DESIGN OF A VISION BASED ASSISTIVE SYSTEM FOR VISUALLY IMPAIRED PERSONS

A Dissertation in
Computer Science and Engineering

by
Peter Arnold Zientara

© 2021 Peter Arnold Zientara

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

August 2021
The dissertation of Peter Arnold Zientara was reviewed and approved by the following:

Vijaykrishnan Narayanan
A. Robert Noll Chair Professor of Computer Science and Engineering
Chair of Committee
Dissertation Co-Advisor

John (Jack) Morgan Sampson
Associate Professor of Computer Science and Engineering
Dissertation Co-Advisor

Jack Carroll
Distinguished Professor of Information Sciences and Technology
Director of Center for Human-Computer Interaction

Ting He
Associate Professor of Computer Science and Engineering

Chitaranjan R. Das
Distinguished Professor of Computer Science and Engineering
Department Head
ABSTRACT

Globally, more than 285 million people suffer from some sort of visual impairment. To those with visual impairment everyday tasks can pose a challenge. Tasks including picking up a dropped object, navigating to a destination, and grocery shopping can be difficult. While there are many potential solutions to these common tasks, most of them rely on help from others to accomplish them, thus limiting the freedom a visually impaired person can experience in their day-to-day lives. Advances in all areas of technology including cameras, mobile devices, algorithms, and connectivity give rise to the question “Can a computer system be made that is able to assist with these tasks?”.

This dissertation presents an assistive visual device that combines wearable cameras, hardware accelerators, and algorithms that enables users with limited or no sight to select products from grocery shelves. In this system, both algorithms and hardware are designed to leverage the interactions with the shopper to accomplish this task. Additionally, through optimization of such a system including its algorithms, hardware acceleration, and network communication, this dissertation explores how to make such a visual assistive system have sufficient performance and responsiveness as well as the development of user feedback mechanisms needed for such a system to be used in real-world scenarios.

Recent advances in deep neural networks (DNNs) have produced enormous gains, especially in the computer vision domain. While the push for ever-increasing accuracy has grown these networks to dozens of layers and billions of operations, there has been a simultaneous push to harness the power of DNNs on heavily constrained embedded and mobile platforms. This has led to approaches that trade accuracy for efficiency and has fueled the investigation of FPGAs as a means to directly embody these limited precision models while still being able to adapt to the
rapid pace of DNN algorithm development. However, compared to the dominant approach for neural network inference in non-embedded domains, i.e., GPUs, FPGAs currently exhibit substantial drawbacks in programmability, especially in terms of multi-device scenarios. This dissertation also introduces an automated tool flow that goes from DNN definition to embedded system implementation with FPGA accelerated inference and demonstrates the capabilities of the proposed framework in distributing the execution of these neural networks among a set of connected devices. We explore, through simulation, the design space of computational capability and connectivity levels expected across different collaborative embedded system scenarios, such as drone swarms. We then use our flow to perform offload in a real multi-FPGA scenario. We show that, when provided with information about current network and ancillary compute availability, distributing the workload of a neural network can result in speedups up to the limit of available compute, providing the greatest benefits for networks that did not originally target mobile devices.

In total, this dissertation presents a) The design and development of a visual assist platform that incorporates a human-in-the-loop feedback mechanism, b) Optimizations of such a system where there are heterogeneous compute elements (CPUs, GPUs, FPGAs, ASICs) that affect the system’s responsiveness, performance, power, and accuracy, and c) The creation of a design automation framework for distributed intelligence on an FPGA based system on chip.
# TABLE OF CONTENTS

LIST OF FIGURES ................................................................................................................. vii

LIST OF TABLES ................................................................................................................... x

ACKNOWLEDGEMENTS ..................................................................................................... xi

Chapter 1 Introduction ......................................................................................................... 1
  - Assistive Devices ............................................................................................................. 2
  - Operating Environments for Assistive Technologies ................................................... 3
  - User Interfaces ............................................................................................................... 4
  - Sensory Inputs ............................................................................................................... 5
  - Compute and Communication ..................................................................................... 8
  - Algorithms ...................................................................................................................... 9
  - Contributions of this Dissertation ............................................................................ 11

Chapter 2 Design and Optimization of Vision Based System Interacting with Persons with Haptic Glove Interfaces ........................................................................................... 12
  - Infrastructure ................................................................................................................. 12
  - Interfaces ........................................................................................................................ 13
    - Smart Glass .............................................................................................................. 14
    - Custom Glove ........................................................................................................ 14
    - Glove Only .................................................................................................................. 16
  - Vision Algorithms ........................................................................................................ 17
    - Confidence Matching ............................................................................................... 18
    - Expanding Matching to The Shelf ........................................................................... 20
  - Automatic Handoff ....................................................................................................... 21
  - User-Initiated Handoff ................................................................................................. 22
  - Glove Configurations .................................................................................................... 23
  - Challenges .................................................................................................................... 23
    - Feedback Latency .................................................................................................... 24
  - System Refinements ..................................................................................................... 26
    - Head Based Camera and World Tracking ................................................................ 26
    - World Tracking / Hand Tracking / Object Persistence ................................................. 26
    - Object Tracking ........................................................................................................ 28
  - Auditory Feedback Enhancements ............................................................................. 30
  - Haptic Feedback Enhancements .................................................................................. 30

Chapter 3 Distributed Assistive Platform ................................................................................ 32
  - Power Aware Network Partitioning ............................................................................ 33
  - Background .................................................................................................................. 36
    - Image Recognition and Deep Neural Networks ......................................................... 36
    - Distributed Computing ............................................................................................ 36
    - Modes of Communication ......................................................................................... 36
Chapter 4 Conclusion

Algorithm Limitations

Compute Limitations

Network Limitations

Other Implications of Work

Bibliography
LIST OF FIGURES

Figure 2-1: System Overview – (A) Wireless Enabled Smart Glasses, (B) Haptic Feedback Glove, (C) Backend Server ................................................................. 15

Figure 2-2: Glove-camera positioning. The red arrows depict the angle of the actual viewpoint for which directions are provided. The green dashed arrows represent the angle at which the user interprets commands. (a) The misalignment of angles causes the user to veer away from the object. (b) Angles are aligned correctly. (c) Angles are aligned but the object is occluded by the palm. ........................................................................... 16

Figure 2-3: SURF matching of target object once it is in at least partial view – Version A. (A) Commands guide the user to move in such a way that the item will be centered with a straight on viewpoint. Center image shows a rotate command, where they must rotate in addition to moving horizontally or vertically. (B) Graph showing confidences for three items. Low bar is the lowest true positive and middle bar is the highest false positive. The top bar is a safety buffer, additional to the actual confidence of the highest false positive, which is the value that must be surpassed for the system to decide it has found an item. With these confidences item represented by the left column would be in the range of uncertainty so instructions would be given to user to move to get a better viewpoint. The center item would be above the range, therefore certain to be in the image. The item on the right would be below the range, so no instructions would be given. ................................................................. 19

Figure 2-4: Illustration of the hierarchy of running SURF to match at the granularity of the shelf first, then, applying planogram information to match and individual item ....... 20

Figure 2-5: User-initiated handoff from shelf matching with smart glasses to item matching with the glove. (a) Third Eye uses shelf matching and a priori information about the store’s shelf contents to guide the user’s steps toward the desired item. Shelf matching can take place when the user is as far away as 12 feet from the target item. (b) Once the user is within arm’s reach of the item, Third Eye uses object matching to guide hand movement. When the cane touches the shelf, the user initiates the handoff by muting voice commands from Third Eye, which is a signal for the system to begin using the glove camera and guide hand movements through glove vibrations ........................................................................................................................................... 22

Figure 2-6: Top Illustration of camera field of view for near vs far distance front target. Bottom illustrates how from further away unintended movements caused by latency can be corrected since target was still in view ......................................................................................... 27

Figure 2-7: Example of world tracking being able to still guide even if out of view. .......... 29

Figure 2-8: Layout of the Bluetooth enabled glove and communication specification. .......31

Figure 3-1: High-Level System Overview ........................................................................ 35
Figure 3-2: Energy-per-bit for varying levels of transmission power for Wi-Fi 802.11ac.....37

Figure 3-3: Estimated bit error rates for common modulation techniques for a given signal to noise ratio generated using the MATLAB BERTool ..................................................38

Figure 3-4: Illustrates the two methods used to insert noise into intermediate layers. Left: The quantization process. Right: Using the raw floating point.................................41

Figure 3-5: Illustrates the layers of Darknet [1] and cut points given as a percentage into the network.................................................................................................................43

Figure 3-6: Classification accuracy when input to layer is an IEEE-754 single precision float with the mantissa set to '0' from previous layer. Networks Used: VGG-16 [2], Darknet [1], Extraction [3], DenseNet201 [4]........................................................................44

Figure 3-7: Size of input to the layers for each network. Networks Used: VGG-16 [2], Darknet [1], Extraction [3], DenseNet201 [4]........................................................................44

Figure 3-8: Classification accuracy improvement for a network trained under noise in both an input (top) and intermediate layer (bottom). .........................................................45

Figure 3-9: Proposed methods to exploit channel noise. ..................................................46

Figure 3-10: Best radio power consumption to achieve a minimum level of classification accuracy for varying channel conditions........................................................................47

Figure 3-11 Theoretical Performance Gain on AlexNet [5] for N Distributed Devices and Fixed Communication Bandwidth ..............................................................................50

Figure 3-12: Example of the Darknet Format..................................................................52

Figure 3-13: Illustrates the flow of our DNN processing tool. It takes as input a given network format, creates an internal representation of the flow graph, outputs high level synthesis-valid C code, generates the scripts to synthesize the different needed cores, and the scripts to create the final bit stream for a target device. Below, we describe these in greater detail stages. ..............................................53

Figure 3-14: Formula for determining the optimal compute partitioning among two devices..........................................................................................................................58

Figure 3-15: Optimal amount of compute to offload to a remote device (as a %) for a given available network bandwidth for varying levels of available compute for the first layer of AlexNet [5]..........................................................58

Figure 3-16: Optimal amount of compute to offload to a remote device (as a %) for a given available network bandwidth for varying levels of available compute for the first fully connected layer of AlexNet..........................................................59
Figure 3-17: Illustration of applying the optimal compute formula recursively when multiple devices are available to use for computation. .................................................... 60

Figure 3-18: Broadcast performance of TCP vs UDP. ........................................................... 62

Figure 3-19: Simulated speedup of AlexNet [5] across various network bandwidths for 4 different device configurations. ................................................................. 64

Figure 3-20: Simulated speedup of Darknet [1] across various network speeds for 4 different device configurations. ................................................................. 64

Figure 3-21: Simulated speedup of YoloV3 across various network speeds for 4 different device configurations. ................................................................. 65

Figure 3-22: Simulated speedup of VGG-16 across various network bandwidths for 4 different device configurations. ................................................................. 65

Figure 3-23: An example scenario of a drone swarm collaborating on finding a target. ......... 67

Figure 3-24: Offload Patterns: A) No available offload; B) Primary device(A) offloading to device set(B) in Master-Slave pattern; C) Multiple initiators with collective sharing of idle resources. ................................................................. 67

Figure 3-25: Optimal computation distribution across 4 equally capable devices for a master/slave offload of AlexNet. ................................................................. 72

Figure 3-26: Optimal computation distribution across 4 equally capable devices for a master/master offload of AlexNet ................................................................. 72

Figure 3-27: Simulator speedup results for three DNNs distributed over a varying number of network-connected devices ................................................................. 73

Figure 3-28: An illustrative scenario in which multiple distributed devices may assist each other to achieve a common task. In this task many shoppers each with an assistive device can share information from different perspectives about a center aisle display in a grocery store. ................................................................. 78
LIST OF TABLES

Table 3-1: Classification accuracy dropping as bit-errors are introduced into the input image for a network trained on ImageNet ILSVRC2012 and CIFAR-10..........................39

Table 3-2: The classification accuracy for a given bit error rate when bit error is introduced into the quantized value for each layer. .................................................................42

Table 3-3: Different types of nodes and their descriptions used for automatic DNN mapping........................................................................................................................54

Table 3-4: Resource differences for cores generated from templated HLS code and generic code hardware units that support different AlexNet layers (16-bit quantized). ..57

Table 3-5: Mobile device performance in GFLOPs....................................................................63

Table 3-6 Simulated DNN speedup on distributed research drone platforms running Ultrascale+ ZU3CG FPGAs............................................................................................71

Table 3-7: DNN execution time on 1-2 Zedboards compared with simulation....................74
ACKNOWLEDGEMENTS

Along this journey there have been many people who have helped and guided me. Firstly, I would like to thank my advisors Dr. Vijaykrishnan Narayanan and Dr. John M. Sampson. Their guidance, support, and assistance in both academic and non-academic contexts have helped me grow academically, professionally, and personally. Secondly, I would like to thank my colleagues from IST, specifically those within the HCI lab with a special recognition of Dr. Sooyeon Lee, Dr. Jack Carroll, and Dr. Mary Beth Rosson, without whom the research presented here would not have been possible. Thirdly, I want to thank all my colleagues from MDL lab, for the experiences that were had there will be unforgettable. Lastly, but certainly not least, I would like to express my gratitude to all my friends and family who, without them in my life either in whole or in-part I would not be the person I am today.

Chapter 2, in part, is a reprint of the material as it appears in A Multitask Grocery Assistance System for the Visually Impaired Smart glasses, gloves, and shopping carts provide auditory and tactile feedback. IEEE Consumer Electronics Magazine, 6(1), 73-81. The dissertation author was the primary investigator and author of this paper.

Chapter 2, in part, is a reprint of the material as it appears in Third Eye: a shopping assistant for the visually impaired. Computer 50.2 (2017): 16-24. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in part, is a reprint of the material as it appears in Noise Aware Power Adaptive Partitioned Deep Networks for Mobile Visual Assist Platforms In 2018 31st IEEE International System-on-Chip Conference (SOCC) (pp. 186-191). IEEE. The dissertation author was the primary investigator and author of this paper.

This work was supported in part by NSF Expeditions in Computing Program: Visual Cortex on Silicon CCF 1317560. Any opinions, findings, and conclusions or recommendations
expressed in this publication are those of the author and do not necessarily reflect the views of the NSF.
Chapter 1

Introduction

Globally, more than 285 million people are visually impaired; for them, tasks that are deemed trivial by those with normal sight—such as picking up a dropped object—are a major undertaking. In recent years, breakthroughs in visual sensing technology have stimulated efforts to help persons with visual impairment (PVIs) become more independent. Signal-processing technologies, central to visual augmentation, help PVIs navigate inside and outdoors [6] and identify objects—activities that are critical in everyday tasks such as shopping, particularly in supermarkets. Even sighted people are often overwhelmed by the amount and variety of products in a typical US grocery store, which can have 35,000 unique items presented in as many as 30 aisles across 45,000 square meters. For PVIs, grocery shopping is like navigating an area the size of a football field with moving carts, people, and displays at every turn. To accomplish this daunting task, some PVIs turn to family, friends, or community volunteers. Others must rely on store assistants, who can make the shopping experience enjoyable or unpleasant, depending on their knowledge and personality. Online shopping and home-delivery services attempt to mitigate the challenges.

Grocery shopping is an essential activity in our daily lives that involves various interconnected activities. Typical shopping tasks include checking the pantry for current inventory, making a shopping list based on planned meals, getting to the store, and making opportunistic and impulsive purchases in response to signage at the store. Each one of these activities poses a significant challenge without visual cues. Consequently, a system which enables assistance with such tasks must utilize a combination of hardware-software mechanisms to
interpret these visual cues and communicate them to a visually impaired user through multiple modalities.

Modern advances in both computation hardware such as CPUs, GPUs, FPGAs, and ASICs as well as the types of algorithms used, are required to provide such assistance to the visually impaired via a computing system. Many vision-based assistive systems have become available on the open market. This thesis describes both the challenges and successes in building one of the first vision-based assistance system for visually impaired persons.

Using computer vision to assist visually impaired persons with a task such as grocery shopping requires technology advancements in all areas from algorithm to compute and in other areas such as battery technology and networking, as well as from state-of-the-art image recognition algorithms to custom hardware used during execution. This thesis focuses on a progression of work in creating one of the first systems meant to assist visually impaired persons with this task.

**Assistive Devices**

Assistive technologies over the last 5 years have gone through many developments. As technology has improved, so have the application of assistive technologies. Various types of assistance devices exist from network connected to standalone on device as well as hybrids of the two.

Some examples of these assistive technologies are phone-based applications such as TapTapSee [7], be my eyes [8], and Seeing AI [9], attempt to assist visually impaired individuals. Each of these applications attempts to assist visually impaired individuals in a different manner. For example, Seeing AI [9] and TapTapSee [7], both leverage light-weight object recognition algorithms that are able to be run on a mobile device, while be my eyes [8] crowd sources
assistance for visually impaired individual by creating a video call session with a volunteer to assist the visually impaired individual. Each of these assistive technologies has their own trade-offs and design decisions that are based around intended use cases, environments, physical form factor constraints, as well as various requirements for either network connectivity or mobile compute ability.

**Operating Environments for Assistive Technologies**

Assistive technologies can be designed to perform their functions in a variety of environments. Each of these environments affects the difficulty of the assistive task. These environments range from fully controlled environments to fully dynamic environments.

The most basic operating environment an assistive device can operate in is a fully controlled environment. In a fully controlled environment, everything about the desired task, or in the case of grocery shopping the intended object/target to pick up, would be known. Such an example of this would be a controlled lab setting. In such a setting the location of a target object and its surrounding object is known and does not change. An assistive technology that is meant to be used in such an environment can take advantage of this knowledge to make the task easier. Additionally, since in a controlled environment objects either do not move or move infrequently, even the knowledge of surrounding objects may be used to assist with the task.

The next level of operating environments are semi-controlled environments. In these environments, objects are able to be moved, or may not exist, however they will still be within the same relative area, in semi-controlled environments it is expected that the environment will tend to reset itself to a known state when changes occur. This type of environment also allows information acquired out-of-band to be used by an assistive system. One such example of out-of-
band information would be a planogram. This body of work started in a semi-controlled environment.

The final class of environment that an assistive system or device can operate in is that which is fully dynamic. A fully dynamic is an environment is one in which there exists minimal if any prior knowledge and information about the environment and objects in it. Knowledge of this class of environment can only be ascertained through sensing and processing sensory information. An example of a fully dynamic environment would be out on the street at a previously unexplored place.

**User Interfaces**

For an assistive device to perform its function of assisting an end user, such a system or device needs to be able to communicate with the user. In humans, this communication can happen via visual, auditory, or tactile feedback.

Visual feedback can use displays to augment the vision of the end user, display messages through text, or indicate through display of other like symbols. In making a system that assists visually impaired individuals this method of communication is the least effective. However, given the wide range of visual impairments that can exist, it is not the case that it is completely ineffective.

Auditory feedback is feedback that is provided to the user through the generation of audio waveforms. These sounds can be in class of verbal instructions similar to that of system which provide text-to-speech functionality or feedback which can use a periodic sound such as a beeping or a tone which may vary in either, pitch, frequency, or intensity (volume).
The third type of feedback which may be utilized by an assistive system is tactile feedback which is providing information through a user’s touch senses. A common usage of this type of feedback are vibration motors.

This body of work used all three types of feedback mechanisms to the end user with the most feedback coming through a combination of auditory feedback and tactile feedback. Auditory feedback used both continuous and discrete sounds, while the tactile feedback was provided through a custom-made glove that the user wore which contained multiple vibration motors within it.

**Sensory Inputs**

Sensor inputs allow for a device to perceive and interpret the world around it. Algorithms meant to process data to produce these interpretations require quality data from any given sensor or multitude of sensors to be accurate. For a computer vision-based system the primary sensor input is a camera sensor. These vision sensors are available in two modalities, 2D and 3D. The 2D camera sensors only capture RGB input and relative position on an X, Y plane. Cameras that are 3D provide additional depth information which can be used to estimate or measure depth as well. These 3D cameras usually achieve this depth measurement through either stereoscopic vision which is using two 2D cameras a fixed distance apart to calculate depth or Active IR cameras which estimate depth by projecting a known pattern then calculating how that pattern is distorted.

Additional sensors may be used by an assistive device which when fused with the primary vision sensor can improve the efficiency and accuracy of the vision algorithms that process the vision data. Audio inputs through microphones can be used to estimate moving objects that make sounds similar to that of sonar. When combined with a speaker, modalities such
as echo location can be achieved. Inertial measurement units (IMU) are sensors which contain a suite of sensors such as magnetometer, accelerometers, and gyroscopes. The environment that these sensors are deployed in can determine how they are utilized. For example, in controlled environments and environments with prior knowledge a magnetometer can be used to assist in determining location. Other uses for an IMU include tracking movement, movement information can be used as a filter for the vision input since frames that are acquired with a large degree of movement are less detailed and may not be as accurate than images captured with a lesser degree of movement. In systems that are either battery or network limited, this can save resources by ignoring inputs that are lower fidelity. Additional uses could center around providing input that can help with feedback to the end user by positioning the sensors on the individual. An example of information that can be acquired include arm and hand position along with information such as movement vectors.

The system developed and described through this body of work relied on its input mainly from off-the-shelf hardware. As a consequence of this the capability and inputs to the system over time evolved. The devices used were typically known as smart glasses: These were chosen due to the requirement of being head mounted, this allowed the frame of reference of the vision input to the system to be closer to the true position of the user. These smart glasses also provided a mechanism for both user feedback via a speaker, and input through some type of touch sensitive pad which could be used to control the device. Early smart glasses used were Recon Jet glasses. These glasses provided a basic camera and network connection, while on board was a basic mobile CPU no meaningful amount of computation could be done on the device. Many of the early devices used in the assistive system were also limited by their computational ability. One of the main limitations of Recon glasses was that it did not contain a hardware image encoder. This meant that order to have a useful frame rate being sent back to the server used for the compute the input image had to be scaled down since it was being sent as a raw frame. Experiments on the
glasses between using a software encoder versus sending raw frames showed that sending raw frames was more performant. When the Recon glasses got replaced by Google Glasses the limitation of raw frames was removed since the Google Glasses contained a hardware image encoder. This allowed the images captured by the device to be offloaded at full resolution and over 20 frames per second. Other aspects of the Google glass were similar to that of the Recon including its input and user feedback mechanisms. Both the Recon and Google Glasses were also limited by battery life and when in use in the assistive application could only last around 15min.

The next device used in this assistive system was the Microsoft HoloLens. The HoloLens provided a jump in capability in all aspects of the front facing device. Capabilities that were comparable to the previous headsets were higher bandwidth Wi-Fi and networking, battery life that lasted 2 hours, and improved image resolution. The HoloLens also provided a new set of capability which further enhanced the visual assist system. One of the primary enhancements that the HoloLens provided was additional cameras and sensors which provided 3D vision that enabled an out-of-the-box world tracking and user hand-tracking capability. This additional provided capability combined with more compute power enabled less computation to be offloaded, as well as having a more responsive user feedback loop as the device was aware of the user’s position relative to objects in the surrounding in real-time. This knowledge of the user in relation to the world also enabled the system to get rid of the glove mounted camera. By removing the camera from the glove, we were able to create a wireless Bluetooth glove that provided haptic feedback to the user. This feedback mechanism could also be controlled from the headset itself through Bluetooth rather than in previous versions of the system having to be controlled from a different compute device.

Although the HoloLens was the last device to be utilized by the system, next generations of the device further refined the capabilities and could be utilized and improve the assistance provided.
Compute and Communication

For a system providing assistance which makes heavy use of computer vision, having a sufficient amount of compute is necessary. To process the large amount of data in a timely manner, computation can be described as either host-based compute, edge-based compute, or a mix of the two. In host-based compute, computations are offloaded to a backend server which can contain a mix of accelerators such as GPUs, FPGAs, or ASICs, as well as more powerful CPUs all of which make the server high power. While host-based compute systems are limited by network connections which introduces additional latency, additional cost, and having to be shared with multiple edge clients. Edge-based compute is typically achieved on mobile and embedded SoCs which are limited by their form factor. Although limited compared to the compute power available to the host-based systems, edge-based compute often benefits from having dedicated hardware accelerators for certain computations such as video compression, as well as having sensor inputs local. Another benefit of edge-based compute is that since both sensing and compute is local and thus no existing infrastructure is necessary. Over time, as technology has improved, the hardware and computational power that can now exist at the edge has alleviated some of the issues with required offloading computation. For example, dedicated image processors on SoC enable higher quality images to be captured at the edge, while vector capable processors enable more performance. Even more recently, as algorithms have evolved to take advantage of deep neural networks, dedicated accelerators for these neural networks have started to become built in on these embedded SoCs as well. Advances in consumer technology have led to these accelerators becoming abundant inside everyday devices such as smart phones and IoT devices. Dedicated ASICs such as Google TPU, Apple Neural Core, Qualcomm Neural Engine, have become common place in many consumer devices. However, even though these improvements enable more computation to be done at the edge, problems which require larger
amounts of computation still create concerns around using offloaded computation. These problems include how to partition various workloads, as well case issues around communications such as latency and bit errors.

**Algorithms**

Over time, the algorithms that have been used for all tasks of image processing tasks including object detection, object recognition, and object segmentation have progressively improved and evolved. Historically these algorithms were handmade and operated using features that were progressively engineered over time. Examples of such feature engineered algorithms are SURF [10], SIFT [11], AKAZE [12] [13], and AIM. These engineered features would require custom accelerators to be implemented in FPGAs to assist with the processing of these algorithms. However, as deep neural networks, specifically deep convolutional neural networks have been applied to more areas, these computer vision algorithms have also benefited from their development. DNNs are trained on data without any pre-engineered features. Instead, these algorithms learn the required features as necessary from the data that are given as input. Consequently, these features are learned, and the structure of the networks are regular. The computations that these networks require are also very regular and are typically implemented using various versions of matrix-matrix multiplication. Regular computation has enabled hardware accelerators of this computation to exist and be implemented across a variety of form factors from server based PCIe accelerator cards to mobile SoCs which can exist within mobile phones.

This evolution of algorithms has enabled the various computer vision-based tasks and algorithms to be available on such mobile devices. This enablement has contributed to the evolution of mobile devices and smartphones. Enabling applications that take advantage of these
algorithms, mobile phones have been able to provide more value to the end user. Additionally, the development of mobile phones has also enabled improvements in the assistive technology space, as other improvements such as form factor, network, and power (battery) can be leveraged to alleviate historical limitations on edge-based compute devices and systems that utilize them.
Contributions of this Dissertation

This dissertation makes the following contributions:

1. Design and development of a visual assist platform that incorporates a human-in-the-loop feedback mechanism.
2. Optimizations of such a system where there are heterogeneous compute elements (CPUs, GPUs, FPGAs, ASICs) which affect both the responsiveness of the system in performance, power, and accuracy.
3. Design automation framework for distributed intelligence on FPGA based system on chip.
Chapter 2
Design and Optimization of Vision Based System Interacting with Persons with Haptic Glove Interfaces

Grocery shopping is an essential activity in our daily lives that involves various interconnected activities. These include checking the pantry for current inventory, making a shopping list based on planned meals, getting to the store, and making opportunistic and impulsive purchases in response to signage at the store. Each one of these activities poses a significant challenge without visual cues. Consequently, as part of the Visual Cortex on Silicon program, this dissertation enables a combination of hardware-software mechanisms to interpret these visual cues and communicate them to a visually impaired user as verbal or vibrational feedback. This chapter elaborates the design exploration of this Third-Eye Visual Assist platform.

Infrastructure

As Figure 2-1 shows, Third Eye consists of off-the-shelf smart glasses—equipped with a camera and audio channel—connected to a back-end server system that supports real-time video analytics. The camera is oriented to the user’s right eye, and the smart glasses’ field of view and resolution dictate the distance at which objects can be properly located and identified. Users also wear a glove with a camera on the hand used to grasp a product. The glove camera, which guides hand movements, must also orient the user to keep the product in view and avoid any occlusion from the arm used to point to and grasp it. Third Eye uses the store’s wireless infrastructure to send the video stream to the server, where computer vision algorithms analyze it and send back results. This feedback then becomes the basis for providing either audio commands or tactile vibration patterns that guide the user’s steps and hands toward the desired item. To avoid latency between image capture and feedback signals, Third Eye incorporates a local wireless routing
infrastructure, which ensures robust, low-latency communication with the server, and customized hardware to accelerate the vision algorithms’ computations. Images from the smart glasses are the basis for directions to move the user closer to the desired item. Images from the glove camera then help orient the user to a view that provides enough information to the vision algorithms.

Inspired by prior efforts that used sensor-equipped haptic gloves for visual search and interaction with smart glasses [14] [15], we incorporated vibrational feedback for hand movements into the user’s glove. The glove has four micromotors to direct movement (up, down, left, and right), and simultaneous vibration of all four motors directs the hand forward to grasp a product that is in position. Although audible feedback has been used to provide directional navigation, based on observing PVIIs and their interactions in store environments, we opted to supplement it with tactile feedback in the form of vibrations. We found that it is not effective in noisy environments and draws unwanted attention from other shoppers. A headset might address these challenges, but it would have to be highly specialized, incorporating bone conduction or the superimposition of instructions while allowing for environmental noise—a key navigational aid for those with visual impairment.

**Interfaces**

For our assistive technology we employ two main modes of providing feedback and guidance to the user. These modes are auditory feedback and tactile feedback. To provide this feedback to the users, we use the glove and the glasses as listed above.
**Smart Glass**

The off-the-shelf smart glass provides the system with a head view as well as network connectivity and speakers for audio feedback. In the assistive system, the glasses are mainly used to guide the person at the aisle level to be in front of their intended/desired product. The commands such as “left”, “right”, “forward”, “back” provide the necessary direction.

**Custom Glove**

The custom glove we employ has both a camera and a series of vibration motors. This camera that is on the glove allows the system to have the viewpoint of what the person is reaching out for. This viewpoint may be different from that of the camera mounted on the headset and is critical to being able to provide guidance all the way to physically picking up the intended product. The attached vibration motors allow the glove the system to provide subtle feedback to the user to convey to them which direction they would have to move their hand to be able to grab the desired product. An example of this would be buzzing the right motor to indicate a rightward motion or the top motor to indicate the person needs to lift their hand.
After our proof-of-concept study, we evaluated the auto handoff we first used in version B of SURF. For this evaluation, we used blindfolded sighted users who were at least 10 feet away from the shelf and gave them only audio feedback and little a priori instruction. In most cases, the audio feedback system successfully brought the user in front of the desired object. In some cases, participants misinterpreted the directions, such as “move left,” which they understood as turning left instead of moving left. Also, directions like “move left” or “move forward” did not specify how far, which led to significant variations in distance moved. Based on these results, we

Figure 2-1: System Overview – (A) Wireless Enabled Smart Glasses, (B) Haptic Feedback Glove, (C) Backend Server
incorporated more precise directions, such as “move one step forward” or “move less than a step left.”

**Glove Only**

In another evaluation, we tested the camera system with only the glove camera and the vibrating motors to help with the final object grab when the user was within 3 feet of the target object. In this test, we used version A of SURF. Camera placement was a major challenge because we had to orient the camera in such a way that hand posture would not occlude the field of view— the main reference for providing movement instructions. In the prototype Third Eye, the camera was parallel to the arm to ensure that the hand and view were aligned. Unfortunately, in some experiments, the camera’s orientation changed because of sudden jerky hand movements, which resulted in a lost field of view and consequent direction to the wrong objects. Figure 2-2 illustrates some of the glove design challenges we faced.

Figure 2-2: Glove-camera positioning. The red arrows depict the angle of the actual viewpoint for which directions are provided. The green dashed arrows represent the angle at which the user interprets commands. (a) The misalignment of angles causes the user to veer away from the object. (b) Angles are aligned correctly. (c) Angles are aligned but the object is occluded by the palm.
We also experimented with several configurations of the glove’s vibrational feedback. After several positional trials, we arrived at a design that placed the micromotors on the fingers of the left hand to guide movement: the thumb to indicate a movement to right, the little finger to indicate a movement to left, one on top of the palm for up navigation, and one below the palm for down navigation. We continue to investigate the effectiveness of vibration intensities, durations, and patterns produced by motors for tactile messaging [16].

In previous tests, users were sighted but blindfolded, which we found does not accurately represent a PVI, who has had time to adjust to visual impairment. We only recently formatively assessed the Third Eye system with a PVI who has been working closely with our NSF project for the past two years, carrying out basic shopping interactions with a simulated store shelf. This individual had also participated in previous studies of a human-recognition prototype and field studies of PVI shopping practices. This experience made this individual an excellent feedback source in evaluating our functional prototype.

**Vision Algorithms**

Third Eye’s vision algorithms assist users from the time they enter the store to the time they leave it. The first task is to identify the correct aisle. When users are in front of the products in the aisles, the camera captures the image and uses a gist-of-the-scene algorithm to localize their position [6]. The algorithm extracts a low-dimensional signature of the entire image that can support scene classification. For each videoframe from the camera, it aggregates statistics about the scene, including color distribution, luminance, and oriented edges. The aggregated statistics summarize the scene contents and translate it to a gist vector—a low-dimensional holistic feature vector—which a support vector machine (SVM) classifier processes to produce a category label for that scene, such as a cereal or coffee aisle. If users are not in front of a desired aisle, they
move to another aisle and repeat the process until they arrive at the desired aisle. Navigation between the aisles is not yet completely automated, but our system effectively integrates with the navigation skills PVIs already have.

Once we have localized the aisle, we employ a feature extraction algorithm called Speeded Up Robust Features (SURF) [10] to locate the product within the shelf. The feature extraction is followed by a template matching algorithm which matches keypoints in a template image with keypoints in the camera frame. Each keypoint match provides a location of the point in the new image along with a confidence of the match.

**Confidence Matching**

Most algorithms which generate confidences either have a fixed threshold to decide when the target item is in the camera frame or, if testing for multiple items, will select the one with the highest confidence. When using a fixed confidence threshold, most algorithms in most datasets will make mistakes [17]. The highest confidence calculated for each item when that item was not in the camera frame - the highest false positive - tends to end up being higher than the lowest confidence calculated when that item was in the camera frame - the lowest true positive. In all cases where the highest false positive is stronger than the lowest true positive there will be a range of uncertainty, where using a fixed threshold will guarantee mistakes. As shown in figure 2-3 our algorithm takes this into account by only giving answers when confidence is above the highest false positive.

However, within this range of uncertainty, keypoints are still able to be matched between the template image and the camera frame. Based on the differences in relative positions of these keypoints, a homography matrix can be calculated. This matrix is a representation of where the camera is relative to the item currently, compared to where it was when it took the template
image, which in our experiment are always front and centered viewpoints. Once the system knows where the camera is relative to the item it can instruct the user to move to get a more front and centered viewpoint. When the camera captures images that are more front and centered, and therefore closer to the same view as the template image, the confidences will be higher.

Figure 2-3: SURF matching of target object once it is in at least partial view – Version A. (A) Commands guide the user to move in such a way that the item will be centered with a straight on viewpoint. Center image shows a rotate command, where they must rotate in addition to moving horizontally or vertically. (B) Graph showing confidences for three items. Low bar is the lowest true positive and middle bar is the highest false positive. The top bar is a safety buffer, additional to the actual confidence of the highest false positive, which is the value that must be surpassed for the system to decide it has found an item. With these confidences item represented by the left column would be in the range of uncertainty so instructions would be given to user to move to get a better viewpoint. The center item would be above the range, therefore certain to be in the image. The item on the right would be below the range, so no instructions would be given.
Expanding Matching to The Shelf

In version A, the goal was to get the user closer to the template orientation view. When the camera captured images that were nearer to this view, confidences were higher. However, when multiple objects on the same shelf have similar features or when the user is not close enough to the shelf for the algorithm to extract a sufficient number of feature points from the viewed object, there is no partial match and no way to calculate a homography matrix to increase confidence. To address this problem, Third Eye uses shelf wide feature extraction and matching, which is based on the store’s planogram (a diagram of how products are placed on shelves), to guide the user closer to the shelf section that contains the desired item. Once it matches a shelf, Third Eye uses the planogram to locate desired items and then analyzes the image only at a shelf.

![Illustration of the hierarchy of running SURF to match at the granularity of the shelf first, then, applying planogram information to match and individual item.](image-url)
location that contains those items. Figure 2-4 illustrates the hierarchical shelf match followed by a planogram-based match. For example, if a shopper wanted a particular brand of spaghetti, Third Eye would not analyze the entire rice and pasta aisle, only the shelf that contained pasta and only the part of the shelf that contained spaghetti. Narrowing analysis to a particular shelf section eliminates some of the problems in version A of our algorithm, in which shoppers had to wait for Third Eye to determine that images were irrelevant before hearing additional instructions. Rather, in version B of our algorithm, key-point matching is for the entire shelf section and the user can be as far away as 12 feet. Consequently, many more feature points are available in determining a match, which enables robust identification among multiple similar objects on the shelf.

**Automatic Handoff**

We gave the PVI the smart glasses and only audio feedback with auto handoff (version B of SURF) but no vibrating glove. The PVI’s initial position was about 10 feet away from a mock shelf containing nine products from the cereal aisle. The PVI then selected the object by moving through an audio menu in the glasses. After selecting the target object, the PVI was able to follow the audio feedback and navigate to a point close to the target using guidance from the speech feedback, which was based on images from the glass’s camera. The speech feedback (for example, “move two steps slightly to the left”) conveyed both direction and magnitude of movement, which enabled her to stay on the desired path. Once the PVI was successfully positioned in front of the identified object, head alignment was not as steady as what we observed for blindfolded sighted users. We also noted that our estimate of the depth from PVI to shelf was not always reliable. Despite these drawbacks, the PVI consistently used Third Eye as an augmentation tool, stopping when her assistive cane touched the shelf.
User-initiated Handoff

Based on the results of the auto handoff test, we conducted another experiment with a user-initiated handoff, in which the PVI employed the cane in transitioning from the glass’s camera and instructions for body movements to the glove camera and instructions for hand movements. This handoff occurred when the PVI’s cane hit the bottom of the shelf, at which time the PVI directed Third Eye to mute audio commands based on the glass’s camera. With this user-
initiated handoff, the PVI was able to pick the desired object in the next trial. However, in other trials, the same PVI selected an object just below the desired object. This user-initiated approach is more robust at this stage than a completely automated handoff user-initiated handoff and depicted in figure 2-5. These trials implied that objects positioned above the glasses camera’s field of view were challenging and that we needed to further refine the directions provided.

**Glove Configurations**

In the final test with the PVI, we evaluated glove configurations to determine the best camera orientation and whether feedback through vibrations or audio was more effective. The PVI seemed to prefer vibration over the audio feedback, pointing out that audio feedback without a headset draws unnecessary attention during shopping and interferes with the environmental sounds that offer cues to a PVI in orienting body position. We also observed that the glove camera’s orientation required more robust engineering. The lack of orientation reference caused the PVI to miss objects by less than a few inches in all trials. Moreover, when a miss occurred, the glove camera’s field of view was occluded or the PVI was too close for the system to confidently identify the object. In these cases, the system stopped providing any directions for missed items because they exceeded the highest confidence score.

**Challenges**

Creating a truly assistive system with a variety of interfaces presents a series of challenges, not all of which are initially obvious. These challenges include guiding the person through the store which includes the challenge of localization, obstacle and person avoidance, and grocery shopping. Other challenges are user centric. These include adapting the frequency of
guidance commands to the speed at which the person is moving, reconciling different camera views to provide correct guidance, and having enough computational power to keep the system real-time. These challenges can be resolved via various methods. To solve the problem of guiding the person through the store, the smart cart could be equipped with various sensors. This could include cameras that not only have RGB information, but even depth and possibly thermal sensors. The use of localization technologies such as indoor GPS, and Bluetooth beacons around the store have the ability to track the user and provide the needed level of localization to the system (e.g., aisle location). The challenge of reconciling different camera views arose from having two camera views that are not always in alignment with one another (the glove and the glasses). An example of when this occurs is while shopping when the user goes to grab a product, they might look away while still reaching in towards the intended product. This poses a challenge to a system giving guidance based on the view from those cameras. One possible solution to this issue would be the addition of sensors to the glasses and the glove. The addition of an IMU and Magnetometer to both edge compute solutions give the ability to correctly provide guidance in this case, and other similar cases. An example how this would work is if the headset camera view indicates the person needed to move right, but the glove camera was pointing straight. The system would be able tell the user to turn just their head to align the two views, rather telling them to step right.

**Feedback Latency**

While all of the listed challenges can be considered implementation details, the biggest challenge that exists in an assistive system is being able to keep up with the real-time demands of the user. Even when the algorithms were running on an IBM POWER8 server with 160 logical cores operating at 3.6GHz, for a 1920 × 1080 image of an entire aisle, our latencies were 375 ms
with 9,000 feature points in the image. We reduced latency to 170 ms by limiting the camera’s field of view to a single shelf, which has around 4,000 feature points. Despite this latency reduction, we observed that delayed feedback from the system along with sudden jerky movements by shoppers caused the cameras to lose the object. To mitigate this problem, we used the NVIDIA GPU K1200 to accelerate object detection (shelf or item), which reduced feedback latency to 110 ms—enough to provide a testbed for a single user. From the results of testing with the K1200, we designed an accelerator to run on field-programmable gate arrays (FPGAs) that further reduces latency to support more concurrent video streams.

With an assistive system, solving this challenge is critical. In order to do this effectively, the system, as a whole, must leverage all available compute, including the compute power, however limited, available at the edge devices and the local infrastructure. As stated earlier, for our cloud compute device we use a high-performance server that is enabled with both FPGAs and GPUs. By leveraging custom architectures and exploiting parallel algorithms, we are able to process 1080p video frames at around 50fps. While this may seem like it meets the real-time constraint it does not. This is because a server needs to be able handle multiple connections at once. Our current accelerated back end would be able handle about 50 streams at 1fps. To make up this gap in performance, tricks need to be played at the local and edge compute devices to make this disparity become imperceptible. Some of the compute that can be offloaded to edge devices and local infrastructure are mainly filtering processes. For instance, during the product detection phase a local infrastructure would be able to run the images being streamed back to the server through one of our hardware accelerated saliency algorithms. Additionally, the edge device could use its sensors to only send a frame when the user has moved enough that the scene needs to be recomputed fully. Once the products are detected, the local infrastructure or edge device would be able to run a computationally less demanding tracking algorithm to be able to continue to guide the user toward the product between communications with the cloud back-end.
System Refinements

In the various iterations of Third Eye refinement, we identified two major problems: feedback latency, which we eventually solved, and power drain, which has been mitigated.

Head Based Camera and World Tracking

While this first system provided many solutions to the challenges based on around the computation aspect of a visual assistance system, many challenges existed around the user facing aspect of the system. These challenges include perception problems, processing limitations, as well as user feedback.

To address the short comings of the first version of the assistance system for visually impaired systems, and to take advantage of the many improvements of technology during the time period, the second version of the system added technologies such as a head-based camera, hand-tracking, and world tracking. These additions along with other processing, battery, and network improvements were able to improve the responsiveness of the system. Each of these additions and their effect are detailed below.

World Tracking / Hand Tracking / Object Persistence

One of the main impediments of the first system was the computation that was available at the edge / on the end user of the system. This meant that the wearable device was mainly used for sensory input such as camera input and user feedback. Any processing on board was directed towards reducing the amount of data that needed to be sent to the backend server for processing either through compression or low-level filtering such as a small saliency filter.
World tracking enabled continuous audio as well as discrete commands which creates more timely commands. In version one of the systems all instructions for user feedback were generate remotely from the backend server which was doing all the image processing. This created two main issues. Each issue was created by the latency between the server and the user wearable device. In the first version system two different cameras were used. One on the head
to shelf track and guide from 12-15 feet away from the shelf up to arm’s length away (2-3 feet). The second camera was on the glove to be able to track the position of the hand used to guide from 2-3ft to the pick-up of the target object. For the larger movements guided by the head mounted camera, the latency between commands and movement was mitigated by the slower larger body movements. Additionally, being farther away mean the camera could see the target shelf through a wider range of motion. This allowed for any errors cause by updated instructions for movement could be corrected if the target was still in view. For example, if the user moved too far in one direction moving past the intended position, a following feedback would be able to guide in the other direction. While unintended movements could more reliably be corrected from these larger slower walking movements, this was not the case for the hand mounted glove camera. Since one’s arm can move faster than the body as a whole (walking), the movements were more likely to require correction due to the latency from the backend server. To compound this error, because of distance involved, the camera may be unable to correct due to the object being out of sight. This problem is illustrated in figure 2-6.

Although the compute on the front-end device was still not powerful enough to perform the full recognition, added features on the device used which was the Microsoft HoloLens [18] enabled World Tracking and hand tracking capabilities. These additional capabilities were able to improve the responsiveness of the system in both processing as well as providing my timely feedback and instruction.

**Object Tracking**

World and hand tracking enable the front-end device to keep track of the location in the physical world even if the user turns away and the target is out of view of the camera. This ability enabled two improvements toward the overall system. The first was that no longer a glove based
camera was necessary to provide the feedback at close ranges. This meant that a user-initiated handoff was no longer necessary. Additionally, even when the target object was out of view the user could still be guided to it since the location of the object was known in 3D coordinates relative the the user. The second benefit to the addition of world and hand tracking was that the server no longer had to perform recognition continously. Once the target object was located, the front-end device no longer had to communicate with the back end and directions could be generated fron the user-worn device. This both reduced the reliance on network connectivity as well as greatly reduced the latency for producing user feedback. Instead of being limited by the time it takes to communicate with a server, the feedback could be generated reliably with a user defined rate. By utilizing object tracking the user-worn device is able to continously give directions even if the desired target is momentarily out of view as depicted in figure 2-7.

Figure 2-7: Example of world tracking being able to still guide even if out of view.
Auditory Feedback Enhancements

User configured feedback latency and locally generated feedback as opposed to server generated enable an additional mode of user-feedback. While the latency of the first system was limited to providing feedback in discrete increments, which resulted in a limited direction set of “left”, “right”, “up”, “down” “forward”, “step back”, and “reach and grab”, user configured feedback rates enabled a continuous auditory feedback mechanism. This continuous auditory feedback is in two different forms, the first being a beeping noise that would beep more frequently as the user got closer to the desired object and the second being a continuous sound which changed pitch as the user moved closer or further to the desired object. Both continuous audio modalities are used to improve guidance of the end user by enabling instantaneous feedback for situations in which a user’s movement did not align with the intended instruction. These new supported audio methods could still be used in conjunction with the previous auditory commands.

Haptic Feedback Enhancements

The first generation of the haptic glove was wired connection driven from a tablet computer. This setup had a limited amount of control on each of the individual motors that utilized simple on/off feedback. The second generation of the haptic glove instead was battery powered with a Bluetooth connection. This Bluetooth connection enabled complete freedom of movement to the wearer as well as allow the headset wearable device (HoloLens) to issue control commands to the motors. Since control was over Bluetooth optimizations, the control of the motors had to be made due to the Bluetooth latency. To keep the latency to a minimum while also enabling fine grain motor control, command instructions for all five motors were sent using a
single 32-bit Bluetooth attribute write. As detailed in figure 2-8 this 32-bit packet was subdivided into five 5-bit segments where each segment represented control for a different motor. Within each of these 5-bit segments a single bit represented 0.1 seconds of motor control with a 0 meaning off and 1 meaning active. In total this new control structure allowed for 0.5 seconds of control for all 5 motors simultaneously. These commands could then be sent continuously every 0.5 seconds from the host-device without experiencing any delays or stalls. Additionally, this level of 0.1s of control allowed for various patterns to be executed using the motors such as a rotating motion or alternating pulses on a set of motors.

Figure 2-8: Layout of the Bluetooth enabled glove and communication specification.
Chapter 3

Distributed Assistive Platform

The second part of my work concentrates on how image classifications are accomplished and how due to the computational requirement’s, many models completely offload the computation over a network connection. Due to limitations in both compute power and battery capacity, many use cases for mobile devices often involve the offload of some portion of a task from the mobile platform to edge or cloud servers. Where there is flexibility in either the degree of offload or the nature of the communication to the remote device, there can be substantial tradeoffs between the amount of energy consumed by the mobile device for a given performance or quality of service (QoS) target. This dissertation investigates these tradeoffs in the specific case of deep neural networks (DNN(s)), which are known to both have noise tolerant properties and present many partitioning options for what portion of a task is computed locally versus remotely, and we present a scheme that exploits DNN QoS tolerance under reduced transmission fidelity to reduce mobile device power requirements. We characterize the error robustness of several networks as a function of cut depth, showing that resilience decreases as a function of layers and the degree of mismatch between training and operational noise levels, and develop an adaptive technique for run-time selection of an appropriate model, offload point, and transmission power level for a given noise environment.

This dissertation also introduces an automated tool flow that goes from DNN definition to embedded system implementation with FPGA accelerated inference and demonstrates the capabilities of the proposed framework in distributing the execution of these neural networks among a set of connected devices. We explore, through simulation, the design space of
computational capability and connectivity levels expected across different collaborative embedded system scenarios, such as drone swarms. We then use our flow to perform offload in a real multi-FPGA scenario. We show that, when provided with information about current network and ancillary compute availability, distributing the workload of a neural network can result in speedups up to the limit of available compute, providing the greatest benefits for networks that did not originally target mobile devices specifically.

**Power Aware Network Partitioning**

Achieving the independence of a sighted individual in a task like grocery shopping remains challenging because the visual assist system requires several simultaneous real-time functionalities, including localization and guidance within the store, product identification, and guidance to assist in the picking of desired product. Additionally, power consumption is a limiting factor in these assist devices because of their limited battery capacity. While computer vision approaches, especially with recent advances in DNN techniques for solving visual tasks [19], can address the accuracy requirements for these tasks, many computer vision algorithms are computationally demanding [20], limiting their full deployment on a single wearable visual assist device. Therefore, a common approach for mobile vision systems looking to maintain processing rates and preserve battery life is to partition computation between the mobile device and a more powerful back-end server at the cost of trading computation-amplification for communication overheads, as shown in Figure 3-1. In such a system the wearable is utilized as a camera input to the system as well as a hub for feedback such as audio or haptic feedback. Additionally, depending on the capability of the front-end device, tasks such as first level filtering and detected object tracking can be done. The tasks that need to be offloaded are the computationally demanding ones such as segmentation and classification.
The technologies that enable such a split, such as Wi-Fi, Bluetooth, or even cellular (4G LTE), can place substantial demands on a mobile device energy budget. However, communication costs in a mobile scenario are fundamentally dynamic. Factors from surrounding shelves to the wireless devices of other shoppers can alter the transmission environment in ways that degrade performance or efficiency: The energy associated with transmitting data is directly related to the power level of the transmitting radios and antennae, and mobile radios and protocols provide for signal strength adaptation to channel conditions, distance, etc. Broadly, the power level is set so that the communication channel will be expected to meet a minimum quality of service, usually defined in terms of bit error rate. As an example, in Figure 3-1, the two points indicated on the shopper's path may have very different transmission energy requirements for the same expected bit error rate and may in turn differ from the other locations along the indicated path.

Prior work [21] has shown that DNNs have some resilience to error for inference tasks, which is what computer vision systems primarily employ DNNs for in visual assist platforms. This represents an opportunity to dynamically re-optimize the QoS targets for communication channels in terms of acceptable bit error rate for the particular task that is being offloaded, rather than against absolute transmission error rates, thereby reducing power without sacrificing inference accuracy. Conversely, if a device is targeting a fixed power expenditure for transmission in an environment where channel quality is dynamic, there is another synergistic feature of neural networks: It is possible to train multiple different versions of the same network for a given task, with different noise assumptions applied during training, to best correlate the training and operating environments.
In this paper, we focus on the power consumption of the mobile component of a visual assist system with remote computation offload, specifically, reducing the power of offloading the image recognition to the cloud while maintaining inference quality. We introduce a technique to reduce the energy consumed by relaxing the requirement of communication bit error rates for sending layer activation records in DNNs. We show that when training a neural network used for image recognition with the noise well-matched to the operational environment, there are significant improvements in the robustness of classification accuracy over a network trained without noise. This improvement can allow the relaxing of the quality-of-service communication protocol enforcement leading to reduced power consumption of the transmitting device. We show Figure 3-1: High-Level System Overview.
that the robustness of DNNs can enable power savings techniques that can be applied to already existing devices.

Background

Image Recognition and Deep Neural Networks

DNNs are the state-or-the-art methods of machine learning and inference. For the ever-increasing popular task of image recognition, these networks have become quite large [19]. While there has been work for smaller quicker networks on a mobile device [3] and custom hardware starting to be developed for mobile platforms, many devices still lack the computational resources needed to run these state-of-the-art networks and require offloading the sensor data they gather.

Distributed Computing

As previously stated, the idea of distributing compute among many devices is not a new concept. In a push for internet-of-things (IoT) and many connected devices, distributed computing is emerging a popular area of research. Distributed computing allows many possibilities for connected devices including sensor fusion, power savings, communication reduction, and performance improvement.

Modes of Communication

Modern devices have a variety of way to communicate wirelessly with external devices. Among the most popular and widespread are Wi-Fi (802.11n or 802.11ac), Bluetooth (Classic
and Low Energy), and cellular communication (4G LTE). Each different mode of communication has a different energy-per-bit, range, and bandwidth. Universal to all these modes of communication is the exponential rate at which power consumption of these devices increase with channel noise, distance, and other conditions a radio will react to. While the energy per bit is different based on modulation scheme, energy per bit will scale proportional to the number of bits per symbol in each modulation scheme. However, universal to all the different schemes is the exponential nature of energy as the radio power increases. As can be seen in figure 3-2 binary phase-shift keying (BPSK) which transmits 1 bit per symbol uses 2X more energy than quadrature phase-shift keying (QPSK) which transmits 2 bits per symbol and reducing the transmit power from 30dB to 23dB reduces the energy per bit by 5X from 1.70x10⁻⁵ mJ and 8.50x10⁻⁶ mJ to 3.41x10⁻⁶ mJ and 1.70x10⁻⁶ mJ for BPSK and QPSK, respectively.

Figure 3-2: Energy-per-bit for varying levels of transmission power for Wi-Fi 802.11ac.
**Bit Error Rate**

Bit error rate is often referred to as a probability that a given bit will become in error during transmission over a communication medium. This is affected by many factors including distance, sensitivity of the receiver, and gain and power levels of transmitter [22]. Wireless methods are also more sensitive to environmental factors such as line-of-sight, signal reflection, and interference from other wireless devices [23]. Bit error rates are often characterized as a result of the signal-to-noise ratio (SNR) of a signal. How error prone a given signal is can depend heavily on the type of modulation and coding scheme (MCS) used. Figure 3-3 shows how as the SNR increases the bit error rate decreases for common MCSs used in wireless devices. As shown, modulation schemes that represent more bits per symbol transmitted such as 256-QAM which transmit 8 bits per symbol are more sensitive to a weaker SNR than 8DPSK which only transmits 3 bits per symbol. Like other works, this data was generated using MATLAB's BERTool toolbox [22] [24].

![Figure 3-3: Estimated bit error rates for common modulation techniques for a given signal to noise ratio generated using the MATLAB BERTool.](image)
**Bit error rate and classification accuracy**

To understand the impact of bit error rate on the classification accuracy of a neural network, we took two previously trained networks and inserted noise into an input image prior to inference.

We studied 2 networks, the Extraction network [3] trained on the ImageNet ILSVRC2012 [25] dataset, and a default TensorFlow [26] network trained on CIFAR-10 [27]. We used these models to represent a big network and small network. The extraction network contains 8 convolutional and 7 pooling layers followed by a final SoftMax layer. The TensorFlow [26] network consists of 2 convolutional layers, 2 pooling layers, 2 normalization layers, and 2 fully connected layers followed by a final SoftMax layer. Both were trained without any noise added although standard image processing techniques such as scaling were applied. When noise was added into the input layer of both networks, the noise was added into the raw pixel data which was 32-bit (RGBA). We observed similar drops in both networks. As shown in table 3-1, both networks dropped from their highest accuracy with no noise inserted to their lowest accuracy when the highest amount of noise was introduced. Due to limitations in our testing framework, the noise levels inserted in the larger network were slightly different from those in the smaller

<table>
<thead>
<tr>
<th>Noise Rate</th>
<th>ImageNet Top-1</th>
<th>CIFAR-10 Top-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Noise</td>
<td>70%</td>
<td>86.50%</td>
</tr>
<tr>
<td>Noise Level 1</td>
<td>59%</td>
<td>23.50%</td>
</tr>
<tr>
<td>Noise Level 2</td>
<td>46%</td>
<td>15.10%</td>
</tr>
<tr>
<td>Noise Level 3</td>
<td>4%</td>
<td>11.30%</td>
</tr>
</tbody>
</table>
network. Specifically, the larger network was using noise levels of 5%, 10%, and 25% while the smaller network was using 6.25%, 12.5%, and 25% for noise level 1, 2, and 3, respectively.

Intermediate Layers

Computations are not always computed fully on the cloud. Since the computation in a distributed system is often split in some way between the local device and remote, we also investigated the effect that noise had on the intermediate layers as well. Testing of noise in intermediate layers was done using only our CIFAR-10 network. Intermediate layers of neural networks are often real values and require some level of decimal precision. There were two ways we explored adding noise to the network. Figure 3-4 illustrates the two methods used to insert noise into the intermediate layers of our network. The first method was to quantize the intermediate value to either a 16-bit or 8-bit representation, introduce errors, and then dequantize. The second involved inserting the bit errors directly into the 32-bit floating point value. However, since the upper bits of a floating-point number (sign and exponent) can produce disproportional changes to the value, noise was only allowed to be inserted in the lower 23-bits of the number.
Table 3-2 shows the effect that bit error has when applied on the quantized intermediate value. As shown, as the layers are deeper into the network, the less tolerant to noise they are. For example, after the pooling1 layer, inserting 6.25% bit error rate drops the classification accuracy to 37.10% while the same bit error rate after pooling2 reduces the accuracy to 10%.

Figure 3-4: Illustrates the two methods used to insert noise into intermediate layers. Left: The quantization process. Right: Using the raw floating point.
When bit errors were introduced into the floating-point values as described we observed that the accuracy of the CIFAR-10 network did not change regardless of the layer or the amount of noise that was inserted. This result led us to explore how classification accuracy would be affected if the lower 23-bits were just set to '0'. This would emulate the lower 23-bits not being transmitted and filled in with set values on the receiving device resulting in a data size reduction of 72%. This method of float trimming was tested on four networks that were previous trained on the ImageNet ILSVRC2012 [25] dataset. How far into a network float trimming could be tolerated was evaluated as a function of depth into the network.

Since different networks may have different layer counts and types the evaluations was normalized to be percentage into the network. Figure 3-7 shows how for the Darknet network, 33% is located between the 3rd convolutional and pooling layer.

<table>
<thead>
<tr>
<th>Layer</th>
<th>0% (BER)</th>
<th>6.25%(BER)</th>
<th>12.5%(BER)</th>
<th>25%(BER)</th>
</tr>
</thead>
<tbody>
<tr>
<td>pool1</td>
<td>86.50</td>
<td>37.10</td>
<td>20.90</td>
<td>12.60</td>
</tr>
<tr>
<td>norm1</td>
<td>86.20</td>
<td>40.70</td>
<td>24.20</td>
<td>13.10</td>
</tr>
<tr>
<td>conv2</td>
<td>86.20</td>
<td>11.00</td>
<td>9.80</td>
<td>10.20</td>
</tr>
<tr>
<td>norm2</td>
<td>86.20</td>
<td>11.00</td>
<td>10.00</td>
<td>10.20</td>
</tr>
<tr>
<td>pool2</td>
<td>86.50</td>
<td>10.00</td>
<td>10.00</td>
<td>10.00</td>
</tr>
</tbody>
</table>

Table 3-2: The classification accuracy for a given bit error rate when bit error is introduced into the quantized value for each layer.
We observed that on these bigger networks depending on which layer the trimming occurs has significant effect on the classification accuracy of the network. Figure 3-6 shows how classification accuracy can vary as a result of trimming the intermediate floating-point values at various stages in a network. Specifically figure 3-6 shows how for some networks such as VGG-16 [2] the float trimming can be an effective means of data reduction up to approximately 70% into the network. A network such as [4] can only support this method in the first 15% of layers. Figure 3-6 shows the input data size to each of the networks to which the float trimming was applied. Universal to all the networks is that the layers with the largest input data size are also among the first layers in the given network. This tolerance to float trimming can be explained by the operations in the early layers of a network utilizing the IEEE-754 floating point format for its dynamic range which is determined by the number of bits in the exponent field rather than the precision capability which is determined by the number of mantissa bits. Looking at figure 3-6 and 3-7 together shows how the layers that can benefit the most from data reduction enabled by float trimming are also the layers which are most resilient to it making float trimming a feasible method for data transmission reduction.

Figure 3-5: Illustrates the layers of Darknet and cut points given as a percentage into the network.
Figure 3-6: Classification accuracy when input to layer is an IEEE-754 single precision float with the mantissa set to '0' from previous layer. Networks Used: VGG-16 [2], Darknet [1], Extraction [3], DenseNet201 [4].

Figure 3-7: Size of input to the layers for each network. Networks Used: VGG-16 [2], Darknet [1], Extraction [3], DenseNet201 [4].
Having characterized how different bit error rates can affect the accuracy of a network at different layers, we proceeded to explore whether or not a network could be trained to be more robust to them. Our CIFAR-10 network was then trained inserting bit error rate noise into different layers of network.

Figure 3-8 shows the accuracy improvement when noise is inserted into a network emulating a bit error rate of a transmission channel. As observed in figure 3-8 when a network is trained on 6.25% bit-error rate on the input layer the accuracy for 6.25% bit error rate increases from 23% to 78%. Additionally, accuracy under 12.5% bit-error rate also increases from 15% to 68%. Figure 3-8 also shows how intermediate layers can also benefit from training with bit error noise. For the same 6.25% bit-error rate the accuracy improves from 54% to 83%. Additionally, we observed that classification accuracy is highest when the network was trained with the amount of noise that was present during inference. We also found that for layers from Table 3-2 that
could not tolerate any noise were unable to trained with noise, thus training for noise is only effective for layers that are early in a network.

Ideas and Results

Having studied how bit error rates affect the accuracy of a deep network, we propose two methods to enhance accuracy in the presence of noise and to decrease radio power while having a minimal effect on overall classification accuracy. Figure 3-9 shows how these methods may be used from local device as well as the remote device.

Figure 3-9: Proposed methods to exploit channel noise.

Channel Condition Aware Device Power Optimization

When a client requires a certain accuracy in the classification, a device would be able to better make a decision whether or not to offload the data based on the available power budget. By allowing for reduced radio power and the bit errors that would be associated with it, communication cost could be lower for a given stage of a network, possibly allowing for the communication in the energy budget of the device, which would otherwise be too expensive with the default QoS requirements. When the communication cost is allowed in the energy budget the device may be able to improve the speed of its classification by offloading the compute to the
cloud. However, if the communication is not allowed within the power budget, the device would still have to finish the computation locally to guarantee a certain level of classification accuracy. The potential power saving using this method is illustrated in figure 3-10. As an example, if a device required an accuracy of at least 70%, the transmitting power could be set depending on channel noise. For this example, if the channel noise was less than 2.4dB the power could be set at 100mW. If the channel noise was between 2.4dB and 5.8dB the power level would be set at 317mW, and max power of 1W for noise above 5.8dB.

![Figure 3-10: Best radio power consumption to achieve a minimum level of classification accuracy for varying channel conditions](image)

Channel Condition Aware Server Accuracy Improvement

Rather than saving power by adjusting radio power, a device could transmit at a fixed power level but using a modified wireless protocol that allows for a greater level of bit errors.
This would reduce the number of retries as well as hold the radio power stable for the duration of the transmission as the radio would not have to adapt the power the current conditions. Instead, as to not sacrifice accuracy from the alternate transmission mode, the server being able to infer the channel condition as well would be able to pick the best network model to run for the input from that connection.

**Related Work**

Neurosurgeon [28] explored multiple methods of performance and energy saving in a distributed system running a DNN. In this prior work performance improvement of the DNN is improved through analyzing a given networks different layers and determining an optimal cut point based on the performance of the mobile device, performance of the offload device, and the size of the data transfer. Power was optimized only to the extent of transmitting a minimal amount of data and comparing to the time and energy of performing that layer’s computation. Other works have also explicitly explored the resilience in DNNs to bit errors [21].

The main focus of this prior work was to increase the performance of the network by allowing for an increase in bit error rate within a solid-state drive (SSD). Increasing the bit rate error in an SSD enabled either a lower overall system power consumption or a faster disk access speed. Allowing for bit error rates in data that were higher than are normally allowed in an SSD allowed the SSD to be run at a lower power. Additionally, running the SSD at a faster read and write speed by allowing for similar bit errors that occur during normal operation was an additional enablement.

**Automatic Partitioning**

Recent advances in deep neural networks (DNNs) have produced enormous gains, especially in the computer vision domain. While the push for ever-increasing accuracy has grown
these networks to dozens of layers and billions of operations, there is a simultaneous push to harness the power of DNNs on heavily constrained embedded and mobile platforms. This has led to approaches that trade accuracy for efficiency and has fueled the investigation of FPGAs as a means to directly embody these limited precision models while still being able to adapt to the rapid pace of DNN algorithm development. However, compared to the dominant approach for neural network inference in non-embedded domains, (e.g., CPUs, GPUs, ASICs), FPGAs currently exhibit substantial drawbacks in programmability, especially in terms of multi-device scenarios. Deep neural networks (DNNs) have become increasingly popular in recent years due to their ability to achieve high accuracy on complex, real-world problems. Increasingly capable DNN solutions, particularly in the domain of computer vision, have enabled new application spaces in robotics and automation, including paradigm shifting advances in autonomous cars and drones. The capabilities of DNNs come at the cost of billions of operations per inference, and current high-performing designs demand large quantities of memory, compute, and bandwidth resources. This limits the portability of state-of-the-art solutions to smaller, power-constrained platforms. While a common approach to enabling embedded inference is to offload portions of the work to the cloud [28], such solutions are not always viable for highly mobile or real-time constrained systems.

These limitations have spurred both the development of embedded-specific neural network frameworks [29] that trade precision for much greater efficiency and of hardware acceleration techniques utilizing FPGAs and ASICs [30] [31] to bring DNN requirements within the budgetary reach of smartphones, lightweight drones, and other endpoint embedded systems.
One promising approach for enabling DNN inference on endpoint devices, in environments where several such devices may be present, is to collectively pool the resources across these weaker platforms in order to tackle the higher compute requirements of DNNs. However, as shown in Figure 3-11 which shows completion time for each layer of AlexNet [5] for a variety of layer wise offload point and offload device counts with 14Gops compute per device and 300Mbps links, depending on the ratio of available communication and communication resources, it is not always beneficial to offload computation to other endpoint devices. Note that naively splitting computation between 8 devices under the given conditions is worse than running on a single device, whereas intelligently splitting the offloading to three
devices based on the given layer requirements could result in a 40% performance improvement over a single device.

While there have been successes in enabling smaller, low-power devices to be able to run state-of-the-art DNNs by moving from the typical, high-end GPUs used for both training and inference to more efficient platforms such as FGPAs and ASICs have shown promise [28], determining the optimal split points, especially in dynamic scenarios, such as drone swarms, is a burden for the system designer.

To address these challenges in enabling DNN inference on low-power embedded devices, this paper presents an automated tool flow for going from network definition to embedded system implementation that removes many of the programming and platform specific knowledge requirements that were previously an impediment to adoption. Additionally, this dissertation presents a framework for the distribution of these neural networks among a set of connected devices and develops a simulator to explore different distributions of computational power and connectivity to explore the performance of running a set of networks over sets of distributed devices. In the remainder of this paper, we present our tool flow and how it can be used to explore design spaces, describe a key use case for our approach, namely drone swarms, and evaluate projected speedups for four image classification DNNs under a variety of deployment conditions.
Neural Network Formats

A variety of neural network definition formats currently exist. These include TensorFlow [26], Caffe [32], Darknet [1], etc. Each of these formats define a network’s structure as a set of layers and number of maps.

Figure 3-12 shows an example for a common neural network definition file. There are three layers defined: convolutional, max pooling, and fully connected. Some information about a layer is implicit, and must be extracted from the surrounding layers, including layer input dimensions, kernel dimensions, and the output dimensions. Additionally, the number of operations within a given layer also needs to be extracted to fully understand its computational requirements. Connections are implicitly dense, at the layer specification level, with sparsity being provided by zeros in a specific, loaded weight matrix.

```
[ convolutional ]
filters = 256
size = 3
stride = 1
pad = 1
activation = relu

[ maxpool ]
size = 3
stride = 2
padding = 0

[ connected ]
output = 4096
activation = relu
```

Figure 3-12: Example of the Darknet Format.
Figure 3-13: Illustrates the flow of our DNN processing tool. It takes as input a given network format, creates an internal representation of the flow graph, outputs high level synthesis-valid C code, generates the scripts to synthesize the different needed cores, and the scripts to create the final bit stream for a target device. Below, we describe these in greater detail stages.
Pre-Processing

Figure 3-13 illustrates the flow of our DNN processing tool. It takes in a given network format, creates an internal representation of the flow graph, outputs high level synthesis-valid C code, generates the scripts to synthesize the different needed cores, and the scripts to create the final bit stream for a target device. Below, these stages are described in greater detail.

Flow Graph Generation

Table 3-3: Different types of nodes and their descriptions used for automatic DNN mapping.

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td>Used to represent external input to the flow graph</td>
</tr>
<tr>
<td><strong>Compute</strong></td>
<td>Compute nodes indicate nodes that have some sort of computation attached to them. Types of nodes include Convolution, Max Pooling, etc.</td>
</tr>
<tr>
<td><strong>Merge</strong></td>
<td>Since the output of many nodes’ connections are many-to-many. A merge node transforms these into many-to-one and one-to-many connections.</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>Represents nodes whose outputs become external</td>
</tr>
</tbody>
</table>

In order to abstract away the specifics of each of the multiple formats supported, we first translate into a common intermediate format, the *flow graph*. The flow graph represents a network of a specific set of limited node types and their connections to each other. Table 1 list the
types of nodes that a network operation can become mapped to for internal use. Although there are only 4 types listed for the purposes of generating a flow graph. Each type of computation gets its own type of node whose base is of type "Compute Node" (e.g., an 11x11x3 convolution node will get a different sub-type than a 5x5x3 convolution), but for the purposes of the tool, they are all considered to be compute nodes. Figure 3-15 also illustrates how each layer in AlexNet will get mapped to a different type of node. The input gets mapped to a node of type input, each operation in the convolution layer gets mapped into a computation node of sub-type convolution, and similarly for the max pooling layer. This mapping occurs after the initial parsing of the network definition and the loading of a corresponding weight file in order to properly annotate all of the nodes in the intermediate representation.

Flow Graph Optimization

The intermediate flow graph representation regularizes and makes explicit certain connectivity patterns, making subsequent passes simpler. For instance, one-to-one connections represent nodes where there is no external data sharing requirement, one-to-many connections represent broadcast opportunities, where as many-to-one can represent concatenation and combination. The generated flow graph goes through optimizations which ensures that every compute node type only gets its input from one node. In order to accomplish this, merge nodes (See Table 3-3) are inserted to replace the previous, implicitly dense inter-layer connections.

The merge node abstraction is particularly useful for identifying both data sharing and partitioning opportunities in the FPGA backend target. Modern FPGAs contain numerous block RAMs (BRAMs) embedded within the FPGA fabric. Each BRAM is limited in size, as is total BRAM capacity, but BRAMs collectively provide extremely high bandwidth and low access latency. This on-chip memory, when correctly utilized, offers a considerable performance benefit
over off-chip, external DRAM. In the flow graph, each merge node provides an explicit opportunity for the utilization of BRAM, rather than DRAM, as the source for broadcasting highly shared data inputs. When a merge node's size is within the resource availability of a given targeted device, our tool identifies this and assigns the location of that shared buffer to an instantiated set of BRAMs. If a given merge node is not able to fit into the on-chip BRAMs, the resultant data which is allocated into DRAM.

Since the library of cores which our tool creates utilizes AXI interfaces, whether or not a given core's data gets mapped to BRAM or DRAM can be assigned based on the addresses passed at runtime. In addition to identifying opportunities for BRAM use and data-reuse, merge nodes also identify natural places for distribution to external devices. Compute nodes which exist immediately following a merge node can be distributed more effectively utilizing the broadcast mechanisms discussed later in this paper.

**Custom Hardware Generation**

Once the flow graph is created and optimized as described above, our tool will automatically create a library of the specific, customized cores needed to run the input network. These cores are customized for the input network by taking the generic implementations of cores, e.g., a generic convolution engine requiring variables for parameters such as image height, image width, kernel height, stride, etc. and fixing these parameters as constants. Setting what otherwise would be variables as constants enables the high-level synthesis tools to perform more aggressive optimizations, such as loop unrolling and merging to create hardware that is usually both higher performing and with lower resource requirements.

Table 3-4 shows the result of generating computation-specific hardware blocks versus generic cores. While templating the code for each specific computational block is a relatively
clear path to optimization, current HLS tools do not support automatically performing this operation. As part of the HLS core generation the HLS tools used automatically produce a set of drivers which interact with the generated hardware cores via sets of control registers (e.g., Idle, Ready, Done, Start, etc.).

Table 3-4: Resource differences for cores generated from templated HLS code and generic code hardware units that support different AlexNet layers (16-bit quantized).

<table>
<thead>
<tr>
<th></th>
<th>DSP</th>
<th>FF</th>
<th>LUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Templated (Layer 1)</td>
<td>1</td>
<td>2003</td>
<td>2726</td>
</tr>
<tr>
<td>Templated (Layer 2)</td>
<td>1</td>
<td>1946</td>
<td>2777</td>
</tr>
<tr>
<td>Templated (Layer 3,4,5)</td>
<td>1</td>
<td>1923</td>
<td>2700</td>
</tr>
<tr>
<td>Generic (All Layers)</td>
<td>8</td>
<td>2721</td>
<td>3414</td>
</tr>
</tbody>
</table>

Our tool automatically creates function wrappers around these driver calls that present an interface similar to that of the original software implementation of the neural network.

**Static Computation Split**

Finding the optimal split of a network distributed among a set of connected compute devices depends on a variety of factors. These include the computational power of the initiating device (typically measured in operations per second), the bandwidth that connects the devices, and the computational power of the remote devices.

Figure 3-14 shows the formulas for statically determining the optimal split based on these factors. In typical image recognition DNNs, each layer is densely connected to the previous and next, so these formulas are applied on a per-layer basis.
\[ A = \text{Total Ops} \quad D = \text{Input Data Size} \]
\[ B = \text{Local Compute Power} \quad E = \text{Average Bandwidth} \]
\[ C = \text{Remote Compute Power} \quad F = \text{Total Output Size} \]

\[ \text{Optimal Split} \% = \frac{ACD - BCD}{AE(B + C)} + BCF \]

Figure 3-14: Formula for determining the optimal compute partitioning among two devices.

Figure 3-15 shows the formula applied for two different layers in AlexNet as communication bandwidth varies from 10 Mbps to 10 Gbps for available compute ranging from 1/2 the power the primary device (400Gops/2 = 200Gops) to 4X the power of the primary device (1600Gops). Figure 3-16 plots the same curves for just the first layer consisting of 96 11x11x3 convolutions. Other layers are not shown due to there being no offload benefit under the assumed conditions.

Figure 3-15: Optimal amount of compute to offload to a remote device (as a %) for a given available network bandwidth for varying levels of available compute for the first layer of AlexNet.
Figure 3-16: Optimal amount of compute to offload to a remote device (as a %) for a given available network bandwidth for varying levels of available compute for the first fully connected layer of AlexNet.
Extending to Dynamic Computation Split

Distributing a network in a dynamic environment can proceed along similar lines to the static splitting approach on a layer-by-layer basis but requires each device that could be used for offload be aware of the current load and connection to other devices.

Given the time constants involved, we expect dynamic splitting systems to make decisions that bind for all of the layers of a specific inference operation, with dynamic changes affecting subsequent inferences, rather than adjusting within a single distributed inference operation.

Figure 3-17: Illustration of applying the optimal compute formula recursively when multiple devices are available to use for computation.
Networking

All the deployment models assume that multiple devices share a common communication plane. Connecting the devices with the Transmission Control Protocol (TCP) ensures that the data transmitted will arrive without error. The User Datagram Protocol (UDP) is not guaranteed to be error free and is typically used in streaming type applications. Since neural networks have been shown to be error/fault tolerant to spurious data inputs and noisy data, we decided to use UDP for certain types of communication to make distributing the computation of a deep network feasible. One capability that UDP enables over TCP is the ability to easily broadcast data such that multiple targets/clients receive it, which is a common pattern between DNN layers. Figure 9 shows how the broadcast benefits of UDP over TCP increase as a function of participating devices. In addition to saving time, by only transmitting any given piece of data once the amount of energy used to distribute a computation is also reduced, which can help greatly in battery-limited embedded scenarios.

Distribution Framework

The distribution framework created is a library of functions that manage offload and communication decisions and abstract the communication network interfaces. The framework uses the generated function call wrappers described in Section Custom Hardware Generation and creates the job queues and networking calls that are used to share data. Functions within this library handle the configuration of shared data broadcast and automatic discovery of
available offload devices. Crucially on the networking side the shared data broadcast networking is configured as well as automatic discovery of available devices.

![Broadcast Time TCP vs. UDP](image)

**Figure 3-18:** Broadcast performance of TCP vs UDP.

**Approach**

To evaluate the speedup potential of distributed mappings for DNNs in a wide range of deployment scenarios, a simulator was created that allows us to set up an environment of any number of devices, each of which could have varying compute capability and network connection bandwidth and latency. The simulator allows us to explore the distribution potential of using broadcast mechanisms as well as varying the availability of nodes that are connected.

Figures 3-19,3-20,3-21,3-22 shows the how the potential for speedup varies as a function of the compute power available to offload to and the bandwidth that connects them. For example, in the simulation environment a compute node which was previously available for computation
may become unavailable or a node which was previously unavailable may become available dynamically during or between inference operations. After exploring viable points with simulation, the tool flow was validated by generating bitstreams and supporting software for running specific scenarios on real FPGA hardware.

**Drones/UAVs: A Prototypical Tool Flow Use Case**

Table 3-5: Mobile device performance in GFLOPs

<table>
<thead>
<tr>
<th>Device</th>
<th>GFLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM-A53 @1Ghz</td>
<td>8</td>
</tr>
<tr>
<td>Intel Max 10 10M50</td>
<td>72</td>
</tr>
<tr>
<td>Zynq UltraScale+ MPSoC XCZU3EG</td>
<td>180</td>
</tr>
<tr>
<td>NVIDIA-Tegra X1</td>
<td>200</td>
</tr>
</tbody>
</table>
Figure 3-19: Simulated speedup of AlexNet across various network bandwidths for 4 different device configurations.

Figure 3-20: Simulated speedup of Darknet across various network speeds for 4 different device configurations.
Figure 3-21: Simulated speedup of YoloV3 across various network speeds for 4 different device configurations.

Figure 3-22: Simulated speedup of VGG-16 across various network bandwidths for 4 different device configurations.
This flow, handling all of the steps from taking a DNN definition to a device to dynamically managing computation distribution, can be used for a variety of deployment scenarios. In this work, we use unmanned aerial vehicles as a prototypical application for exploring the utility of the tool. UAVs are becoming more ubiquitous and are able to handle a variety of tasks. These tasks include package delivery, emergency response, and aerial reconnaissance. Each of these activities require some variation of image processing, in addition the two other real-time control algorithms that control the flight of the UAV. Unmanned aerial vehicles and drones in general are platforms that can particularly benefit from this type of customized acceleration due to their aeronautical nature imposing physical size and mass limitations that restrict compute capability per device. Flight time is typically limited by the amount of energy storage (batteries) a drone has, and therefore a drone's computational ability is limited by both battery capacity and weight. Drones provide a unique characteristic for dynamism in offloading scenarios on account of being mobile and self-adjusting, meaning that it is possible for them to physically relocate themselves as the mission/task demands. Table 3-5 shows the compute power of the embedded devices typically used in these types of mobile platforms. Battery capacity and compute power are often at odds with each other for mobile device running time/design. This is even more crucial in UAVs, whose usefulness is often measured by the combination of flight time and capacity to carry additional equipment, sensors, cargo, etc. Such platforms also contain numerous communication mechanisms. All these system needs compete for the same battery energy, further constraining the resources available for DNN computation. Constrained by these limitations, and with the communication make drones are prime use cases for such distributed CNN's. This is especially true on-board devices such as drones/UAVs where the weight of the device limits both battery size, and available computational ability.
Figure 3-23: An example scenario of a drone swarm collaborating on finding a target.

Figure 3-24: Offload Patterns: A) No available offload; B) Primary device(A) offloading to device set(B) in Master-Slave pattern; C) Multiple initiators with collective sharing of idle resources.
This ability to relocate leads to a unique set of challenges and opportunities for distributing computation among different devices. Consider the scenario depicted in Figure 13 which shows a set of drones searching an area for an object of interest. Each drone's camera field of view (FOV) is indicated by the blue circle around it. Additionally, each drone could be equipped with a communication module. Each of these communication modules is assumed to have a distance that is beyond that of the camera FOV represented by the larger, red circle. With limited compute available, each drone would be first executing a general object detection task, typically a lighter weight algorithm than object recognition, to locate regions potentially containing the target. When a drone detects an object in its FOV it can then switch to the more computationally demanding recognition algorithm which is a more demanding task. The drone can then utilize the other nearby drones and their excess compute capacity to distribute the heavier computation in order to perform the recognition quicker.

As shown in Figure 15 a swarm of drones could be in various configurations when the object recognition task triggers. Configuration A represents a single device with a task and no ability to offload. Configuration B shows available devices in a master/slave pattern in which a primary device has a task to complete, and the additional compute devices are made fully available to the primary device without any of their own tasks to run. Such a situation could occur if the first drone to locate any preliminary evidence was allowed to commandeer all available resources for confirmation. Configuration C would occur if multiple concurrent recognition tasks are underway, with overlapping resource sharing among initiators. In the following section, we evaluate offload scenarios corresponding to all three configurations for a variety of DNNs. If one of the drones were able to locate what it believed was the target, it could communicate to the other drones for the additional compute (to possibly run a different more computationally demanding task) to verify. In such a situation the other nearby drones could stop their own tasks and only provide support to the primary.
Experimental Evaluation

The speedup potential of distributed mappings was evaluated for DNNs using two different approaches. First, the acceleration potential in various multi-drone systems running low-cost, resource constrained FPGAs and communication equipment was explored. Also, the tool's effectiveness was explored in a wide range of deployment scenarios using a simulator that allows us to set up an environment of any number of devices, each of which could have varying compute capability and network connection bandwidth and latency.

The drones in our networks are based on existing research platforms with embedded FPGAs and wireless connectivity. These platforms utilize a processing system based on the Zynq UltraScale+ ZU3EG [33] chip, which is utilized to collect additional platform sensor data in addition to FPGA CNN acceleration. The drones have a single-channel 802.11ac radio for inter-drone communication, which has a maximum throughput of 800 Mbit/s with no obstructions between drones.

The simulator is a complete distributed system simulator authored in C#. The simulator supports modeling the modeling of both compute and data transfers across a multitude of network connectivity's and compute capabilities. Additionally, the simulator can be configured to model energy usage for communication, compute, and movement.

Once a system is configured within the simulator, the flow-graph from the CNN generation aspect of the tool can be applied onto the model system and execution can be simulated. Using the formula described in the above section, the simulator will create a distribution of the workload and model its execution and dependencies. By sweeping parameters of the system, execution profiles can be created for many different scenarios. These simulated runs can then be logged as a lookup table on-board a device that it can use to find the appropriate static mapping for its own current situation. Each device is assigned an independent upload and
download bandwidth to enable simulation of heterogeneous systems and is capable of varying the data rates (or fully disconnecting a node) on-the-fly to simulate transient devices. Additionally, the virtual network switch is capable of broadcasting data to all devices on the network. The simulator gives us more freedom to explore the distribution potential of using broadcast mechanisms as well as varying the availability of nodes that are connected.

Figure 3-27 shows the how the potential for speedup varies as a function of the compute power available to offload to and the bandwidth that connects them. For these potential speedups, the simulator was configured to have all the peer-devices on the same network with each device having an equal amount of compute power (as measured in operations per seconds). The distribution schedule was created using the assumption that UDP would be able to be used as the network protocol and each device would be able to broadcast its partial results to all the other peers. The ability for a node to broadcast its partial results means that time and data can be saved by avoiding having to aggregate all the partial results on a dedicated node then redistribute. Instead, each node can perform its own aggregation of the data for its next layer computation assignment. Computation assignments are calculated in a way so that all the work (which includes compute and transmit) completes at approximately the same time while factoring in congestion on the network (i.e., no two devices will attempt to transmit simultaneously, instead a device will have more time to compute while another device is sending its results).

After exploring viable points with simulation, we validate our tool flow by generating bitstreams and supporting software for running specific scenarios on real FPGA hardware.

Results

We evaluate our tool on AlexNet [27], Darknet [1], and VGG [2], and YoloV3 [3]. These networks were chosen due to their common use and availability in the darknet format. Our
simulator models vary both the communication capability and compute capability of each node, as well as the number of nodes in the system. The simulator allows for the design space exploration where each compute element can handle a certain amount of computation. Additionally, the simulator allows for these compute units to be connected together using various types of emulated interconnects with varying degrees of performance.

For our evaluations, our simulator was configured to emulate a gigabit Ethernet connection which was available on the devices that would be used for on-board evaluation of our framework. We performed experiments with 0, 1, 3, and 7 additional devices available offload computation to in an environment where all compute devices have equal computational ability. Figure 3-6 shows the speedup results of running each of the chosen networks through our simulator for the different configurations.

Table 3-6: Simulated DNN speedup on distributed research drone platforms running Ultrascale+ ZU3CG FPGAs.

<table>
<thead>
<tr>
<th>DNN</th>
<th>1 Drone</th>
<th>2 Drone</th>
<th>4 Drones</th>
<th>8 Drones</th>
<th>16 Drones</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNET</td>
<td>86(ms)</td>
<td>1.98X</td>
<td>3.9X</td>
<td>7.7X</td>
<td>14.8X</td>
</tr>
<tr>
<td>Darknet</td>
<td>38(ms)</td>
<td>1.5X</td>
<td>2.4X</td>
<td>2.9X</td>
<td>3.5X</td>
</tr>
<tr>
<td>VGG</td>
<td>1600(ms)</td>
<td>1.9X</td>
<td>3.7X</td>
<td>6.7X</td>
<td>12.5X</td>
</tr>
<tr>
<td>MAVNET</td>
<td>82(ms)</td>
<td>1.4X</td>
<td>1.5X</td>
<td>1.9X</td>
<td>2.2X</td>
</tr>
</tbody>
</table>
Figure 3-25: Optimal computation distribution across 4 equally capable devices for a master/slave offload of AlexNet.

Figure 3-26: Optimal computation distribution across 4 equally capable devices for a master/master offload of AlexNet.
Our simulator results indicate that AlexNet benefits the most from having the computation distributed. Figure 3-26 shows the optimal computational split for AlexNet using the distribution equations from Figure 3-14 and Figure 3-17 for our simulated scenario using 4 devices. As shown in Figure 3-25 when communication is considered and there is not a master device (e.g., configuration C) as a bottleneck for result aggregation, more devices can be effectively utilized. For the Configuration B pattern, only the first layer benefits from distribution.

Figure 3-27: Simulator speedup results for three DNNs distributed over a varying number of network-connected devices.
Real Device Evaluation

Table 3-7: DNN execution time on 1-2 Zedboards compared with simulation.

<table>
<thead>
<tr>
<th>DNN</th>
<th>1 Device</th>
<th>2 Device</th>
<th>Speedup</th>
<th>Speedup (Sim.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNET</td>
<td>11.81(s)</td>
<td>5.53 (s)</td>
<td>2.1X</td>
<td>1.9X</td>
</tr>
<tr>
<td>Darknet</td>
<td>6.43(s)</td>
<td>4.96(s)</td>
<td>1.2X</td>
<td>1.8X</td>
</tr>
<tr>
<td>VGG</td>
<td>26.54(s)</td>
<td>20.57(s)</td>
<td>1.3X</td>
<td>1.9X</td>
</tr>
</tbody>
</table>

After demonstrating speedup potential in simulated environments, distributions for the four DNNs were created using our framework and deployed to Zedboards [34], embedded devices each containing a Zynq XC7Z020 [35]. Due to resource limitations in the network connectivity of these boards, only two devices were used for distribution.

Table 3-7 shows the results of speedup for each of the networks running on the Zedboards [34]. All DNNs showed speedups from distribution. As indicated by our simulator results, AlexNet was the network able to benefit the most from being distributed while the others had less improvement than predicted. This discrepancy in speedup between the simulator and the real device can arise from a number of factors. One factor which would limit the speedup is the memory bandwidth when running on the real device; when the required memory gets mapped onto the hardware the placement of placing data in BRAM versus DDR, as previously discussed, has a significant performance impact that is sensitive to final resource usage. Another factor is variation in the resource utilization of the given device. Actual DSP usage may fall short of predicted DSP due to routing and other constraints. Also, our simulator does not take into account precise DRAM timing parameters.
Notably, the single-device implementations do not yet match state-of-the-art performance levels. Initial investigations point to underprediction of the AXI interface resource usage as a key contributor, more than the DNN compute elements and memory. A remedy for this could be either moving to a larger FPGA or having each interconnect service many processing elements. However, the predicted scaling trends are still validated, and tuning of single-node performance for very small nodes is an ongoing aspect of tool development.

Related Work

Many approaches use FPGAs as DNN acceleration targets. These include tooling, custom device mapping, optimized computation blocks, and network offloading.

Works such as AngelEye [36] and Zhang, et al. [37] show how to design optimal computational units that make the most use out of FPGA resources and schedule CNNs onto a targeted FPGA. The networks are then deployed and statically scheduled among the number of cores they are able to place for a given chip. Additionally, existing tools, such as the Xilinx Deep Neural Network Development Kit (DNNDK), provide flows that go from common neural models to Xilinx FPGA optimized implementations. One limitation of this tool, common to current FPGA targeting frameworks, is that the cores that get deployed are pre-optimized hardware blocks that can support a variety of operations rather than being fully customized for the specific network's fine-grained constants, as with the library of cores emitted by our tool flow.

Separately, other works have explored the challenge of running DNNs on mobile and embedded devices through distribution and offload of computation. For example, Kang, et al [28] present latency optimizations for splitting DNN computation between a smartphone and cloud servers. It uses network conditions, the DNN model, and other factors to determine the optimal point in a DNN at which to offload remaining computation to the more powerful remote server.
Additionally, works targeting CNNs on cloud-scale infrastructure have been approached as in [38].

This work differs from prior efforts in two keyways. Firstly, our framework is implementation agnostic. When our tool reads in a network definition file, the cores that it generates are specific to that network, and the network does not need to be modified to run on multiple backend targets, e.g., by reworking the network in terms of the interfaces of fixed target blocks, such as a specific fixed width choice for operators rather than floating point values. An example of such modification would include things as changing a 32-bit floating point operation to 16-bit fixed. While 16-bit fixed point will show great performance and efficiency improvement over 32-bit float. This framework will simply generate a core using for the desired bit-width and format from the original, unmodified network specification.

This framework also differs in its targeted offloading strategy. It is designed to optimize the distribution of a DNN among peer embedded devices with heterogeneously available compute capability. This strategy differs from a one-shot offload to a powerful cloud server where communication time only has to be considered once, as large intermediate results are expected to stay on the device to which computation is offloaded. Contrastingly, in a collaborative computation among peers, there is a strong assumption of return traffic across partitions. This places different requirements on the ability and stability of the device that may be used for offload, as well as being less sensitive to rapidly changing network conditions.

Conclusion and Potential Use Cases

This dissertation presents both a tool for automatically creating hardware accelerators with supporting software from a given DNN definition file, and a framework for distributing the execution of a DNN over a set of distributed heterogeneous resources, thereby using automation
to ease programmability burdens and distribution to allow the continued use of individually inexpensive FPGA fabrics for acceleration. A simulation model can be developed to explore the design space of distribution options across various compute and communication availability scenarios. The simulated exploration is validated by using the automated tool flow to distribute computation to a real multi-FPGA system and demonstrate substantial speedups over a single-device solution. The results show that networks that have not been as heavily optimized for execution on mobile devices benefit more from such distribution schemes and we show that available communication bandwidth is the main limiting factor for distributed computation speedup.

Additionally, since DNNs are already tolerant to some level of noise during inference which can be exploited in distributed mobile-plus-offload execution environments. By training a deep network with additional noise this tolerance can be increased even further, allowing dynamic selection of the best model to use for the current expected error environment. This dissertation has shown that increased tolerance for bit errors can be traded for power efficiency in wireless communication which consumes a substantial portion of mobile platform energy and characterized how this tradeoff changes as a function of layer depth within common image recognition DNNs.

This proposed approach, enables both a lowering of the radio power to achieve a given accuracy, as well as managing the selection of the best-tuned neural network model to use under current channel conditions. This approach also has a secondary benefit of being able to reduce transmission retries that are often part of communication error handling protocols, thus improving latency as well. In general, we show that for a given channel condition it would be possible to reduce transmit power by enough to lower the SNR by approximately 7dB and only suffer an accuracy degradation of approximately 10%. For a noiseless channel, this power reduction would create an approximate power savings of 5X.
Assistive Technology Use Case

While the methods, techniques, and tools developed and presented in this dissertation have been developed under the use case of mobile UAVs. The application of optimized distributed compute can also be applied to any system or use case in which there are multiple compute enabled devices, each with the ability to communicate with one another.

In the use of assisting visually impaired persons, in the scenario of multiple individuals shopping in the same grocery store, each individual shopper would have their own assistive device. Each of these assistive devices could be compute and network enabled which can enable collaboration for tasks with the other peer shoppers. This scenario is depicted in figure 3-28.
which depicts multiple shoppers with assistive devices each looking from a different angle at a common shelf. In this scenario each device could assist the other devices with offloaded compute workloads as well as offloading its own compute. This type of sharing can also enable a sharing of information across devices to enable other shoppers to not have to recompute previously discovered objects and their locations.
Chapter 4

Conclusion

In conclusion, this dissertation successfully integrates user-feedback mechanisms with existing algorithms, being run on a mix of existing compute devices including GPUs and custom accelerator on FPGAs to enable a responsive visual assist system for visually impaired individuals to assist in the context of grocery shopping. In addition to refinement and optimization of vision algorithms for use in an assistive context for the task of grocery shopping, by leveraging both edge and cloud compute, as well the utilization of human-in-the loop algorithms, this dissertation provides key insights into both the user-facing and compute requirements that must be considered when making an such an assistive system.

Secondly, through the creation of an automated tool-flow and simulator that goes from network definition to FPGA implementation. This dissertation helps bridge the gap of programmability for FPGA when compared to CPUs or GPUs. Additionally, through a tightly coupled integration with the distributed system simulator this dissertation enables both the exploration of multi-device scenarios as well as providing insight into potential avenues from accelerating the running of these large neural networks on multiple distributed embedded devices.

Although significant progress has been made in the area for assisting the visually impaired, current systems still have numerous limitations. Such limitations exist as both functions of technology and the underlying research.

Algorithm Limitations

1. Vision based algorithms (SURF, SIFT, etc.) cannot be applied to all types of recognition.
2. Neural Network based vision algorithms have had great success at various aspects of computer vision including object detection, segmentation, and recognition.
   a. Networks require lots of training data to become reliably accurate.
      i. Training offline can help alleviate this issue however the ability to learn in deployment is limited.
   b. Computationally heavy, require ASICs to perform well at the edge (Google TPU, Apple Neural Engine, etc.).
   c. Memory bandwidth heavy which limits how much or how fast an algorithm can execute depending on its data requirement, including input, output, and intermediate.

3. As processors and ASICs improve the ability of neural networks will proliferate as a solution to vision assistance.

**Compute Limitations**

Technology limitations include resource limitations such as limited battery and compute power at the edge. This limitation of compute is further exacerbated by the form factor requirements that exist around on-body devices. Devices that are designed to be on body need to fit within a certain set of limitation such as size and weight. In addition to being small and light these devices are also more thermally constrained when compared to devices that are not meant to be worn on body. All of these limitations mean designers must choose between battery life which involves incorporating a bigger battery at the expense of compute power, and weight, to achieve a longer run time or sacrificing battery life and runtime to have more computational power.

As technology continues to advance however many of these the effect that each of these limitations have will diminish. As battery capacity and energy density improves the battery will
allow for more compute at the same run-time and weight. Likewise, as processors improve both architectorally and technology nodes (14nm, 7nm, 5nm, etc.) this will improve all aspects of the limiting factors reducing both power consumption and thermal requirements while also improving the computational ability of such devices.

**Network Limitations**

Despite vision algorithms being heavy users of compute resources, some of the issues around vision based assistive systems can be solved with offloaded compute even without improvements in compute technologies. This offloaded compute exposes the third limitation of such systems, connectivity.

To be an effective assistive system, computation/user feedback must be given within a certain latency. As explained above, failure to provide user instruction in timely matter can lead to errors and difficulties in the usability of such a system. Such an ability would be given as instruction to grab forward for an object after the user has already passed the intended object. With this limitation any type of offloaded computation would have strict deadlines of when results must return to the edge device. This time includes time to send, time to compute, and time to return. Since network connections have limited bandwidth and a minimum latency.

**Other Implications of Work**

Many of the challenges, solutions, and learnings of work done in this thesis maintain their relevancy and applicability in areas that go beyond assisting visually impaired persons within a grocery environment. In addition to the greater consumer/retail space, areas and technologies that
rely on computer vision such as automated manufacturing, warehouses, autonomous vehicles including, but not limited to, drones and cars can also apply the learnings.
Bibliography


Peter Arnold Zientara earned his Ph.D. in Computer Science and Engineering at The Pennsylvania State University. He joined the department in Fall 2014 as part of the Microsystems Design Laboratory advised by Dr. Vijaykrishnan Narayanan and Dr. John M. Sampson. His researched focused on the creation of a visual assist system designed to assist a visually impaired person with task of grocery shopping. As part of this, he also researched the acceleration of Deep Convolutional Neural Networks (DCNNs) across multiple distributed devices using FPGAs and other heterogenous compute devices which included the creating of a design automation tool to go from network definition to hardware implementation. He earned is B.S in Computer Engineering from York College of Pennsylvania in York, PA. In Spring of 2019 he started working in industry focusing on bring-up a next-generation embedded device for an augmented reality (AR) company.