the state-of-the-art classification accuracy when compared with typical GPU platforms.

Finally, in Chapter 5, we assessed the feasibility of using PUF-based architectures as the core component in the authentication protocol. We explored three machine learning based attack models, including two novel architecture-independent models, that can successfully clone various strong PUF architectures under a variety of scenarios such as unencrypted, encrypted and obfuscated CRPs to simulate real world conditions. We find that the use of unsupervised, generative pretraining helps capture the underlying relationship between the challenge and response, even through the introduction of noise through encryption and obfuscation. This was one of the first frameworks to successfully clone PUF-based authentication in an IoT setting, without any physical access to the device and any prior knowledge of the underlying PUF architecture. We also show that the approach can be extended, through unsupervised noisy, pretraining, to handle two (2) standard encryption protocols and three (3) common PUF architectures, which form some of the more common node authentication setups in practice. Finally, we introduce a machine learning-based defense
called a *discriminator* to help distinguish between the cloned and original PUF to help bolster the authentication process.

Despite such progress, there are some key questions that need to be addressed before the IoMT framework can be successfully extended to millions of users. For example, rapid advances in deep learning has made an essential part of the IoMT framework, particularly for user-centric applications. However, the compute-constrained nature of devices at the edge limits the extent to which the data can be processed effectively, especially given the computational complexity of modern deep learning approaches. How can we move beyond the collection and transmission of data, raw or encrypted, across networks? Decentralized learning is one of the more promising directions, but poses some concerns such as white-box attacks, man-in-the-middle attacks and performance degradation through muddled precision characteristics for compressed models. Additionally, current deep learning models have a “closed world” assumption. They assume that all possible “labels” or rather categories are known during training time, and that novel categories cannot be observed during inference. This assumption is not pragmatic; the real world is dynamic and cannot be captured in a training set. Hence the need to identify novel categories is critical. Finally, deep learning models can require significant resources such as compute power and labeled training data. Enabling the deployment of deep learning models on constrained platforms requires a trade-off between accuracy and SWAP, which might not be an option for mission-critical applications such as real-time healthcare analytics. These are some of the questions that we must explore to enable the development and deployment of truly immersive, secure IoMT frameworks that can scale to millions of users.
References


## References


APPENDIX A: COPYRIGHT CLEARANCE FORMS

The permission below is for the use of material in Chapter 4.
The permission below is for the use of material in Chapter 5.
SPRINGER NATURE LICENSE
TERMS AND CONDITIONS

Jun 28, 2020

This Agreement between Vishalini Laguduva Ramnath ("You") and Springer Nature ("Springer Nature") consists of your license details and the terms and conditions provided by Springer Nature and Copyright Clearance Center.

License Number  4857981244578
License date     Jun 28, 2020

Licensed Content Publisher  Springer Nature

Licensed Content Publication  Springer eBook
Licensed Content Title       Latent Space Modeling for Cloning Encrypted PUF-Based Authentication
Licensed Content Author      Vishalini Laguduva Ramnath, Sathyanarayanan N. Aakur, Srinivas Katkoori
Licensed Content Date        Jan 1, 2020
Type of Use                 Thesis/Dissertation
Requestor type              academic/university or research institute
Format                      print and electronic
Portion                     full article/chapter
Will you be translating?    no

https://s100.copyright.com/AppDispatchServlet
6/28/2020 RightsLink Printable License

Circulation/distribution 1 - 29

Author of this Springer Nature content yes

Title Machine Learning for the Internet of Things: Applications, Implementation, and Security

Institution name University of South Florida

Expected presentation date Jul 2020

Vishalini Laguduva Ramnath
4202 E Fowler Ave ENB 249

Requestor Location TAMPA, FL 33620
United States
Attn: University of South Florida

Total 0.00 USD

Terms and Conditions

Springer Nature Customer Service Centre GmbH
Terms and Conditions

This agreement sets out the terms and conditions of the licence (the Licence) between you and Springer Nature Customer Service Centre GmbH (the Licensor). By clicking 'accept' and completing the transaction for the material (Licensed Material), you also confirm your acceptance of these terms and conditions.

1. Grant of License

1.1. The Licensor grants you a personal, non-exclusive, non-transferable, world-wide licence to reproduce the Licensed Material for the purpose specified in your order only. Licences are granted for the specific use requested in the order and for no other use, subject to the conditions below.

1.2. The Licensor warrants that it has, to the best of its knowledge, the rights to license reuse of the Licensed Material. However, you should ensure that the material you are requesting is original to the Licensor and does not carry the copyright of another entity (as credited in the published version).

1.3. If the credit line on any part of the material you have requested indicates that it was reprinted or adapted with permission from another source, then you should also
seek permission from that source to reuse the material.

2. Scope of Licence

2.1. You may only use the Licensed Content in the manner and to the extent permitted by these Ts&Cs and any applicable laws.

2.2. A separate licence may be required for any additional use of the Licensed Material, e.g. where a licence has been purchased for print only use, separate permission must be obtained for electronic re-use. Similarly, a licence is only valid in the language selected and does not apply for editions in other languages unless additional translation rights have been granted separately in the licence. Any content owned by third parties are expressly excluded from the licence.

2.3. Similarly, rights for additional components such as custom editions and derivatives require additional permission and may be subject to an additional fee. Please apply to Journalpermissions@springernature.com/bookpermissions@springernature.com for these rights.

2.4. Where permission has been granted free of charge for material in print, permission may also be granted for any electronic version of that work, provided that the material is incidental to your work as a whole and that the electronic version is essentially equivalent to, or substitutes for, the print version.

2.5. An alternative scope of licence may apply to signatories of the STM Permissions Guidelines, as amended from time to time.

3. Duration of Licence

3.1. A licence for is valid from the date of purchase ('Licence Date') at the end of the relevant period in the below table:

<table>
<thead>
<tr>
<th>Scope of Licence</th>
<th>Duration of Licence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post on a website</td>
<td>12 months</td>
</tr>
<tr>
<td>Presentations</td>
<td>12 months</td>
</tr>
<tr>
<td>Books and journals</td>
<td>Lifetime of the edition in the language purchased</td>
</tr>
</tbody>
</table>

4. Acknowledgement

4.1. The Licensor’s permission must be acknowledged next to the Licenced Material in print. In electronic form, this acknowledgement must be visible at the same time as the figures/tables/illustrations or abstract, and must be hyperlinked to the journal/book’s homepage. Our required acknowledgement format is in the Appendix below.

5. Restrictions on use
5. 1. Use of the Licensed Material may be permitted for incidental promotional use and minor editing privileges e.g. minor adaptations of single figures, changes of format, colour and/or style where the adaptation is credited as set out in Appendix 1 below. Any other changes including but not limited to, cropping, adapting, omitting material that affect the meaning, intention or moral rights of the author are strictly prohibited.

5. 2. You must not use any Licensed Material as part of any design or trademark.

5. 3. Licensed Material may be used in Open Access Publications (OAP) before publication by Springer Nature, but any Licensed Material must be removed from OAP sites prior to final publication.

6. Ownership of Rights

6. 1. Licensed Material remains the property of either Licensor or the relevant third party and any rights not explicitly granted herein are expressly reserved.

7. Warranty

IN NO EVENT SHALL LICENSOR BE LIABLE TO YOU OR ANY OTHER PARTY OR ANY OTHER PERSON OR FOR ANY SPECIAL, CONSEQUENTIAL, INCIDENTAL OR INDIRECT DAMAGES, HOWEVER CAUSED, ARISING OUT OF OR IN CONNECTION WITH THE DOWNLOADING, VIEWING OR USE OF THE MATERIALS REGARDLESS OF THE FORM OF ACTION, WHETHER FOR BREACH OF CONTRACT, BREACH OF WARRANTY, TORT, NEGLIGENCE, INFRINGEMENT OR OTHERWISE (INCLUDING, WITHOUT LIMITATION, DAMAGES BASED ON LOSS OF PROFITS, DATA, FILES, USE, BUSINESS OPPORTUNITY OR CLAIMS OF THIRD PARTIES), AND WHETHER OR NOT THE PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION SHALL APPLY NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY PROVIDED HEREIN.

8. Limitations

8. 1. BOOKS ONLY: Where ‘reuse in a dissertation/thesis’ has been selected the following terms apply: Print rights of the final author’s accepted manuscript (for clarity, NOT the published version) for up to 100 copies, electronic rights for use only on a personal website or institutional repository as defined by the Sherpa guideline (www.sherpa.ac.uk/romeo/).

9. Termination and Cancellation

9. 1. Licences will expire after the period shown in Clause 3 (above).

9. 2. Licensee reserves the right to terminate the Licence in the event that payment is not received in full or if there has been a breach of this agreement by you.
Appendix 1 — Acknowledgements:

For Journal Content:
Reprinted by permission from [the Licensor]: [Journal Publisher (e.g. Nature/Springer/Palgrave)] [JOURNAL NAME] [REFERENCE CITATION (Article name, Author(s) Name), [COPYRIGHT] (year of publication)]

For Advance Online Publication papers:
Reprinted by permission from [the Licensor]: [Journal Publisher (e.g. Nature/Springer/Palgrave)] [JOURNAL NAME] [REFERENCE CITATION (Article name, Author(s) Name), [COPYRIGHT] (year of publication), advance online publication, day month year (doi: 10.1038/sj.[JOURNAL ACRONYM]).]

For Adaptations/Translations:
Adapted/Translated by permission from [the Licensor]: [Journal Publisher (e.g. Nature/Springer/Palgrave)] [JOURNAL NAME] [REFERENCE CITATION (Article name, Author(s) Name), [COPYRIGHT] (year of publication)]

Note: For any republication from the British Journal of Cancer, the following credit line style applies:
Reprinted/adapted/translated by permission from [the Licensor]: on behalf of Cancer Research UK: [Journal Publisher (e.g. Nature/Springer/Palgrave)] [JOURNAL NAME] [REFERENCE CITATION (Article name, Author(s) Name), [COPYRIGHT] (year of publication)]

For Advance Online Publication papers:
Reprinted by permission from The [the Licensor]: on behalf of Cancer Research UK: [Journal Publisher (e.g. Nature/Springer/Palgrave)] [JOURNAL NAME] [REFERENCE CITATION (Article name, Author(s) Name), [COPYRIGHT] (year of publication), advance online publication, day month year (doi: 10.1038/sj.[JOURNAL ACRONYM])]

For Book content:
Reprinted/adapted by permission from [the Licensor]: [Book Publisher (e.g. Palgrave Macmillan, Springer etc) [Book Title] by [Book author(s)] [COPYRIGHT] (year of publication)]

Other Conditions:

Version 1.2

Questions? customercare@copyright.com or +1-855-239-3415 (toll free in the US) or +1-978-646-2777.
About the Author

Vishalini Laguduva was born and raised in Chennai, Tamil Nadu, India. She received her Bachelor’s Degree in Electronics and Communication Engineering from Velammal Engineering College, Anna University, Chennai in May 2013. Her undergraduate research experience with Dr. Sujatha Rajkumar on communication protocols in the Internet of Things introduced her to the field of scientific research and computing. Following her Bachelor’s degree, she obtained her Master’s Degree in Computer Science with an emphasis on Data Science and Big Data from the University of Illinois at Springfield while working as a Business Analyst at CTSI-Global, India. She later moved to Tampa to work with Dr. Srinivas Katkoori in January 2018. Vishalini was the President of the IEEE Computer Society student chapter at the University of South Florida and has organized many talks and exhibits as part of her leadership responsibilities. She was also an active member of the Women in Computer Science and Engineering (WICSE) in USF CSE Department.