


5-2018

Improving the Efficacy of Context-Aware Applications

Jon C. Hammer

University of Arkansas, Fayetteville

Follow this and additional works at: <https://scholarworks.uark.edu/etd>

 Part of the [Graphics and Human Computer Interfaces Commons](#), and the [OS and Networks Commons](#)

Recommended Citation

Hammer, Jon C., "Improving the Efficacy of Context-Aware Applications" (2018). *Theses and Dissertations*. 2703.
<https://scholarworks.uark.edu/etd/2703>

This Dissertation is brought to you for free and open access by ScholarWorks@UARK. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of ScholarWorks@UARK. For more information, please contact scholar@uark.edu, ccmiddle@uark.edu.

Improving the Efficacy of Context-Aware Applications

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Computer Science

by

Jon C. Hammer
University of Arkansas
Bachelor of Science in Computer Science, 2012
University of Arkansas
Master of Science in Computer Science, 2016

May 2018
University of Arkansas

This dissertation is approved for recommendation to the Graduate Council

Michael S. Gashler, Ph.D.
Dissertation Director

John Gauch, Ph.D.
Committee Member

Jingxian Wu, Ph.D.
Committee Member

Xintao Wu, Ph.D.
Committee Member

ABSTRACT

In this dissertation, we explore methods for enhancing the context-awareness capabilities of modern computers, including mobile devices, tablets, wearables, and traditional computers. Advancements include proposed methods for fusing information from multiple logical sensors, localizing nearby objects using depth sensors, and building models to better understand the content of 2D images.

First, we propose a system called Unagi, designed to incorporate multiple logical sensors into a single framework that allows context-aware application developers to easily test new ideas and create novel experiences. Unagi is responsible for collecting data, extracting features, and building personalized models for each individual user. We demonstrate the utility of the system with two applications: adaptive notification filtering and a network content prefetcher. We also thoroughly evaluate the system with respect to predictive accuracy, temporal delay, and power consumption.

Next, we discuss a set of techniques that can be used to accurately determine the location of objects near a user in 3D space using a mobile device equipped with both depth and inertial sensors. Using a novel chaining approach, we are able to locate objects farther away than the standard range of the depth sensor without compromising localization accuracy. Empirical testing shows our method is capable of localizing objects 30m from the user with an error of less than 10cm.

Finally, we demonstrate a set of techniques that allow a multi-layer perceptron (MLP) to learn resolution-invariant representations of 2D images, including the proposal of an MCMC-based technique to improve the selection of pixels for mini-batches used for training. We also show that a deep convolutional encoder could be trained to output a resolution-independent representation in constant time, and we discuss several potential applications of this research, including image resampling, image compression, and security.

ACKNOWLEDGEMENTS

I would like to thank my mother, Kelley, and my father, Robert, for their continued support throughout my educational career, as well as all of my friends, who have made the experience that much more enjoyable. I would also like to thank my advisors, Dr. Gashler and Dr. Yan, who have continually pushed me to exceed my own expectations, and my committee members, whose advice and experience has been vital in the success of this project. Finally, I would like to thank the University of Arkansas Computer Science and Computer Engineering department and the Graduate School for supporting my education and research.

TABLE OF CONTENTS

| | | |
|-------|---|----|
| 1 | Introduction | 1 |
| 1.1 | Context | 1 |
| 1.2 | Key Problems | 3 |
| 1.3 | Thesis Statement | 4 |
| 1.4 | Dissertation Overview | 4 |
| 2 | Background | 6 |
| 2.1 | Logical Sensor Fusion | 6 |
| 2.2 | Depth-based Positioning | 8 |
| 2.3 | Resolution-independent Image Representations | 9 |
| 3 | Unagi | 12 |
| 3.1 | Introduction | 12 |
| 3.2 | Related Work | 15 |
| 3.3 | Unagi Architecture | 17 |
| 3.4 | Energy-efficient Usage Analytics Framework | 18 |
| 3.5 | Leveraging OS Context to Enable Usage Pattern Inference | 20 |
| 3.5.1 | Automating usage pattern inference | 22 |
| 3.5.2 | Online learning and model updating | 25 |
| 3.6 | Case Studies using Unagi | 26 |
| 3.6.1 | Adaptive Notification Filtering | 26 |
| 3.6.2 | Network Content Prefetching | 29 |
| 3.7 | System Implementation | 30 |
| 3.8 | Evaluation | 30 |
| 3.8.1 | Cost of Virtual Sensing Framework | 30 |
| 3.8.2 | Cost of inference framework | 33 |
| 3.8.3 | Breakdown of the sensing and inference overhead | 38 |
| 3.8.4 | Performance of two driving applications | 39 |
| 3.9 | Conclusion and Discussions | 44 |
| 4 | Depth-based Positioning | 46 |
| 4.1 | Introduction | 46 |
| 4.2 | Background | 49 |
| 4.2.1 | Indoor Localization | 49 |
| 4.2.2 | Optical Imaging and Context Localization | 50 |
| 4.3 | Depth- and Inertial- based Localization | 50 |
| 4.3.1 | Depth Error Model | 51 |
| 4.3.2 | Rotational Error Model | 52 |
| 4.3.3 | Localization Error. | 54 |
| 4.3.4 | Depth Processing. | 55 |
| 4.4 | Creating a Localization Chain | 57 |
| 4.4.1 | One Hop Localization | 57 |
| 4.4.2 | Multi-hop Object Localization | 61 |

| | | |
|-------|---|-----|
| 4.5 | Statistical Optimization | 62 |
| 4.6 | Evaluation | 64 |
| 4.6.1 | Experimental Setup | 64 |
| 4.6.2 | Single Object Localization | 65 |
| 4.6.3 | Multi-hop Object Localization | 67 |
| 4.6.4 | Statistical Optimization | 69 |
| 4.7 | Discussion and Limitations | 70 |
| 4.8 | Conclusion | 72 |
| 4.9 | Future Work | 72 |
| 5 | Learning Resolution-independent Image Representations | 73 |
| 5.1 | Introduction | 73 |
| 5.2 | Related Works | 76 |
| 5.2.1 | Hypernetworks | 76 |
| 5.2.2 | Super-resolution | 77 |
| 5.2.3 | Compression | 77 |
| 5.3 | Learning a Representation for Single Images | 78 |
| 5.3.1 | Formulation | 78 |
| 5.3.2 | Error Metrics | 79 |
| 5.3.3 | Batching | 79 |
| 5.3.4 | Model | 79 |
| 5.3.5 | Training | 80 |
| 5.4 | Learning a Generic Encoder | 82 |
| 5.4.1 | Model Architecture | 83 |
| 5.4.2 | Training | 84 |
| 5.5 | Applications | 85 |
| 5.5.1 | Image Resizing | 85 |
| 5.5.2 | Compression | 86 |
| 5.5.3 | Security | 87 |
| 5.6 | Evaluation | 88 |
| 5.6.1 | Resolution-independent Image Encodings | 88 |
| 5.6.2 | General Image Encoder | 90 |
| 5.6.3 | Scaling | 91 |
| 5.6.4 | Encoding Size and Reconstruction Error | 91 |
| 5.7 | Conclusion | 92 |
| 6 | Summary | 93 |
| 6.1 | Final Thoughts | 95 |
| | Bibliography | 97 |
| A | Appendix | 105 |
| A.1 | List of Published Papers | 105 |
| A.2 | IRB Approval | 106 |

LIST OF TABLES

| | |
|--|----|
| Table 3.1: A sample list of virtual sensors used in Unagi | 20 |
| Table 3.2: Cumulative Confusion Matrix - Adaptive Notification Filtering | 42 |
| Table 3.3: Cumulative Confusion Matrix - Network Prefetching | 44 |
| Table 4.1: Common Notation | 51 |

LIST OF FIGURES

| | | |
|--------------|---|----|
| Figure 3.1: | Overview of the Unagi cognitive framework for mobile devices. | 18 |
| Figure 3.2: | Unagi’s virtual sensor manager. | 21 |
| Figure 3.3: | Architecture of the Unagi inference framework. | 23 |
| Figure 3.4: | Cumulative energy cost effect of multiple virtual sensors. Red squares represent active sensors which require explicit polling. Green circles are passive sensors, which do not require polling. | 31 |
| Figure 3.5: | A comparison of raw vs. processed data sizes for each user in the study. | 32 |
| Figure 3.6: | A comparison of the ratio of raw vs. processed data sizes for each user in the study. | 33 |
| Figure 3.7: | Amortization of Training Cost. | 34 |
| Figure 3.8: | Peak power usage compared to the number of training instances. Note both axes are in log-scale. | 35 |
| Figure 3.9: | Power consumption as a function of the number of instances (left) and the window size (right). | 36 |
| Figure 3.10: | Count of model updates as a function of the number of instances (left) and the window size (right). | 37 |
| Figure 3.11: | Time required to repeatedly rebuild or update a model. Y axis is in log-scale. | 38 |
| Figure 3.12: | Estimated Power Cost Per System Component. | 39 |
| Figure 3.13: | UI displayed to user for human labeling of training data instances. Used for the adaptive notification filtering example. | 41 |
| Figure 3.14: | Cross-validation accuracy over time for the adaptive notification filter. . | 42 |
| Figure 3.15: | Cross-validation accuracy over time for the network content prefetching service. | 44 |
| Figure 4.1: | An empirical model of the calibration curve for a Project Tango tablet. . | 52 |
| Figure 4.2: | Overview of the chaining method. | 55 |
| Figure 4.3: | Relationship between the parameters for a single hop. | 58 |
| Figure 4.4: | The laboratory setting of our experiments. Left: The UI of our Tango application. It provides a camera view for the user to point the tablet to the object to be positioned. Right: We use tripods to simulate users and objects in the scene. The user tripod (on the right) holds the Tango tablet, and others represent scene objects. | 66 |
| Figure 4.5: | Using more objects to localize a single point tends to result in less variance. | 67 |
| Figure 4.6: | Error of several 3-hop chains with different ranges. | 68 |
| Figure 4.7: | A long chain at long range. Note that the error does not monotonically increase between hops. | 69 |
| Figure 4.8: | Statistical optimization tends to improve performance when the distance between objects is large. | 71 |
| Figure 5.1: | Architecture of the encoder. We use convolution and max pooling to identify important image features and fully connected layers to learn the mapping to θ_f | 84 |

| | | |
|-------------|---|----|
| Figure 5.2: | A network has been trained on each individual image to generate a corresponding resolution-independent encoding. Images were then reconstructed at the original resolution. | 88 |
| Figure 5.3: | A comparison of three batching methods on two different images. Left: an arbitrary example from the MNIST dataset. Right: Lenna. | 89 |
| Figure 5.4: | A generic encoder was trained on 10,000 images from the MNIST dataset. Reconstructions were produced without any additional training. | 90 |
| Figure 5.5: | Various means of upscaling an element from the MNIST database. Images were increased in resolution from 28 x 28 pixels to 512 x 512 pixels. | 91 |
| Figure 5.6: | Larger topologies yield better image reconstructions. From top: an example from the MNIST dataset, Kodak #6, Lenna, the topology of \hat{f} using $\tanh()$ activations. | 92 |

1 Introduction

1.1 Context

For many years, an important goal for researchers has been to create devices, such as smartphones, tablets, and wearables, that are capable of observing and reacting to changes in the state of a user. As opposed to simpler devices that do not take the user's state into account, these enhanced devices offer the ability to realize many practical applications, including personalized content recommendation, personal fitness companions, and indoor navigation systems. Such aspirations can be realized by creating personalized models for each user, making use of both hardware and software sensors on the device to record important contextual information.

Traditionally, *Context* has referred to three aspects of an individual: what you are doing, where you are, and what (or who) is near you [1]. Context represents the "state" of a person and can be decomposed into three major facets corresponding to the three previous aspects of the definition. *Physical context* includes physical activities (e.g. sitting or working), *locational context* includes one's current physical location (e.g. GPS coordinates or a label, like "home"), and *proximate context* includes objects that are nearby and one's social setting (e.g. whether you are currently communicating with a colleague).

Devices that are *context-aware* are able to infer one or more facets of an individual's context, usually by analyzing data from one or more internal sensors. For example, activity recognition techniques are commonly used to infer a user's physical context. These techniques use information provided by inertial sensors such as the accelerometer or gyroscope to determine the physical activity of the user, which may include sitting, standing, walking, driving, or other similar activities [2]. Another example is determination of a user's *logical*

location, or a high level label, such as “home” or “work”, that describes the location of the user. These labels can be inferred based on sensory information from internal radios and temporal information (e.g. time of day or day of week) [3].

Recent trends in context-aware computing suggest traditional techniques for context-inference (e.g. processing information solely from internal sensors) may not be entirely sufficient for some problems. While inertial sensors are quite useful for activity recognition, for example, it is not immediately apparent that they would be helpful in inferring a person’s mood or state of mind. This suggests that alternative sources of information might be needed to satisfy the requirements of future context-aware applications.

Apart from the sources of information, the traditional methods for processing that data may also be unsuitable for novel applications. For example, while depth sensors can easily be applied to the problem of determining the relative location of a nearby object, novel techniques are needed to address their limited effective range in practice. New algorithms and techniques will have to be developed to address these problems in the future.

In this dissertation, our goal is to combine traditional context-sensing techniques with novel ones to expand the capabilities of context-aware systems. To do so, we look deeper into each of the three facets of context mentioned earlier (physical, locational, and proximate). We extend physical context by incorporating logical information about the user (e.g. how he is *feeling*) into the system model. Rather than simply localizing the user, we extend locational context to include the position of objects near the user. Lastly, we extend proximate context to include other information about nearby objects by explicitly building models to approximate scenes present in images. In the next section, we will discuss each of these advancements individually.

1.2 Key Problems

Physical context is defined by the physical activity of an individual user. Previous research has demonstrated that logical activity (for example whether or not the user is busy or stressed) can also be important for realizing context-aware applications [4], but the lack of development frameworks that focus on providing context sensing capabilities hinders the ability of application developers to take advantage of those features. The first problem we approach is that of incorporating multiple logical sensors into a framework that allows context-aware application developers to easily test new ideas and create novel experiences. We explore the costs associated with collecting relevant features about the user and with creating personalized models on the device itself to minimize latency and security concerns.

Localizing nearby objects is another important context-awareness problem, as such knowledge simplifies the process of developing novel applications that can observe and react to their environment. Developers are currently limited in their ability to take advantage of the locations of nearby objects due to comparatively high sensing costs. Highly accurate solutions (e.g. vision-based methods) tend to be prohibitively expensive in terms of power consumption, computational complexity, and sensor cost, while highly efficient solutions (e.g. RF-based methods) tend to be prohibitively inaccurate. The primary challenge in addressing this limitation is to create a solution that is both reasonably accurate and fairly inexpensive that can be applied to consumer-level products. We address this issue by exploring techniques to improve the accuracy of inexpensive depth sensors that can be embedded within certain mobile devices, such as Google's Project Tango tablet.

Proximal context-awareness requires an understanding on some level of the environment around an individual. Just as humans need eyes, ears, and other sensory organs to collect data about their environments, machines need hardware sensors such as cameras or

microphones to accomplish the same goal. While collecting that data is generally a straightforward task, processing it is more difficult. As a result, there are many open problems involved with the transformation from raw data to useful contextual information. For example, image processing applications typically work with the discrete representations of 2D images that are produced by digital cameras, but for certain tasks, a continuous resolution-independent representation could be a more natural approach. We focus on the problems of learning such representations using neural networks and applying those representations to important tasks, such as image resampling, compression, and security.

1.3 Thesis Statement

We claim that existing methods for inferring context are insufficient for the problems described in the previous section and that our contributions in this work will directly address those limitations, lowering barriers that prevent context-aware applications from being utilized to their fullest capacity.

1.4 Dissertation Overview

In this section, we summarize the remainder of the dissertation.

Chapter 2 provides the necessary background information about each of the three primary topics: combining logical sensors to enable novel application development, improving depth-based positioning, and learning resolution-independent image representations.

In Chapter 3, we discuss our work regarding the fusion of logical activity sensors in detail. At a high level, we propose a framework, Unagi, that collects information about the user and uses it to build an individualized model for some user-defined task. We use the motivating example of notification filtering on a mobile phone to demonstrate the utility of this framework in practice. We also present a thorough evaluation of the system in terms of

several metrics, including power consumption and model accuracy.

Chapter 4 focuses on our work regarding depth- and inertial-based positioning. We discuss DIPS, a novel indoor positioning method that uses depth and inertial sensor data to localize both mobile users and objects near them. Our primary contribution is a method to extend the effective range of the depth sensor in order to accurately localize objects farther away than the depth sensor would natively allow. As with our work with Unagi, we present evaluation results to demonstrate the effectiveness of the technique in practice.

In Chapter 5, we discuss our work in learning resolution-independent image representations by fitting a multi-layer perceptron (MLP) to the individual pixels of an image. We then demonstrate that such compressed image representations can be determined in constant time by training a deep convolutional encoder network to map from the original image to its representation. Finally, we demonstrate the utility of such representations by examining how they can be used for image resampling, image compression, and security.

Finally, in Chapter 6 we conclude our dissertation, summarizing each of the important contributions we have made with this work.

2 Background

In this chapter, we will provide more information about each of the three key problems that will be addressed by the dissertation: combining logical sensors to enable novel application development, improving depth-based positioning, and learning resolution-independent image representations. For each problem, we will provide background information that will be useful for motivating our work that is explained in more detail throughout the remainder of this dissertation.

2.1 Logical Sensor Fusion

Especially in the field of mobile computing, a significant amount of research has been done to create novel applications of the many hardware sensors already present in modern mobile devices [5, 6, 7, 8]. For example, accelerometers and gyroscopes have been applied to the problem of activity recognition – identifying the physical action being performed by the user, and GPS, Bluetooth, and Wifi sensors have all been used for localization – determining where the user is at any given point in time, especially if the user is inside a building where location cannot be measured directly. This research has directly led to a multitude of practical applications ranging from medicine to travel to gaming.

However, as most mobile devices are limited in terms of their energy budgets, making full use of hardware sensors for the purposes of physical context inference has proven to be a difficult challenge [9, 10]. This has motivated the search for alternative data sources, especially those for which data can be extracted in an energy-efficient manner whilst still being practically useful.

One such alternative is smartphone usage information, which has been shown to be both very inexpensive to collect and highly related to several important tasks. Communication history and application usage, for example, can be used for effective mood monitoring and prediction [11, 12]. Generally, these approaches collect a set of information from the device of the user and apply features derived from that information to address different problems.

We recently showed that these logical sensing techniques can be used to detect the feelings or emotions of users [4]. This problem is interesting because it explores a significant connection between users and their devices that has only recently begun to have been explored, and has several practical applications, such as a mood-sensitive music recommendation system. More formally, we defined several *logical statuses* of the user, including `isBusy`, `isStressed`, `isAlone`, and `isHappy`. We also demonstrated how each status can be inferred accurately by making use of information about applications, notifications, and the state of the screen on a mobile device.

In this dissertation, we step forward from that previous work to the problem of using logical sensing techniques to facilitate development of novel applications. We present Unagi, a system designed to allow developers easy access to various primitive logical sensors, such as those mentioned in the previous paragraph, as well as higher level constructs such as the `isHappy` logical status. Unagi extracts features from these raw data sources and builds personalized machine learning models for each individual user. It then correlates the features to personalized goals set by the user, performing all processing on the device for optimal latency and security.

2.2 Depth-based Positioning

Smart mobile devices, particularly wearables such as Microsoft HoloLens [13], Google Glass [14] and other similar systems [15, 16], have recently become popular, especially for highly context-aware applications such as Augmented Reality (AR). A core challenge of applications like these is determining the precise location of mobile users and the objects in their immediate vicinity.

Locating mobile users and ambient objects in their environment is an active research field in mobile computing. Odometry [17] is a classic method to realize simultaneous localization and mapping (SLAM) in a GPS-denied environment. Such systems integrate acceleration and rotation samples in order to estimate the location of the device. Inertial navigation systems can suffer from integration drift, however, where small errors accumulate to cause inaccurate localization results over time [18].

RF-based localization is one widely investigated approach that is well-suited to locating moving objects, as well as objects attached to RF transceivers in indoor environments. The primary idea of RF-based methods is to take advantage of the unique RF signal fingerprints at different places for localization purposes. Common RF-based methods include WiFi localization [19], Near-field communication [20], and Ultra-wideband methods [21]. Recent RF-based methods can locate an object within 1-2 meters [22]. RF-based methods have the immediate disadvantages of typically requiring an extensive infrastructure and relatively poor accuracy in comparison to other methods.

Image-based indoor localization is another well-studied field [23]. Stereo vision-based techniques [24] can compare a scene from two vantage points and extract 3D information from the objects in the scene. Stereo vision can produce very accurate results, but implementation on mobile platforms faces several practical challenges, including high computational and

energy costs [25].

A popular alternative to the other approaches is an infrared projector paired with a specialized camera tuned to the IR spectrum. These sensors (e.g.: OmniVision OV4682) typically possess regular RGB imaging capabilities as well as a depth sensor. Compared with pure CV-based 3D construction methods such as stereo vision, IR cameras are typically more accurate in distance measurement and are much more energy-efficient in practice [26, 27]. Industrial systems, such as AICON 3D Systems [28] can reach an accuracy of centimeter or even millimeter level but are difficult to integrate with mobile systems. Consumer-level platforms, like Microsoft’s Kinect [29] and Google’s Project Tango [30], tend to be more practical and are equipped with small form-factor IR optical sensors. Such devices have recently begun to show the tremendous potential of optical-based localization for mobile devices [31, 32].

In this dissertation, we wish to demonstrate that pure depth-based localization can be effective using relatively inexpensive (and inaccurate) consumer-level IR sensors, especially that found in Google’s Project Tango tablet. In particular, we present a thorough theoretical analysis of the problem of assigning a location to an object in 3D space using only (potentially noisy) depth information. Next, we discuss a set of techniques that can be used to localize objects farther away than the stated range of the device using a multi-hop principle. We further show that a customized optimization technique can be used to remove a significant amount of the human error involved with such a process, achieving localization errors less than 10cm in practice over a range of 30m.

2.3 Resolution-independent Image Representations

Inferring proximal context requires sensors capable of collecting data about the environment around them. Depth sensors, as discussed in the previous section, are one method

of doing so. However, while depth sensors can be very useful in localizing nearby objects, they are not as well suited to object recognition or modeling as other sensors such as traditional cameras. This is due to the fact that a depth sensor (even a highly accurate one) only provides information about the shape of an object, while traditional cameras can provide information about both shape and texture. The images produced by cameras then, are a valuable resource for a proximally context-aware application.

A significant amount of research has focused on the processing of images for different goals, including feature extraction [33, 34], shape detection [35], object recognition [36, 37, 38], style transfer [39, 40], partial image completion [41, 42], and complete image generation [43, 44, 45]. Many of these tasks will implicitly construct some model of the content of the image in order to function. For example, one means of accomplishing image completion is to fit a model to the pixels that are available (and perhaps to the pixels of several related images). The model would then be used to predict the values of the missing pixels.

A common method for addressing this problem is to identify a mathematical expression that represents some part of the image. For example, the Hough Transform [34] can be used to identify equations of lines and circles present in the image, and the Generalized Hough Transform [35] can be used to detect arbitrary shapes. Other methods can be used to represent the intensity values of the image instead of simply the shapes of objects in the scene. The Fourier [46] and Cosine Transforms [47], for example, are alternative approaches that represent the image as the summation of sinusoids of different frequencies and phases.

Of particular interest in this dissertation is an image model that is invariant to any particular sampling resolution. Such a model would allow a context-aware application more flexibility with its processing, since images could be resampled arbitrarily. In order to do so, we will fit some nonlinear function to the pixels of the original image. The parameters of that function will constitute a resolution-independent representation of the image. In our

work, we will use a multi-layer perceptron (MLP) as the mapping function, as they have been shown to be capable of approximating any arbitrary function [48].

In this dissertation, we will first propose a set of techniques that simplify the process of learning a resolution-independent image representation using an MLP. We will then show that it is possible to train a separate network (called an encoder) to map between the pixels of the original image and its resolution-independent representation. The encoder produces the weights of the mapping function as its output. As a result, resolution-independent representations can be generated in constant time, avoiding a lengthy optimization process at runtime. Next, we will discuss several potential applications of a resolution-independent image representation, including image resampling, compression, and security.

3 Unagi

Cognitive mobile applications have surged in recent years. As people spend more time on mobile devices, usage information has become a critical contextual source for them. Compared to hardware sensors, usage information is significantly less expensive to retrieve and process, and provides new contextual information dimensions. Despite such unique advantages, developing usage pattern-based cognitive mobile applications faces several daunting challenges, in particular managing the overwhelming amount of information stemming from various data sources and mining meaningful patterns to meet application goals. In this paper, we present Unagi, a novel data analytics and inference framework that aims to simplify the development of usage-based cognitive applications. Unagi provides high-level abstractions to a plethora of usage data sources and provides an energy-efficient runtime that supports efficient data acquisition and online pattern learning methods. We implement two novel cognitive mobile applications on top of Unagi: an adaptive notification filter and a network content prefetching module. We find that Unagi applications require two orders of magnitude less code than standalone counterparts, and the runtime incurs negligible system overhead for continuous usage, data acquisition, and pattern inference.

3.1 Introduction

Recent years have seen the rise of cognitive mobile systems [49]. As current smartphones and wearable devices have evolved to include sophisticated sensing and computation capabilities, future mobile devices are poised to become an intelligent platform, capable of understanding and adapting to the usage patterns of mobile users and providing personalized user experiences.

As mobile systems evolve to be more application-centric, usage data has become a noticeable source of user-specific, contextual data. Recent market surveys have reported that average mobile users not only spend significant amounts of time on smartphones on a daily basis, but more importantly, they have greatly broadened the scope of smartphone usage from traditional applications, such as telephony and games, to other territories such as mobile payments and airline check-ins. Compared to hardware sensors such as the GPS and microphone, usage information is significantly less expensive to retrieve and process. More importantly, usage information has a direct correlation to user preferences and behavior patterns. For example, recent studies have shown clear personalized patterns on how mobile applications are used [50]. Even amongst similar mobile applications, the way people use them can be dramatically different. Furthermore, usage patterns are closely correlated to the context of the mobile users, especially location, time, social, and even emotional factors.

Despite the unique advantages of usage information as a data source, developing usage pattern-based cognitive mobile applications faces daunting challenges. Mobile operating systems provide a massive amount of system usage information. However, it is impractical to leverage all of this information due to formidable energy and storage costs. It poses a significant challenge to application developers to choose an appropriate set of usage data sources and to extract and process data from them without consuming excessive system resources. In addition, in order to enable pattern inference, there must exist a universal interface to query these OS context sources. However, usage data sources are heterogeneous, and so no such interface currently exists. Moreover, practical cognitive applications typically require online pattern inference, and the cost of enabling such pattern inference on resource-constrained devices can be prohibitive.

This paper presents the Unagi (meaning “complete awareness”) framework to address the aforementioned challenges. The fundamental goal of Unagi is to enable fast prototyping

of cognitive mobile applications with low system overhead. To achieve this goal, we systematically investigate heterogeneous usage information from three major data sources: broadcast events, OS services, and OS logs. Unagi also contains a virtual sensor manager to enable universal access to these data sources with negligible system overhead and introduces a generic learning framework to enable online usage pattern inference with low system overhead. This learning framework includes a set of descriptive features, application-specific feature ranking and selection methods, and online learning and model updating schemes. Based on the sensing and inference frameworks, Unagi further introduces a set of programming APIs to allow fast prototyping of cognitive applications that exploit these usage statistics. To evaluate the performance of Unagi, two cognitive applications were created: a adaptive notification filter and a network content prefetching module.

The key contributions of this paper are as follows:

- We propose the design and implementation of Unagi, a cognitive framework based on usage information of mobile systems.
- We present a virtual sensing framework that transforms system usage information into a set of software sensors that are highly correlated to usage patterns and are accessible through a universal interface. Our evaluation results indicate that the continuous use of our virtual sensing framework incurs less than 1% battery life degradation, which is imperceivable to most mobile users.
- We design a generic learning framework that converts contextual information from the OS into descriptive features, automatically selects a customized subset of features for specific applications, and incorporates a lightweight pattern learner suitable for multiple applications at the same time. Our evaluation results on two sample applications indicate that our learning framework incurs negligible energy overhead to enable online

learning and model updating, achieving an average of more than 90% accuracy for both applications.

- We design a programming interface for fast prototyping of cognitive applications based on usage information. We show that the Unagi framework can significantly ease the difficulty of writing cognitive apps.

The rest of this paper is organized as follows. We first discuss cognitive mobile systems and the process of using system usage as a contextual data source in Section 3.2. We then present the overall architecture and the architecture of the sensing and inference components in Sections 3.3, 3.4, and 3.5, respectively. Two case study applications are introduced in Section 3.6. After that, we present the system implementation of Unagi in Section 3.7 and evaluate its performance in Section 3.8. We conclude this paper with a discussion of future work in Section 3.9.

3.2 Related Work

Cognitive phones, as coined by Campell et al in [49], emphasize the capability of understanding user behavior under different contexts and adapting the system accordingly. Decades of research has been conducted towards this vision by taking advantage of the rich set of onboard sensors within mobile devices. Examples include motion[2, 51, 52, 53], proximity[54, 55, 56], location[57, 58, 59, 60], and vision sensors[61, 62, 63]. Sensor-based approaches have two intrinsic limitations. First, continuous sensing incurs formidable energy and computation expenses. Recent research indicates that using motion, proximity, and location sensors can cause over 30% degradation of battery life[9, 64]. Complex signal processing methods, such as FFT[52], MFCC[65], and SIFT[63], incur further computational costs, which can degrade responsiveness[7]. Second and more importantly, hardware sensors

are effective for recognizing physical contexts such as activity and mobility patterns, but are not directly correlated to resource usage and user interaction patterns. For example, a user sitting alone in a coffee shop could be misclassified as “in a group” if only ambient noise were considered[55].

Mobile operating systems provide a large amount of usage information through data sources such as OS logs, services, and callbacks. This information is collectively referred to as the *OS context*, which contrasts the physical context obtained from sensors. There are two unique advantages of the OS context. First, as the usage of smartphones increases, so too does the salience of OS context sources because a rich set of system usage information is provided. Second, retrieving the OS context incurs much lower cost than retrieving the physical context through hardware sensors[66, 67]. In recent years, many cognitive mobile systems based on OS context have emerged. Examples include logical status inference[68], mood and stress inference[69, 11, 70, 71], app prediction[72, 73, 74], and many others.

Existing cognitive applications tend to follow a “chimney” methodology. In order to design a cognitive application, developers have to decide which OS context sources are useful for their applications and design both features to represent the OS context and learning algorithms to make sense of the contextual data. This process (data collection, feature design, and learner design) can be daunting for those developers, since the development of cognitive applications requires extensive knowledge of both the mobile OS context and machine learning techniques. We believe that much of this process can be automated to ease the difficulty of cognitive application development. In this paper, we argue that it is critical to design a generic architecture to fully unleash the potential of the OS context and to enable cognitive applications. In particular, we focus on the following three questions. First, how can we efficiently extract the OS context from the heterogeneous and unstructured data sources, including OS logs, services, and callbacks? Second, how can we infer system usage

patterns from the OS context, in particular for resource usage and user interactions? Third, how can we enable fast prototyping using the complete sensing and inference framework?

3.3 Unagi Architecture

To tackle the three challenges mentioned in the previous section, the following considerations were taken into account with the design of the Unagi framework.

- **Lightweight.** The framework must not incur excessive overhead, in terms of CPU, power use, or storage.
- **Flexible.** The framework should be generic enough for many heterogeneous applications.
- **Configurable.** The framework should greatly simplify the task of writing cognitive applications while still allowing complexity when it is desired.

An overview of the Unagi cognitive framework is shown in Figure 3.1. The framework consists of three layers. The sensing layer provides a set of virtual sensors that encapsulate a variety of OS context sources and a universal query interface to access these sensors. The inference layer provides a generic pattern learner that can infer a variety of system usage patterns from the virtual sensors. Data pre-processing and model updating approaches are also proposed to improve the performance of the pattern learner. The personalization layer provides interfaces for the design of cognitive applications, especially resource management applications, such as battery management, and personalized user interactions, such as app preloading. In addition, our framework also includes a set of cognitive APIs to allow third-party applications to access all three layers.

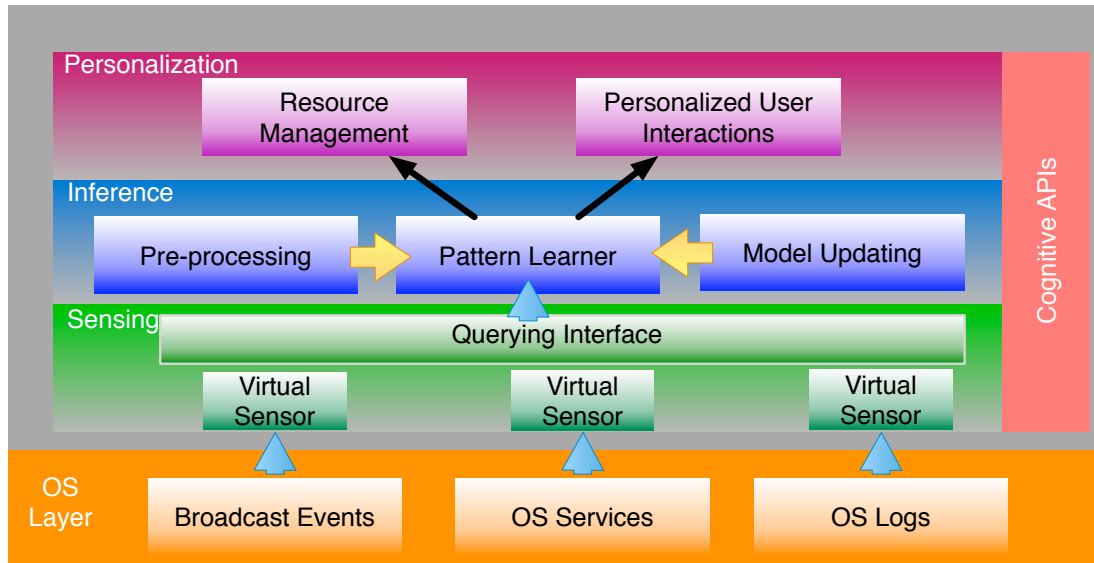


Figure 3.1: Overview of the Unagi cognitive framework for mobile devices.

3.4 Energy-efficient Usage Analytics Framework

The OS context includes complex information about how the system is used. However, the sources of OS context are usually heterogeneous. For example, the OS contexts of both Android and iOS can be retrieved from three primary data sources: OS logs, services, and broadcast event callbacks. There are dozens of such OS context sources, and the accessing interface for each varies significantly.

Initial OS context management approaches have largely focused on manually screening the context sources for specific applications. For example, Pathak et al. exploits a set of OS system calls to monitor the energy usage of smartphones[67]. Huang et al. uses the screen status for network traffic analysis[75]. AppScope uses kernel activities for application-level energy measurement[76]. Hammer et al. exploit application usage and user locations for logical status prediction[68]. Although these approaches provide significant insight on how useful the OS context can be, the challenges of managing the heterogeneous OS context sources still exist.

One goal of our framework is to simplify collection of usage information. Towards this goal, several design considerations need to be addressed. First, the framework has to include a diverse set of OS context sources that can be probed without modifying (e.g. rooting) the kernel or the device. Second, collection of this data has to incur negligible overhead when continuously running. Third, the framework has to provide a universal interface for 3rd party applications that wish to make use of the data sources we provide.

We first systematically study the problem of heterogeneous OS context sources. In Android, there are three major types of usage data sources, namely broadcast events, OS services, and OS logs. The means of retrieving usage data for each of these data sources varies significantly. Broadcast events are naturally suited to *passive* data collection methods, in which the data collector should only be active when an event is fired by the OS. OS services, however, are better suited to *active* data collection methods, in which periodic or ad-hoc polling is required to access the data sources. Passive data collection methods are effectively energy-free, but active polling does incur certain energy costs. Therefore, while our design does include several uniquely valuable active data sources, such as the current foreground application and CPU and network logs, passive ones are generally preferred. Table 3.1 lists several representative OS context sources found in our framework. A complete list contains more data sources.

Based on these data sources, we designed a virtual sensing manager to provide a universal query interface and handle the overhead of accessing them. The architecture of the virtual sensor manager is depicted in Figure 3.2. Inspired by Android’s SensorManager and the Funf Sensing Framework[77], our manager treats each usage data source as a “virtual sensor”, similar to that stated in previous work [78]. The virtual sensor manager allows clients to register for data from specific virtual sensors by specifying a set of key parameters, such as the associated OS context source, sensor type, and sampling rate (if necessary).

Table 3.1: A sample list of virtual sensors used in Unagi .

| Virtual Sensor | OS Context Sources | Type |
|----------------|--|-----------------------|
| Applications | Foreground app, Background services | Broadcast, OS Service |
| Notifications | Notification time, sender, and user responses (ignore or process) | Broadcast |
| Screen State | Screen lock and unlock, Passive or active unlock | Broadcast |
| Battery | Residual battery, battery level change, charging time, charging duration | Broadcast |
| WiFi | Connected BSSID and SSID, Connection status change | Broadcast |
| Bandwidth | TCP/UDP connections, Statistics of Tx/Rx bytes and packets | OS logs |
| Network | Network connection type and duration, Connection status change | Broadcast, OS logs |
| Downloads | time and name of downloaded apps | Broadcast |
| Processor | CPU usage, Processes, and wakelocks | Broadcast, OS logs |
| Memory | Memory usage, eviction statistics | OS logs |
| Storage | Storage usage, storage status changes | Broadcast |
| Package | App download, install, removal | Broadcast |

Configuration of the virtual sensors is performed using a descriptive language, JSON in our case, which also allows new sensors to be registered with our framework. We also designed a universal query interface that supports both ad-hoc and continuous queries. Note that our design also allows users to adopt the same paradigm used in accessing hardware sensors to our virtual sensors wherein OS context sources are treated as *black boxes* that return data, regardless of the type or access method.

3.5 Leveraging OS Context to Enable Usage Pattern Inference

Mobile users have immense diversity in using smartphones[50, 79], which suggests that usage patterns are personalized and user behaviors can be inferred from them. An

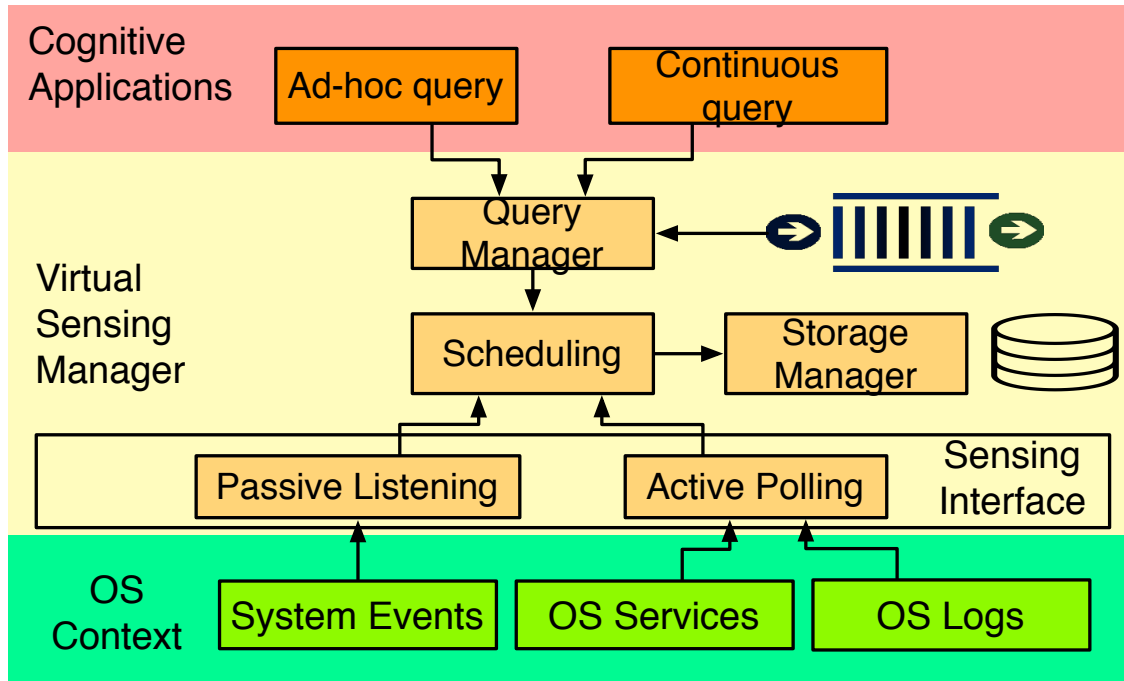


Figure 3.2: Unagi’s virtual sensor manager.

example is application usage prediction, which takes advantage of the fact that application usage behavior is tightly correlated to certain OS context sources, such as the temporal patterns of historical application usage[74] and recent notifications[68].

Although existing work provides useful insights on several individual smartphone usage patterns, designing a usage pattern learning algorithm is not straightforward. A typical design of such algorithms usually involves selecting OS context sources and designing features to represent them, as well as designing and experimenting with machine learning algorithms for accurate pattern inference. As future cognitive smartphones should be able to infer a set of usage patterns simultaneously, manually designing the inference algorithms for each cognitive goal cannot effectively scale and meet such requirements.

The goals of Unagi’s inference framework are two-fold. First, it aims to automate the inference process by designing a generic learning framework to support a variety of cognitive applications; second, it updates the inference model over time to enable online learning.

Based on the virtual sensing framework presented in previous section, it is feasible to access a large spectrum of the OS context and incorporate those sources into pattern learning. However, given a specific cognitive goal, we still need to tackle the following questions to automate the inference process: 1) how do we determine the right set of OS context sources for that goal? 2) What is the right set of features that should be extracted from the OS context sources? 3) How do we design a generic learner to fit different cognitive goals?, and 4) how do we ensure low overhead in the inference process? Besides automating the learning process, it is also critical that the pattern learner can evolve over time, enabling online pattern learning. The rest of this section presents the design of Unagi 's inference framework that addresses these two challenges.

3.5.1 Automating usage pattern inference

Unagi's inference framework divides the inference process into four steps, as shown in Figure 3.3. The first step is to discretize the raw data from the sensing framework in order form coherent bundles of data. This data is fed into a set of feature extractors, which collectively produce a single instance vector. That vector is added to a training data set, along with a training label provided by the client, and an internal learning model is updated with the new information. After enough training data instances have been generated, a feature selection process is applied to reduce the size of the feature space, effectively disabling part of the framework for efficiency.

Discretization

Since all of the sensed data that is sent to Unagi's inference framework is temporal in nature, discretization is important in order to segment it. One approach is to use fixed width time windows as a segment. In this case, the resulting instance vector represents the state

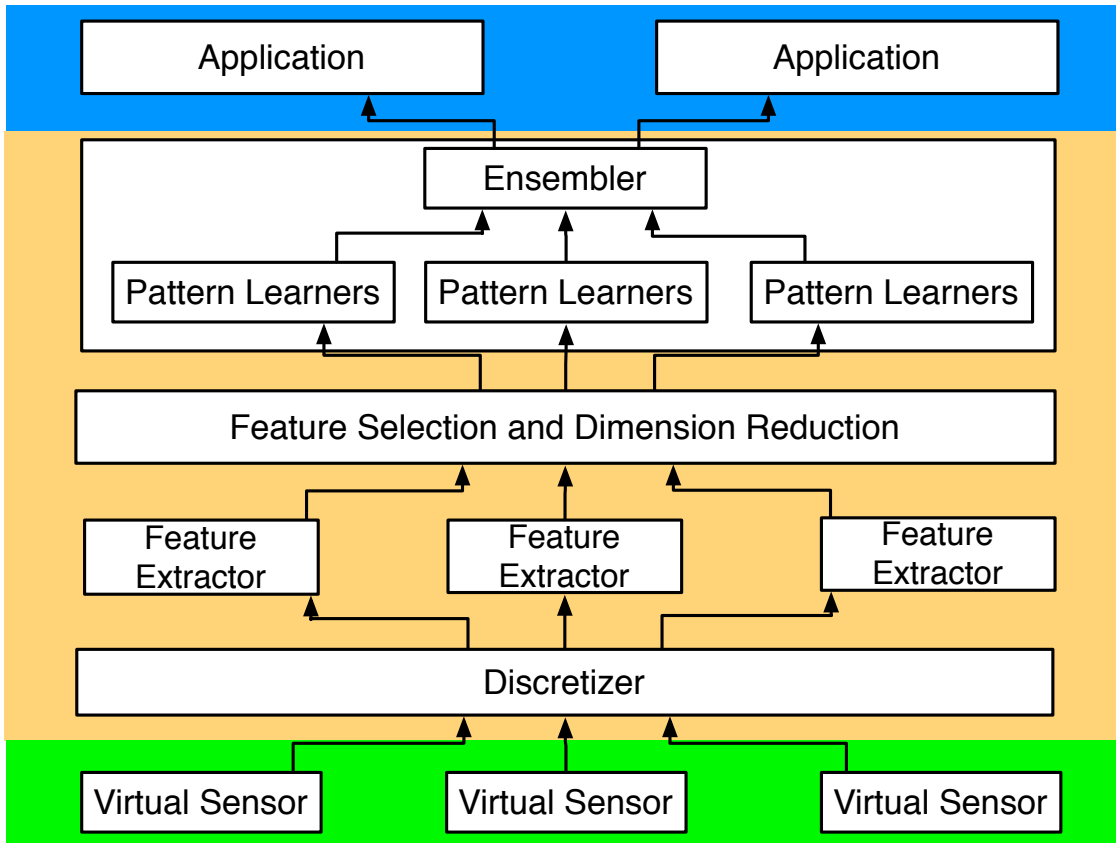


Figure 3.3: Architecture of the Unagi inference framework.

of the system during that time window. Another approach is event-driven discretization. In this case, most pieces of information will be pooled as they arrive. When certain data from a specific source arrives, the entire pool is sent to the feature extractors as one coherent unit. An example of this method is a notification arrival. In such a circumstance, each instance of training data will correspond to one arrival event. This method is used by our adaptive notification filter application, which is explained in more detail below. Since discretization is primarily a pooling technique, it tends to incur very little power consumption on average.

Feature extraction

Extracting informative features from the OS context sensors is a vital step towards usage pattern inference. One benefit to making use of these sources is that they are in-

trinsically more highly correlated to some individual usage patterns than hardware sensors. Another is that they tend to require less processing in order to extract relevant features, since expensive computations (e.g. FFT) are usually unnecessary. Several feature categories that we make use of in our framework include statistics calculated from the raw data, current system states (e.g. screen ON/OFF, SSID, foreground app), and historical aggregations (e.g. average notification response time), which have been shown to be useful in previous works[80, 72, 56, 74]. All of these require very little additional processing to extract, so our feature extractors can initially generate a complete feature table for each application by exhaustively generating all potential features for all OS context data sources. In order to optimize this process, each feature that is required by any application will only be extracted one time. Those features that are used by multiple applications will be shared to reduce unnecessary computation. Feature selection, the next stage in the process, provides another optimization.

Feature selection

The primary goal of feature selection is to select the most relevant subset of features for each specific cognitive goal. Feature selection techniques have been widely studied in large-scale machine learning problems for improved accuracy and decreased training time[81, 82]. Another benefit is that they allow the framework to disable those features that are not required for any of the inference goals, considerably reducing the feature extraction cost.

Feature selection occurs exactly once in most cases (during the transition between the training and prediction phases), so the cost of the reduction is inconsequential. The choice that remains is which technique to choose. Two common approaches include greedy attribute selection[83], based on information gain, and Principal Component Analysis (PCA). Given that our goal is to remove certain features from the superset completely, greedy attribute

selection is a better choice as PCA requires that all features be calculated so they may then be transformed into lower-dimensional space.

Generic learning

Conceptually, an ideal learner for general problems would be an ensemble of multiple weak learners where the model that performs the best is chosen from the ensemble to handle whichever problem it has been presented with. Unfortunately, this technique quickly becomes infeasible due to the linear increase in cost with the number of weak learners. Instead, several common weak learners were evaluated in isolation, such as J48 Decision Trees, K-NN, Naive Bayes, and Random Forests. Of those tested, decision trees tended to provide the best tradeoff between cost and accuracy and so were chosen as the default model for Unagi's inference framework. Several other models are supported, however, so the client can adjust results based on their unique application needs.

3.5.2 Online learning and model updating

After generating the generic learner, Unagi's inference framework also needs to update the inference model over time to support online learning. Model updating is critical to the practicability of our inference framework, as many user behaviors evolve over time, new patterns emerge and old ones become obsolete. However, model updating also faces the cost and performance tradeoff. Updating the model too often could incur high energy and computational costs, while updating the model infrequently might cause inaccurate behavior pattern learning results. There are at least two potential methods to mitigate the liability that comes from excessive updates: 1) reduce the frequency of model updates, and 2) substitute a learning model that is capable of updating incrementally. These two approaches have the added advantage of being mutually orthogonal to one another. Either can be used

in isolation, or the two can be used in combination.

In order to reduce the update frequency, there are two primary methods: instance-based and time-based approaches. Using the instance-based approach, the model is updated every K instances, rather than every instance. Using the time-based approach, the model is updated every t seconds instead. Both approaches are evaluated later in the paper.

The models produced by certain learning algorithms, such as artificial neural networks, are readily amenable to online training, wherein the model is updated after each instance incrementally. Other algorithms, such as decision trees, are not. When a model is used that does support online training, that model can be updated incrementally, which can allow for potentially drastic performance increases. Models that do not have this property must be completely rebuilt when an update is desired. Rebuilding costs, as we will show later, increase linearly with the size of the training data, and so “offline” models are not ideal for large-scale problems, even though they may still be suitable for smaller ones.

3.6 Case Studies using Unagi

In order to demonstrate the usefulness of Unagi , we present two concrete applications of the framework: one for adaptive notification filtering, the other for network content prefetching. We chose these two applications not only because they are useful for mobile users, but also due to the fact that they represent two different scenarios of using our framework, with explicit and implicit labels respectively. We present the detailed setting of these two applications, and evaluate the performance of our framework for them in Section 3.8.

3.6.1 Adaptive Notification Filtering

Individual applications often communicate with their users through status bar notifications. These are typically used to inform the user about some asynchronous events, such

as the arrival of an email or text message. However, such notifications could be distracting to the user. Some notifications are simply spam; some other notifications, even those that are not spam, may be perceived as distracting when issued at an inappropriate time. Most mobile operating systems, such as Android and iOS, allow users to block notifications from a given application to combat this problem. Some, such as newer versions of Android, allow notifications to be filtered so that the user is only interrupted when a priority notification arrives. These solutions are helpful, but do not currently provide enough granularity. Emails, for example, might need to be filtered only for junk mail, but not for co-workers. Messages from family members might be distracting when sent when the user is working. The burden falls on the application developer to decide which notifications should be prioritized.

One possible improvement is to develop an adaptive notification filter. This filter would monitor the user's behavior and learn how to differentiate between those notifications that should and those that should not be filtered out. Unagi could, in theory, remove much of the complexity of this task. Rather than developing a custom system to collect data and apply machine learning in order to decide when a notification should and should not be filtered, an app developer needs only to interface with the framework and use the predictions that were made as a decision engine in their own app. The basic process is outlined in the pseudocode given in Algorithm 1

In this case, the application interacts with Unagi using an "event-driven" approach. The application first connects to the Unagi service and registers a desired Action. Note that in this case, it is a supervised learning problem, so an Action includes a source of training labels. For this particular problem, it is reasonable to simply ask the user for a label (e.g. via a notification of our own) of if a notification should be filtered in the future. A training instance contains information about one notification arrival, such as current system values, historical responses to similar notifications, and anything that can be mined from the

Algorithm 1 Adaptive Notification Filter

```
function INIT()
    Action a = new Action()
    a.useEventDrivenDiscretization(Notification.Arrival)
    a.setTrainingLabelSource(LabelSource)
    a.setCallback(CallbackFunc)

    UnagiService.registerAction(a)
end function

function CALLBACKFUNC(boolean filter)
    Perform filtering based on 'filter'
end function

function LABELSOURCE(InstanceVector)
    Issue notification to ask user of label for 'InstanceVector'
    return label provided by user
end function
```

notification itself (e.g. the sender). When Unagi decides enough training data is available, it begins to make predictions and notifies the application client every time a prediction is made. In other words, the application client will begin to receive guidance from Unagi . The filter can then act according to those predictions that were made by actually performing the filter operation.

The entire adaptive notification filter application is implemented in Algorithm 1. As shown in this implementation, the application developer no longer need to spend time on collecting and making sense of the OS context. Instead, the virtual sensing and generic learning modules of Unagi take the objective of the application (given by the training label specified by the application) and automatically generate a learning model that filters notifications based on the OS context sources.

3.6.2 Network Content Prefetching

An obvious drawback of the previous application is the explicit requirement that training labels must be collected from users, which can be impractical for many cases. In this case study, we explore how Unagi's cognitive abilities can be used to generate rule-based training labels and then use those implicit labels in the inference process. We use network content prefetching as an example.

Our network content prefetching module tackles a simple version of the app prefetching problem. We assume that there are two types of network connections, say WiFi and Cellular networks, with different associated costs. In particular, WiFi has a strictly lower cost per bit than cellular. Under this assumption, when a user switches from WiFi to cellular, the application being used during the transition (and the ones right after the transition) can all benefit from prefetching the network content when still on WiFi. The key to our app prefetching problem is to predict when a WiFi to cellular switch will occur based on current and previous OS context data and if the current application should prefetch network content.

Our framework can handle this task as well, though some minor modifications need to be made. For example, event-driven discretization is not necessary anymore. Instead, a frame-based approach is more reasonable. One instance is now a fixed-width time frame, containing information about the state of the system during that frame (e.g. WiFi connected/disconnected, SSID of connected network, charging/not charging). Another difference is that the training data labels in this case need not come from the user of the device. In this case, a set of rules based on certain features might be more appropriate. For example, when the WiFi state of one frame is connected and the WiFi state of the next frame is disconnected, network data should be preloaded. In this manner, Unagi essentially operates autonomously, requiring no interaction with the device owner and communicating only with

the prefetching service.

3.7 System Implementation

Our framework was implemented for unmodified Android devices as an application library that can be included with individual client applications. Our implementation consists over 12K lines of Java code, built on a highly modified version of the Funf sensing framework [77]. Unagi runs as a system service that handles two major tasks: 1) sensing and 2) inference. In the sensing component, 64 virtual sensors were implemented, covering a large spectrum of OS context sources, from CPU usage to battery status. The virtual sensors are responsible for communicating with the various data sources currently in use and transporting the sensed data back to the core system service. The inference module supports two types of feature extractors: event-driven features and frame-based features. The inference model also supports 9 weak learners, including Decision Trees, K-NN, and K-means.

3.8 Evaluation

To evaluate Unagi , each component of the framework will be examined in more detail. The system as a whole will then be evaluated by examining two driving applications, one for adaptive notification filtering, the other for network content prefetching. All system performance experiments were performed on Nexus 4 devices running Android 4.4.

3.8.1 Cost of Virtual Sensing Framework

Energy cost of virtual sensors

In order to evaluate the sensing cost, the rest of the framework was disabled, and a single virtual sensor was activated. After a fixed amount of time, the power consumption was measured, and another sensor was added. This process was repeated to include each of

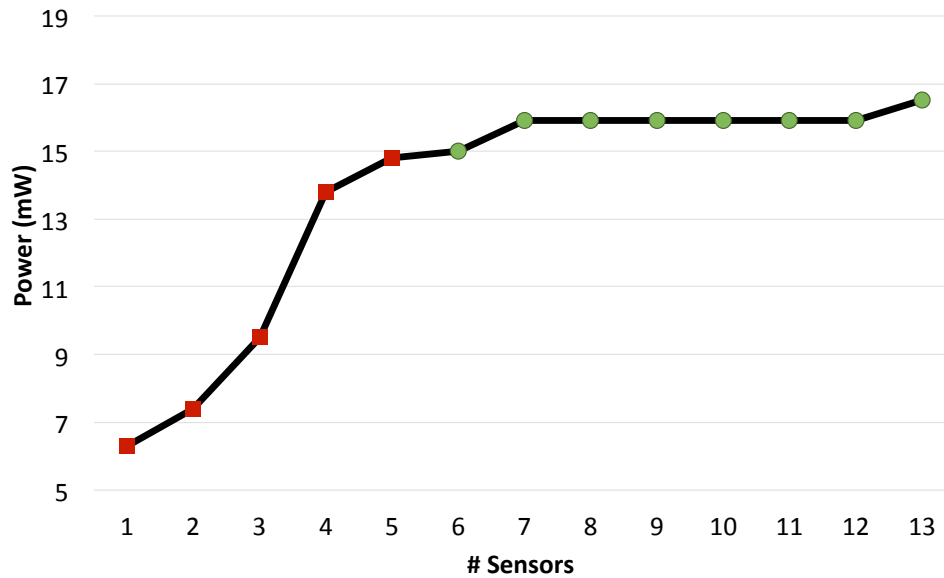


Figure 3.4: Cumulative energy cost effect of multiple virtual sensors. Red squares represent active sensors which require explicit polling. Green circles are passive sensors, which do not require polling.

the 13 sensors Unagi provides by default. Figure 3.4 demonstrates the cumulative impact of these sensors on the net sensing cost. The first five sensors, (as shown in the red squares), are active, meaning they incur a non-negligible cost due to polling. In particular, the first two are OS service-based, meaning they query some OS service to collect data, while the next three require file system accesses, and so are more expensive. The green circles represent passive sensors that collect data as a response to some OS-level event. These incur almost no additional overhead beyond what the OS is already doing, so data can be collected from many of them simultaneously. As this figure indicates, Unagi scales well with passive sensors, but the number of active sensors should be restricted to avoid exorbitant sensing costs.

Storage cost of virtual sensors

Because of the online nature of the framework, only a small fraction of the total information sensed will ever need to be written to storage on the device. Much of the

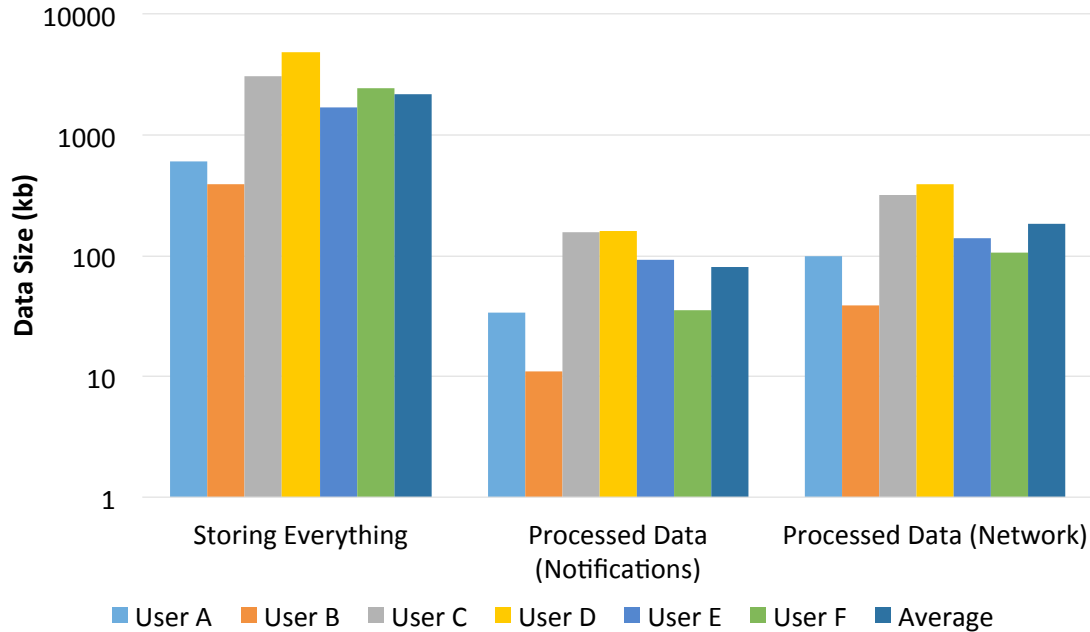


Figure 3.5: A comparison of raw vs. processed data sizes for each user in the study.

raw data is converted into a useful form by the feature extraction process, and only those processed instances will be persisted. To investigate this further, we conducted an in-situ study involving six mobile users in which they collected data for two test applications, adaptive notification filter and network content prefetching. During this experiment, those users used Unagi on their devices, and both their raw data (which was shared between the two applications) and processed data were written to files (each including some meta-data). The results, shown in Figure 3.5, were sizable. Note that the y-axis is in log scale. The raw data files were typically 1-2 orders of magnitude larger than the processed data in either applications. As seen from Figure 3.6, the average improvement was about 12x for the network application and nearly 27x for the notification application. Although the processed files are relatively small already (an average of 183kb for the network example over the course of one week), they could be shrunk further by using a binary encoding, rather than a plain text one, and by using compression techniques.

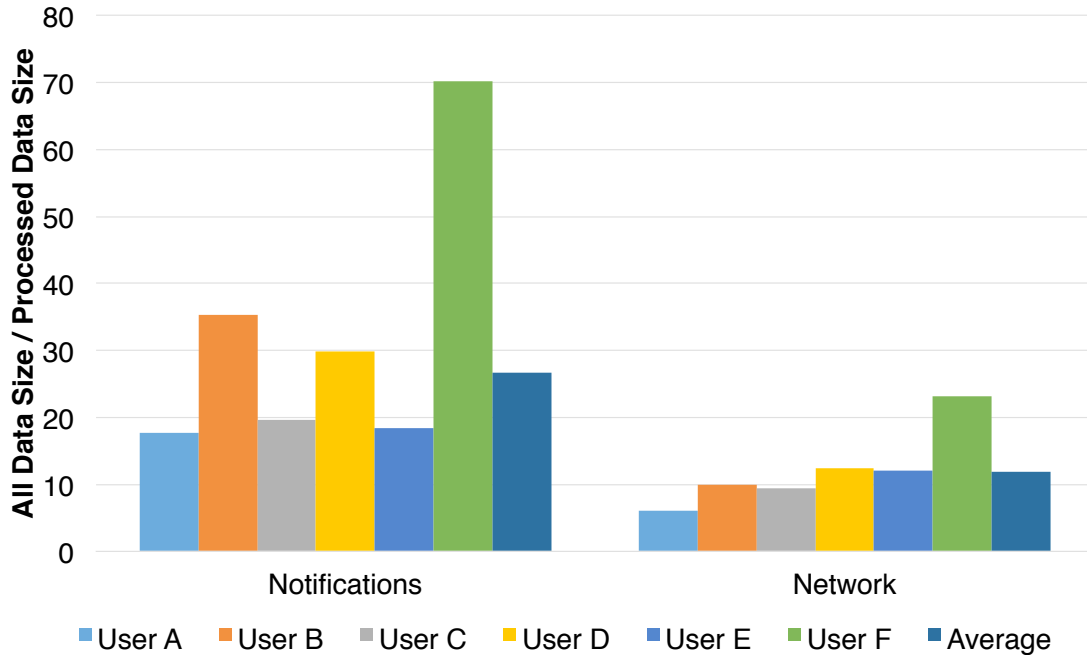


Figure 3.6: A comparison of the ratio of raw vs. processed data sizes for each user in the study.

3.8.2 Cost of inference framework

Amortizing Inference Cost

The cost of performing inference on the device could potentially be prohibitive for a mobile system, but several factors help to mitigate those expenses. Firstly, because the training and prediction phases are separate, and prediction is typically much less expensive than training, what cost there is is born primarily by the training phase. Since time spent in the prediction phase will likely dominate that of the time spent in the training phase, the lower cost of the prediction phase will also likely dominate over time. Secondly, as shown in Figure 3.7, the training cost is divided between long idle periods (which incur almost no cost) and short spikes that occur when a new instance arrives and the model is updated. Because these spikes are infrequent, the amortized cost is much closer to the idle value than the peak value. For the case when the given model can be updated online, these peak values

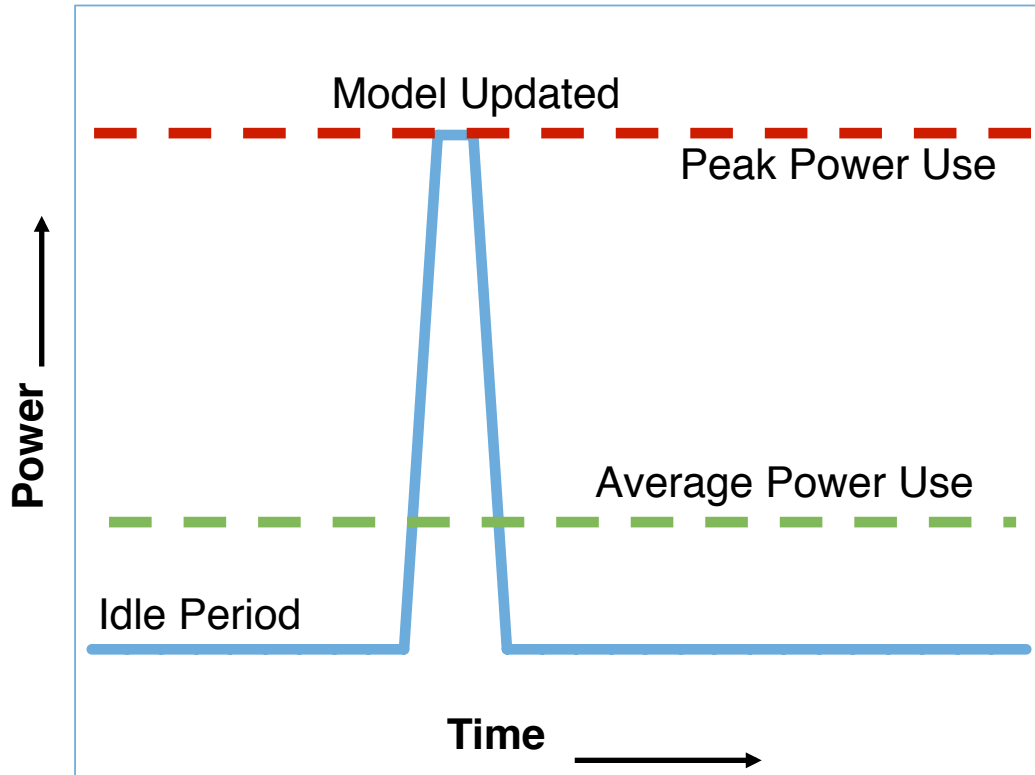


Figure 3.7: Amortization of Training Cost.

have an approximately constant height, and therefore the amortized inference cost for the framework is constant over time.

Unfortunately, when the model cannot be updated online, (as is the case for decision trees), these peak values will continuously increase as new instances are added. This is due to the fact that the model will have to be rebuilt after every training instance is received. To visualize this impact, a J48 decision tree was trained repeatedly on the same dataset of a certain size, N , over the course of five minutes, after which the power consumption was measured. By dividing the total consumption by the number of times the tree was rebuilt in that time, we obtain an estimate of the average power consumption per iteration. Because the time frame was fixed, fewer iterations were run for larger datasets than for small ones. Figure 3.8 shows that the peak power per iteration value increases linearly with N (note

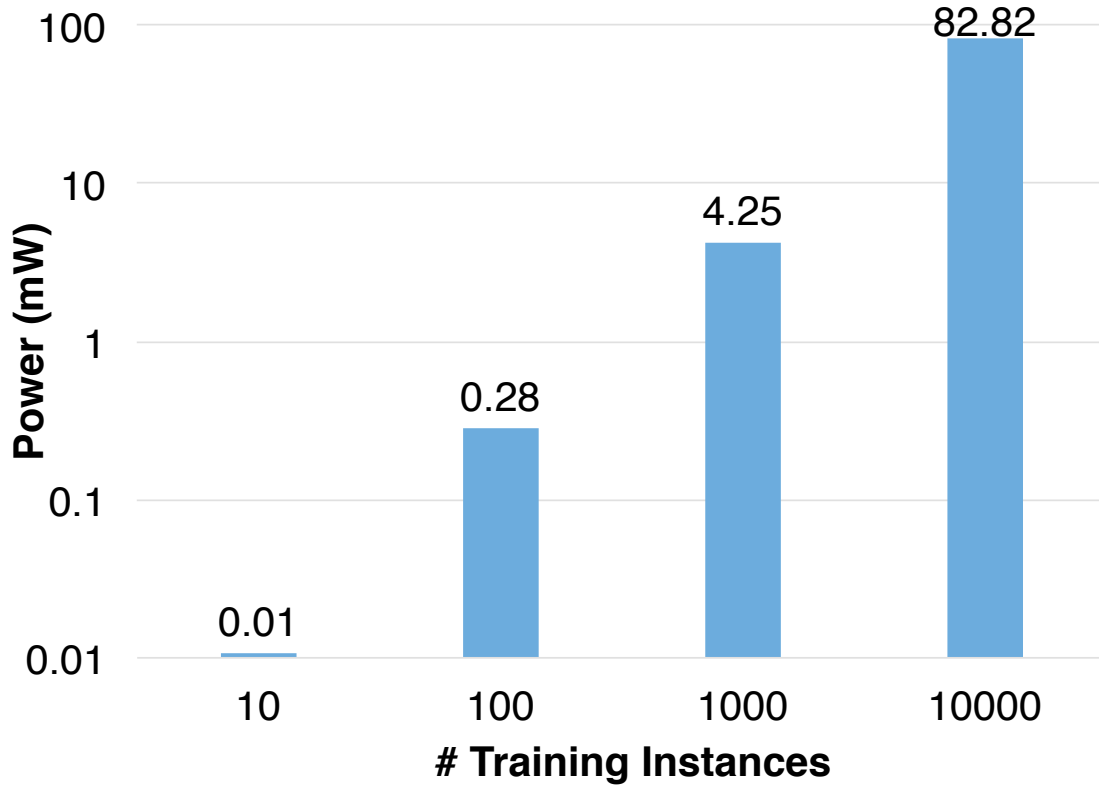


Figure 3.8: Peak power usage compared to the number of training instances. Note both axes are in log-scale.

that both axes are in log-scale), which implies that the framework’s inference cost will rise over time if left unchecked, since the idle cost cost is constant, as before.

There are at least two potential methods to mitigate this liability: 1) reduce the frequency of model updates, and 2) substitute a learning model that is capable of updating incrementally. These two approaches have the added advantage of being mutually orthogonal to one another. Either can be used in isolation, or the two can be used in combination. We will now examine both in more detail.

First we investigate how to reduce the model update frequency. Two methods for doing so are to use instance-based and time-based approaches. In other words, the model can be updated after every K instances or after every t seconds. In order to analyze the potential savings of both methods, a controlled experiment was performed. In this experiment, a

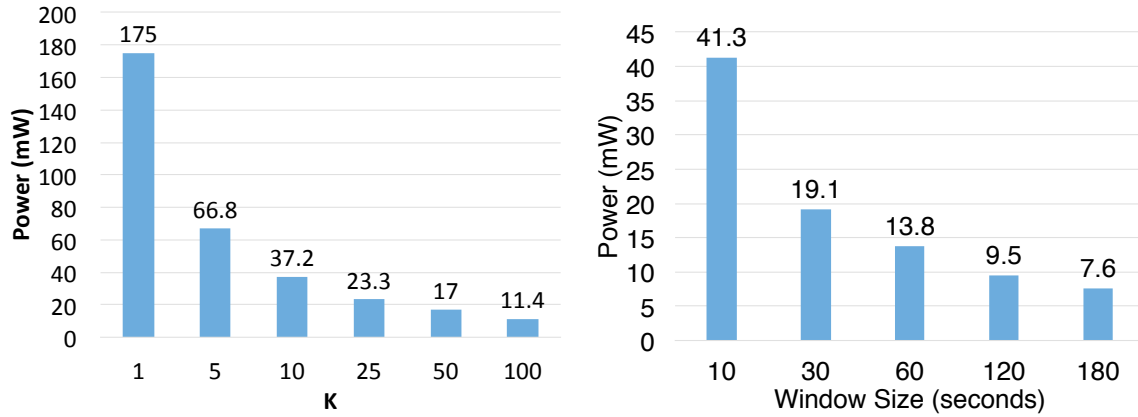


Figure 3.9: Power consumption as a function of the number of instances (left) and the window size (right).

random training instance was generated and added to a growing training dataset. A single J48 decision tree was then trained on the entire dataset, and the process was repeated. Instances were generated at a rate of one per second, and the entire experiment was allowed to run for a set amount of time (5 minutes). This gives a baseline value, where $K = 1$. The experiment was then repeated with the modification that the model was only trained either every K consecutive instances or every t seconds, for varying values of both K and t . The power consumption of both approaches were measured and the results are summarized in Figure 3.9. Using either method, the power consumption would appear to be inversely proportional to the magnitude of the independent variable. Larger values of that variable provide more efficiency at the cost of an additional time delay.

By examining the number of update operations in the same experiment, we can better understand the relationship between the invariant (either K or t and the power consumption. Figure 3.10 shows that as the invariant becomes larger, fewer and fewer update operations occur. With fewer updates, less work is done in the same amount of time, thus reducing the overall power consumption and the cost of updating an offline model.

The problem of rising inference cost with offline models can also be avoided completely by using an online model instead. The models produced by certain learning algorithms, such

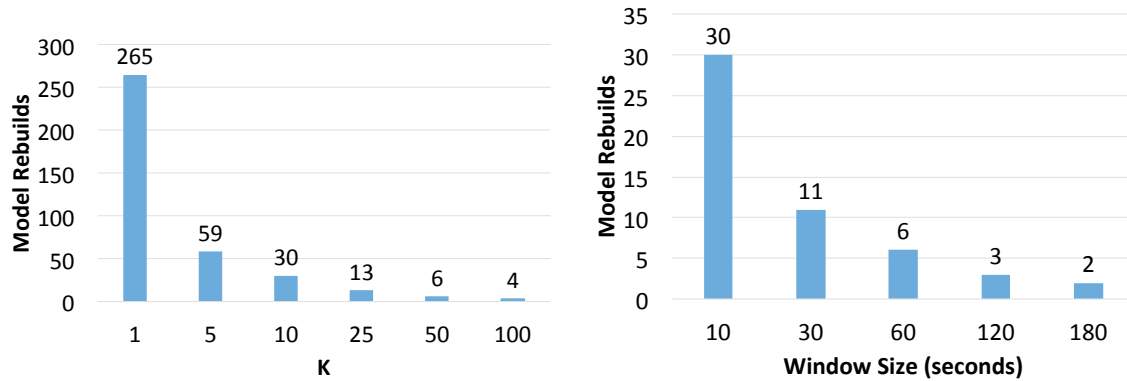


Figure 3.10: Count of model updates as a function of the number of instances (left) and the window size (right).

as artificial neural networks, are readily amenable to online training. As we've mentioned, others, such as decision trees, are not. When a model is used that does support online training, that model can be updated incrementally, rather than rebuilt, which can allow for potentially drastic performance increases. This is because the inference cost will remain constant over time, and that constant value will be lower on average than the continuously increasing cost of the offline model.

To examine this effect in more detail, another controlled experiment was performed. As in the previous experiment, random training instances were generated and added to a growing training dataset, which was used to train a classifier capable of training either in an online or offline manner (Naïve Bayes). No delay was added between training the classifier and adding the next training instance, unlike in the previous experiment. The time required to complete the process once for each of N total training samples was measured, using both the online and offline methods, and the results are shown in Figure 3.11. Note first that the y -axis is in log-scale. For small values of N , the offline approach is only an order of magnitude slower than the online method. This difference quickly progresses to three orders of magnitude, even for relatively modest training data sizes ($N = 2000$). Therefore, an online training algorithm should be substituted for an offline one when it is feasible to do so

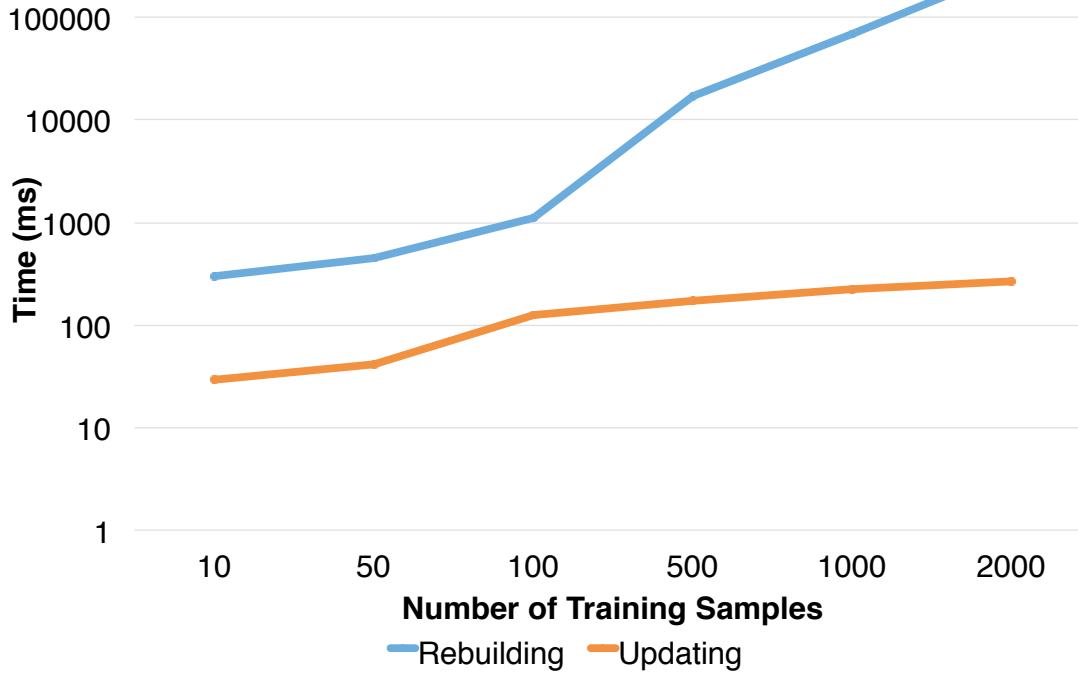


Figure 3.11: Time required to repeatedly rebuild or update a model. Y axis is in log-scale.

because the inference cost remains mostly constant with larger training set sizes.

3.8.3 Breakdown of the sensing and inference overhead

In order to evaluate each of the components of the framework empirically, power consumption was measured with various pieces of the pipeline disabled over a fixed time window. Initially, only the sensing component was enabled, which includes the framework’s overhead costs. Then the storage component was enabled, followed by the feature extraction component, and then inference (in both the training and prediction phases). Unagi was run in two conditions, one representing a normal environment, with few active sensors and events, and another representing a stressed environment, containing many active sensors with high sampling rates and more outside events. The results are summarized in Figure 3.12. First, note that feature extraction consumes more power in the stressed case, as more events imply more features had to be extracted. Also note that there is a more significant reduction

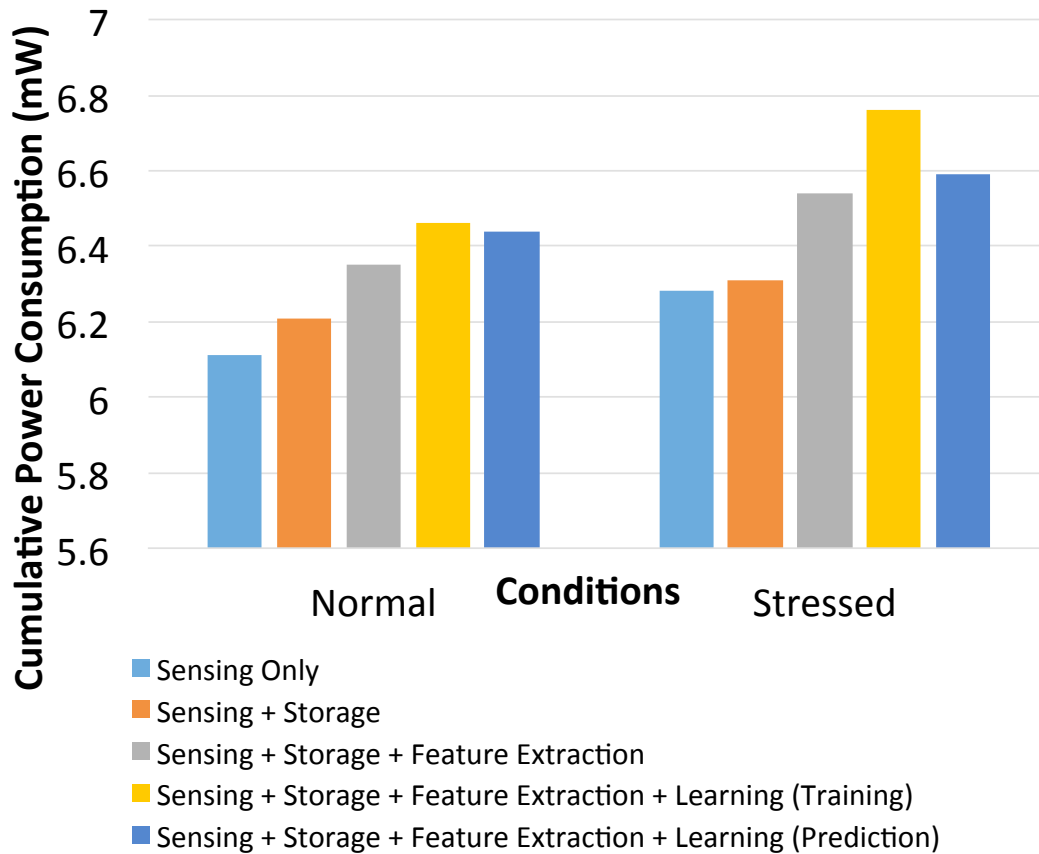


Figure 3.12: Estimated Power Cost Per System Component.

in cost during the prediction phase (which includes fewer features and no online learning) in the stressed example. The same features were extracted in both cases, but the costs of updating the model were more expensive in the stressed environment because there were more instances generated.

3.8.4 Performance of two driving applications

To evaluate the framework in its entirety, an in-situ study was performed using 6 participants over the course of 2-3 weeks. Our study is approved and monitored by the IRB of the University of Arkansas under protocol #13-09-068. Each participant installed a custom Android application <https://play.google.com/store/apps/details?id=edu.uark.inference.app> on their device that implemented the Unagi framework, as well as two

placeholder Action modules—one for each of two driving applications. We will now discuss both in more detail.

Performance of Adaptive Notification Filtering:

The adaptive notification filter module was designed to prevent cumbersome notifications from distracting the user. During training, a secondary notification, like that shown in Figure 3.13 is posted when a normal notification arrives. The secondary notification simply asks if the primary notification should have been filtered or not. In this case, each instance in the training data corresponds to features extracted when the primary notification arrived first arrived, an event-driven approach. The label was provided by a human in order to facilitate supervised learning. After a sufficient amount of training data is collected, Unagi will begin returning results to the client application, based on the predictions of its model. At that point, every time a new notification arrives, the framework will make a prediction as to whether it should be filtered or not and return that result to the client application. The client can then perform the actual notification filtering based on that new knowledge.

The framework will automatically determine when it is appropriate to stop collecting data by occasionally measuring predictive accuracy using standard 10-fold cross-validation. When that accuracy passes a specified threshold, Unagi transitions into a prediction phase where it begins to make predictions. Using all of data collected from the individual participants, we can compare their individual accuracy values at various points during the study. Figure 3.14 visualizes this. Along the x -axis is the percent of total training data used (e.g. the first 10%, the first 20%, and so on). Along the y -axis is the cross-validation accuracy measured using the data up to that point. Percentages are used in lieu of actual instance counts because different participants had comparable, but different data sizes.

We can see that for most users, accuracy drops slightly while the framework collects

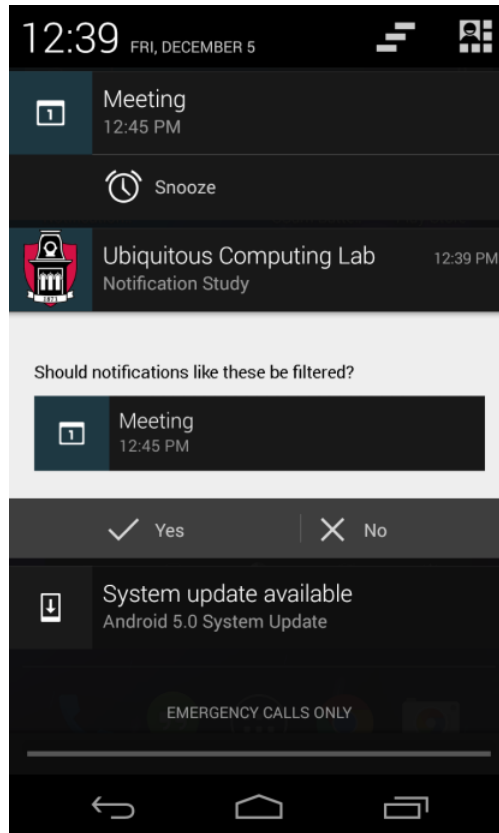


Figure 3.13: UI displayed to user for human labeling of training data instances. Used for the adaptive notification filtering example.

representative data before increasing again when the model stabilizes. User E's patterns are unique in that it takes much longer to collect representative data, compared to the other users. This implies that User E's usage pattern is likely more complicated than that of the other users, either due to lifestyle or noise. Using all of the training data, 5 of the 6 participants reported an accuracy of 90% or better. User E only had an accuracy of 79%, but if the trend shown in the graph continues, that accuracy will continue to increase over time.

The cumulative confusion matrix from all users is shown in Table 3.2. The two columns show what the instance was classified as, the rows show what the ground truth provided by the user was. Based on this table, the cumulative precision was 83.5% and the cumulative recall was 86.5%.

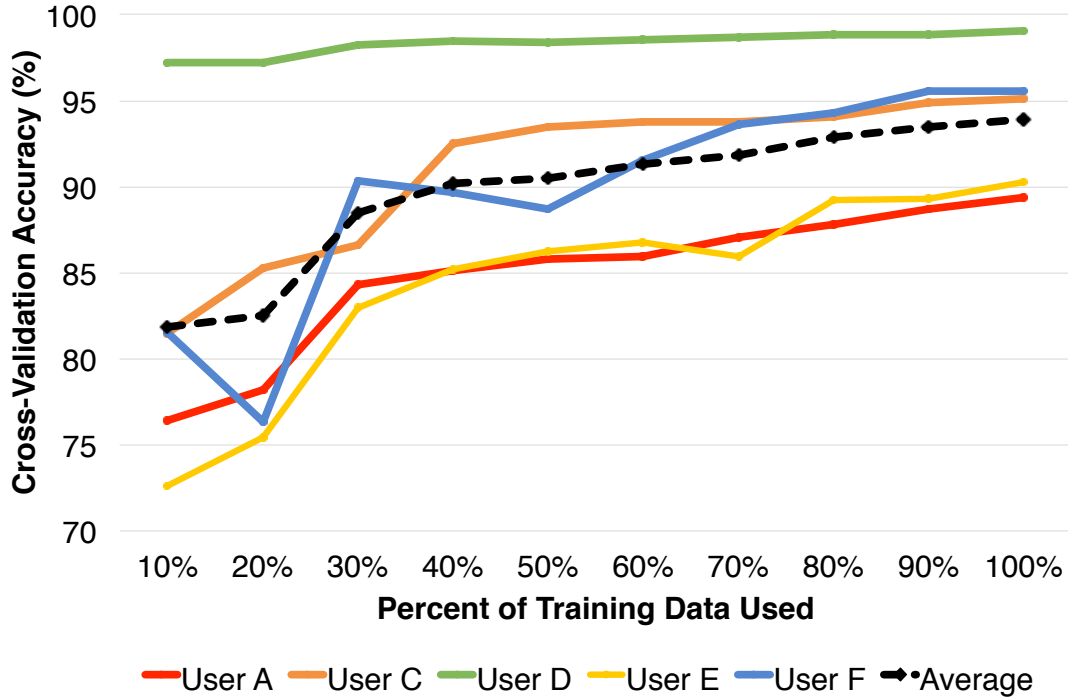


Figure 3.14: Cross-validation accuracy over time for the adaptive notification filter.

Table 3.2: Cumulative Confusion Matrix - Adaptive Notification Filtering

| | Predicted Yes | Predicted No |
|----------|---------------|--------------|
| True Yes | 711 | 111 |
| True No | 141 | 1376 |

Performance of Network Content Prefetching

The network content prefetching model was designed to predict when network content for an app should be prefetched. Such circumstances arise when moving from region with an active WiFi connection to one without, especially when using applications that require internet access. This application differs from the adaptive notification filter in that it is not event-driven. Instead, each instance is a fixed-width time window. It also differs in the source of training labels. While the notification filter required human labels, the labels for the network content prefetching model were generated by the following rules:

```
if Current.WifiState is CONNECTED
    and Current.App is not ‘‘Launcher’’
    and Next.WifiState is DISCONNECTED:
    Label = YES
else:
    Label = NO
```

In other words, when transitioning from region with an active WiFi connection to one without and when the user is actively using applications, we should prefetch network content for that application. This rule set is demonstrative in nature. We do not argue that this is the best way to achieve the desired goal. It is simply one means of achieving that goal. The purpose is to show that Unagi can be useful in varying conditions.

However, there is still one issue to deal with regarding this data—the classes are considerably unbalanced. Because of the strict nature of our rule set, very few affirmative instances will be present in the dataset, compared to an overwhelming majority of negative ones. A baseline algorithm, one that always predicts NO, will perform very well in terms of pure accuracy. One way to alleviate this problem is to stratify the training data by probabilistically resampling it, which creates repetitions of the affirmative classes, effectively balancing the dataset. We have adopted this approach for Unagi .

Similar to the previous application, Figure 3.15 shows the accuracy over time for the same users using the resampled dataset. User B had no positive instances, so was removed from consideration. Stratification was performed after the appropriate percentage was extracted from the original data. Similar to the previous example, the accuracy values are initially somewhat chaotic while representative data is collected, but they even out as time progresses and more instances are added. Using all of the training data, each user achieved more than 90% accuracy. The average across all users was nearly 94%.

As before, the cumulative confusion matrix is shown in Table 3.3. Based on this table, the cumulative precision was 94.5% and the cumulative recall was 97.9%.

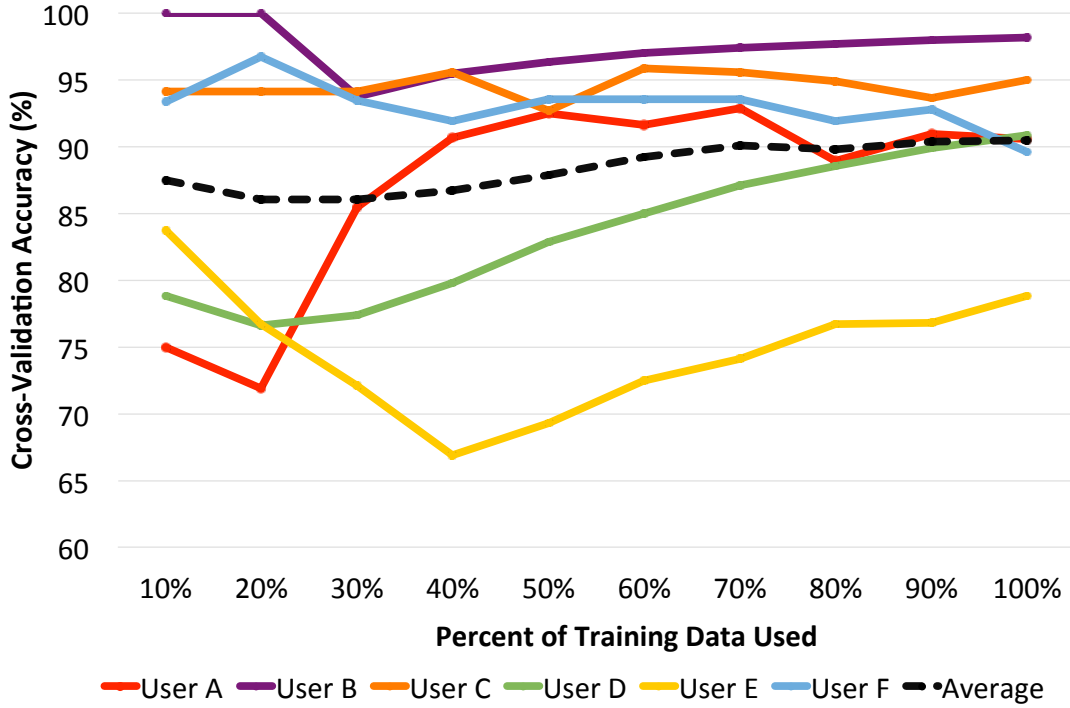


Figure 3.15: Cross-validation accuracy over time for the network content prefetching service.

Table 3.3: Cumulative Confusion Matrix - Network Prefetching

| | Predicted Yes | Predicted No |
|----------|---------------|--------------|
| True Yes | 8115 | 170 |
| True No | 471 | 8019 |

3.9 Conclusion and Discussions

This paper presents the design and implementation of Unagi , a sensing and inference framework for OS context-based cognitive applications. Unagi provides a comprehensive investigation of three OS context sources and a virtual sensor design for universal access of heterogeneous data sources. Based on a set of virtual sensors, Unagi further automates the process of user behavior pattern inference with a set of feature design, feature selection, and model updating methods. Benchmark results indicate that continuous running of Unagi on mobile phones incurs negligible energy, computation, and storage cost. Two case studies

indicate that the automatically generated inference models can achieve high accuracy while simultaneously incurring low system overhead for both supervised and unsupervised learning scenarios.

While Unagi can significantly simplify the development of cognitive applications, it also faces several limitations. Unagi takes advantage of the fact that the OS context is inexpensive to retrieve, and therefore a comprehensive list of virtual sensors is feasible. However, the cost of accessing hardware sensors is not negligible, which means we cannot extend the methodology of Unagi directly to hardware sensors. A hybrid approach, where virtual sensors are used as inexpensive triggers for hardware sensors, could potentially solve this problem. Unagi also sacrifices specially for generality. For some specific applications, the automatically selected features or the behavior model might not be the best choice. Unagi does, however, allow developers to insert their own feature generator and/or behavior pattern learner into the framework to handle the special cases. Unagi should also evolve itself with new OS features and/or new machine learning methods. We also evaluated Unagi with limited applications, with a small-scale user study. A larger scale study would be helpful to further validate its use in this domain.

4 Depth-based Positioning

This chapter focuses on our work regarding depth- and inertial-based positioning. We discuss DIPS, a novel indoor positioning method that uses depth and inertial sensor data to localize both mobile users and objects in their ambient context with up-to centimeter-level accuracy. The key contributions of this technique include a thorough theoretical analysis of the problem, a chaining method that extends the effective localization range from a few meters to more than 30 without significant error accumulation, and an optimization that can further reduce accumulative error. We have implemented our method on Google's Project Tango platform, and early evaluation results show that the localization error can be less than 10 centimeters in practical settings using an unmodified consumer-level device.

4.1 Introduction

With the proliferation of smart mobile devices, especially wearables, we have seen a surge of new context-aware mobile applications. A typical example is Augmented Reality (AR), which integrates augmented information and user-interactions with the physical context. Microsoft HoloLens [13], for example, is an AR platform that incorporates physical real-world elements, such as walls or the surface of a vehicle, into the presentation of application content. Google Glass [14] and other similar systems [15, 16] are pioneer platforms that enable egocentric AR applications with an optical head-mounted display (OHMD). When augmented information can be tightly integrated with physical objects, natural user interactions can be enabled.

A core challenge of new context-aware applications is determining the precise location of mobile users and the objects in their ambient context. The two primary approaches

that are currently used to enable ambient context sensing are those that rely on embedded RF transceivers (e.g.: NFC and Bluetooth), and those that rely on computer vision (CV). RF-based approaches suffer from the primary limitations of scale and accuracy. Because they typically require certain infrastructure elements, the heavy cost of both deployment and maintenance of such elements is a substantial hurdle to the ubiquity of ambient context-sensitive applications. Furthermore, while RF-based systems are capable of reasonable accuracy, for some applications, such as AR, modest accuracy is simply not enough. In order to enable ubiquitous AR applications, reliable centimeter level accuracy is required. CV-based approaches are comparatively much more accurate than RF-based ones, and they can be deployed without a significant infrastructure, but they do suffer from a different drawback—they are usually quite expensive, in terms of both energy and computation. Due to the small form factor of mobile and wearable devices, existing CV-based methods are typically too resource intensive for practical use.

Due to the intrinsic limitations of both RF- and CV-based context sensing methods, alternative technologies have recently started to become more popular. One example is an infrared projector paired with a specialized camera tuned to the IR spectrum like that used in Microsoft’s Kinect and Google’s Project Tango. These sensors (e.g.: OmniVision OV4682) typically possess regular RGB imaging capabilities as well as a depth sensor. Compared with pure CV-based 3D construction methods such as stereo vision, IR cameras are typically more accurate in distance measurement and are much more energy-efficient in practice [26, 27].

In this chapter, we discuss our current work with a novel context positioning method named DIPS, which stands for “Depth- and Inertial-based Positioning System”, that is able to accurately co-localize mobile users, as well as critical objects in their context. The core of the proposed method is a technique that combines information from both the inertial sensors and an IR-based depth sensor in order to achieve localization accuracy within 10 cm over a

range of 30 meters (and beyond) using off-the-shelf commercial hardware and no additional infrastructure elements. To realize this goal, we first provide a theoretical basis and practical design considerations for simple short-range localization goals. Next, we propose a multi-hop approach that extends the localization range with a chaining method that alternately localizes mobile users and critical objects in the environment while simultaneously minimizing the accumulation of error over time. In addition, we propose an optimization method to further eliminate error accumulation over multiple hops, which can help to realize the goal of high localization accuracy over long distances if there is enough uncertainty in the data. Lastly, we demonstrate the utility of the system under real world conditions with a prototype implementation. To summarize, the core contributions of this paper are as follows:

- We propose DIPS, a method that enables indoor localization of both mobile users and contextual landmarks with an accuracy of just a few centimeters.
- We perform a thorough theoretical study of the technique to provide a solid mathematical justification for its use.
- We propose a chaining method that extends the effective localization range from a few meters to more than 30 without significant error accumulation.
- We also propose a statistical optimization to the chaining method that allows the framework to use noisy measurements to its advantage.
- We prototype DIPS on Google's Project Tango tablet and use it to evaluate our method in real world conditions.

4.2 Background

4.2.1 Indoor Localization

Locating mobile users and ambient objects in their environment is an active research field in mobile computing. Odometry [17] is a classic method to realize simultaneous localization and mapping (SLAM) in a GPS-denied environment. However, inertial navigation systems can suffer from integration drift, where small errors accumulate to cause inaccurate localization results over time [18].

RF-based localization is another widely investigated approach that is well-suited to locating moving objects, as well as objects attached to RF transceivers in indoor environments. The primary idea of RF-based methods is to take advantage of the unique RF signal fingerprints at different places for localization purposes. Common RF-based methods include WiFi localization [19], Near-field communication [20], and Ultra-wideband methods [21]. Recent RF-based methods can locate an object within 1-2 meters [22].

Image-based indoor localization is another well-studied field [23]. Stereo vision [24] can compare a scene from two vantage points and extract 3D information from the objects in the scene. Stereo vision on mobile platforms faces several practical challenges, in particular its high computational and energy cost [25].

Optical methods, such as IR-based localization are a fairly recent trend. Industrial systems, such as AICON 3D Systems [28] can reach an accuracy of centimeter or even millimeter level but are difficult to integrate with mobile systems. Consumer-level platforms, such as the widely-used Microsoft Kinect [29] and Google's recent Project Tango tablet [30], are equipped with small form-factor IR optical sensors and have begun to show the tremendous potential of optical-based localization for mobile devices [31, 32].

4.2.2 Optical Imaging and Context Localization

Compared with RF- or vision-based approaches, optical sensors have a unique advantage: they provide direct (and therefore fairly accurate) depth measurements of a scene. As such, optical sensors, often combined with regular RGB cameras, have been used to construct 3D scenes in indoor environments, enabling applications such as robot navigation [84] and AR [85]. Recent 3D scene mapping and reconstruction techniques, such as live 3D scanners [86], can reach an accuracy of a few centimeters but only within a short range of around a meter [87, 88]. Extending the distance of optical-based localization is a significant challenge, as optical sensors have a limited range (typically less than 5m), and the depth measurements become increasingly noisy as distance increases. Many existing approaches handle the range challenge by integrating stereo RGB information [84], but despite their effectiveness, their disadvantages on mobile platforms severely hinder their broader use. In this work, we propose a novel method of using information from both optical and inertial sensors in order to enable improved location accuracy with comparatively less computational cost.

4.3 Depth- and Inertial- based Localization

In this section, we first formally define the error model used for depth-based localization. Equation 4.1 represents the fundamental technique used for our localization problem. Given a known location, A , if we have access to a distance measurement, d , and a unit direction vector, \vec{v} , we can calculate the position of an unknown object, B as follows.

$$B = A + d\vec{v} \quad (4.1)$$

Based on Equation 4.1, it is clear that any localization error can be decomposed into

Table 4.1: Common Notation

| Notation | Meaning |
|-------------------------|---|
| $A, B, \text{ etc.}$ | A location of a 3D point |
| \hat{A} | An estimated location of a 3D point |
| \bar{A} | An average of several points |
| d | A true distance |
| \dot{d} | A measured distance |
| \vec{v} | A true unit vector |
| $\dot{\vec{v}}$ | A measured unit vector |
| \vec{X} | A vector (not necessarily of unit length) |
| $\vec{X} \cdot \vec{Y}$ | Inner (dot) product |
| $\ \vec{X}\ $ | Vector magnitude |
| \sim | Distributed according to |

two orthogonal components, depth error and directional (or rotational) error. In other words, localization error can be a result of inaccuracies with the depth measurement, the direction vector, or some combination of the two. In order to account for and correct this error, we must model each individual component and analyze their joint effects on localization accuracy.

4.3.1 Depth Error Model

Generally, depth error stems from deficiencies in either the hardware or the associated software of the depth sensor, but it could also be a result of lossy processing of depth data on the client side. Regardless of the source, modeling depth error is a simple matter of comparing many measured depth values to their associated ground truths. Given an empirical model derived from such data, depth measurements can be calibrated appropriately.

For an arbitrary depth sensor, the depth error can be modeled as:

$$e_d(\dot{d}) = |\dot{d} - f(\dot{d})| \quad (4.2)$$

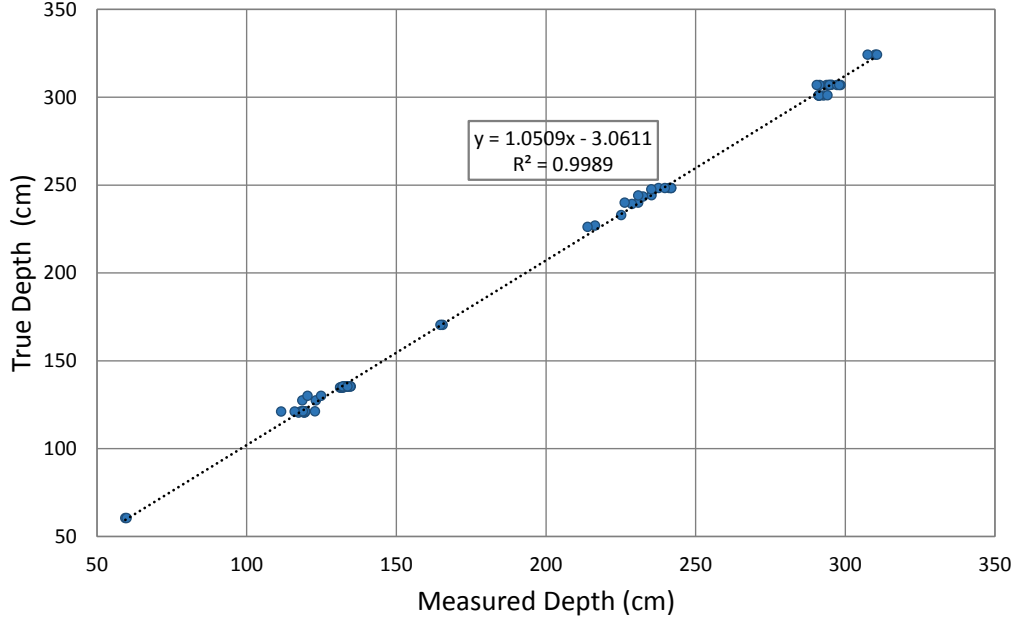


Figure 4.1: An empirical model of the calibration curve for a Project Tango tablet.

where $f(\dot{d})$ is any function of the measured depth value, \dot{d} . $f(\dot{d})$ is essentially a *calibration curve*. In order to calibrate depth measurements, one simply needs to replace the measured depth value, \dot{d} , with $f(\dot{d})$ in localization calculations. In Figure 4.1, we show the calibration curve for a Google Project Tango tablet that was used in our laboratory setting. We measured approximately 70 objects at various known distances and plotted them against their ground truth values, which were measured separately and manually. Based on these observations, $f(\dot{d}) \approx 1.0509\dot{d} - 3.0611$ (depth is measured in centimeters) for that device. If another sensor is used, however, the calibration curve will have to be recalculated.

4.3.2 Rotational Error Model

Rotational error is more difficult to analyze because it contains components of both system and user error. System error arises from the use of one or more hardware sensors, such as an accelerometer or a gyroscope, as well as inertial drift of the data from those

sensors over time. User error is a result of a system requirement that the user must look directly towards an object in order to perform localization. Even if the user fully intends to cooperate, it is highly likely there will be an angular difference between the true direction and the perceived direction from the user.

To fully characterize the rotational error, we need to account for both sources. More formally, we describe this error as $e_r(\vec{v}) = f(s(\vec{v}), u(\vec{v}))$, where $e_r(\vec{v})$ is the rotational error, $s(\vec{v})$ is the system error, $u(\vec{v})$ is the user error, and $f(s, u)$ is the accumulative effect of these two factors.

System error can generally be modeled as a simple (e.g. polynomial or logarithmic) relationship between the error and time, but in the case of Project Tango, it can at least partially be removed before the client receives the data by applying drift correction [89]. After this correction has been performed, system error is typically reduced enough that the user error tends to dominate the rotational error equation. Under these conditions, the rotational error model can effectively be simplified to $e_r(\vec{v}) = u(\vec{v})$.

We are left with a need to model the user error, $u(\vec{v})$. To do so, we will assume the user is cooperative, meaning we assume the user is honestly attempting to look directly at the object of interest. (Working with uncooperative users is a topic for future research.) Because users rarely behave deterministically in practice, the user error is best described probabilistically instead. We model the user error as $u(X) \sim \mathcal{N}(b, \sigma)$, where X is a random variable that represents the angular offset between the true direction, \vec{v} , and the measured direction, $\dot{\vec{v}}$. In the formula, b and σ represent an individual user's bias and average deviation, respectively.

In the last equation, another way of visualizing X can be to apply the following equation: $X = \cos^{-1}(\vec{v} \cdot \dot{\vec{v}})$. The dot product between the true direction and the measured direction gives the cosine of the angle between them. If we then take the inverse cosine, we

obtain the angle itself. That angle is what is normally distributed, so ideally, $b = 0$ and σ is very small, which implies that the user is actually looking directly at the object of interest. With this addition of $u(X)$, the complete model for rotational error becomes simply:

$$e_r(X) \sim \mathcal{N}(b, \sigma) \quad (4.3)$$

4.3.3 Localization Error.

Now that we have established models for both the depth and the rotational error, we can discuss their effects, both individually and combined.

Isolating Depth Error

The effect of isolating the depth error when localizing an object is the following. Suppose we are currently standing at point A and we are attempting to localize point B . The true location of B is given by Equation 4.1. Due to inaccuracies in our depth measurement however, we actually measure \dot{d} , which means our estimate of B is $\hat{B} = A + \dot{d}\vec{v}$. The error is the distance between the two, or $e = \|\hat{B} - B\| = \|(A + \dot{d}\vec{v}) - (A + d\vec{v})\|$. After simplifying, we get $e = \|\dot{d}\vec{v} - d\vec{v}\| = |\dot{d} - d|$. In other words, the isolated depth error is equal to the difference between the measured depth value and the true depth value.

Isolating Rotational Error

As with the depth error, we can isolate the effect of rotational error when localizing an object. Again, suppose we are currently standing at point A and we are attempting to localize point B . Due to inaccuracies in our direction measurement, we actually measure $\dot{\vec{v}}$, which means our estimate of B is now $\hat{B} = A + d\dot{\vec{v}}$. The error is once again the distance between the two, or $e = \|\hat{B} - B\| = \|(A + d\dot{\vec{v}}) - (A + d\vec{v})\|$. After simplifying, we get

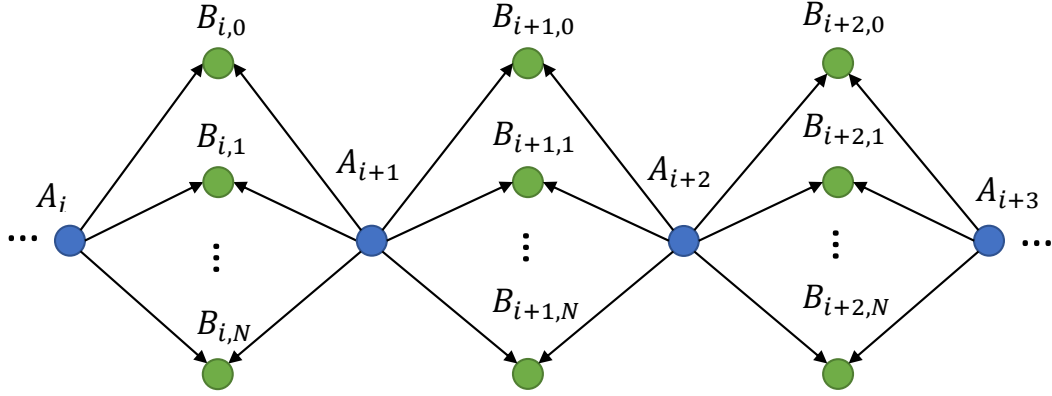


Figure 4.2: Overview of the chaining method.

$e = \|d\dot{\vec{v}} - d\vec{v}\| = \sqrt{d^2 + d^2 - 2d^2(\dot{\vec{v}} \cdot \vec{v})} = \sqrt{2d^2 - 2d^2 \cos \theta}$, where θ is the angle between \vec{v} and $\dot{\vec{v}}$. A result of this equation is that the isolated rotational error depends not only on distance between the objects, as was the case when the depth error was isolated, but also on the angular offset between the true direction and the measured direction.

Combined Error

When we apply a similar logic to the task of localizing an object with *both* depth and rotational error, we end up with $e = \|\dot{d}\vec{v} - d\dot{\vec{v}}\|$, which works out to $e = \sqrt{\dot{d}^2 + d^2 - 2(\dot{d})(d) \cos \theta}$. This formula has an important interpretation: rotational error is typically going to be more significant than depth error, as the former will dominate the latter as the range between the two objects increases. This is primarily due to the fact that any rotational error that is present will be scaled by the distance to the object.

4.3.4 Depth Processing.

We now address a practical challenge that arises with regards to the error model. Google's Project Tango tablet, as well as several other commercial depth sensors, provide depth data as a 3D point cloud, but our various localization equations require only a single

value. We must first process the point cloud in order to obtain this value before we can proceed with localization. The first step is to create a 2D depth map from the raw point cloud data using a perspective transformation. The values stored in the depth map are the distances to the corresponding 3D point from the center of the depth camera.

Ideally, the cell of the depth map we are interested in has a value, but as the map is sparse, that is unlikely to be the case in general. If nearby points do have values, one approach is to use nearest-neighbor sampling to fill in the gaps between valid depth values in the depth map. In other words, for each pixel that does not have a valid depth value, we search a small neighborhood for pixels that do have valid depth values and average them to estimate the value for that pixel. Depending on the size of the neighborhood window used, the depth map can be smoothed by varying amounts before being sampled directly.

If it is known a priori which cells in the depth map we are interested in, we can perform a significant optimization: we can choose to interpolate only those cells, reducing the computational cost from $O(WN)$ to $O(Wn)$, where W is the size of the window used, N is the number of pixels in the image, and n is the number of pixels that are of interest to us.

In order to make the selection of depth points more robust to noise, we use a variable-sized median filter. Ideally, if we are looking for a depth value in a particular cell, that cell will already have a valid value. As this is unlikely, however, we can create a small window (different from that used for interpolation) and return the median value of all the valid depth values in that window. If there were no valid values, the window size can be increased (up to a user-specified maximum value) until such a search is successful.

4.4 Creating a Localization Chain

If we have available to us both a depth measurement and a direction vector, localizing an object is a simple matter of applying Equation 4.1. However, as all depth sensors have a limited range for which they are effective, we would only be able to localize objects that are nearby with that method. To extend the useful range of depth sensing devices, we propose the following extension to the idea.

At a high level, our goal will be to create a “chain” of localizations from a known starting point. Figure 4.2 details the process visually. The procedure is as follows. From the initial starting point, A_1 (whose location is known), we can use Equation 4.1 to localize N points in the immediate vicinity ($B_{1,1} \dots B_{1,N}$). Those N intermediate points can then themselves be used to determine another location, A_2 . This sequence constitutes a single *hop*. To increase the effective localization range, we add more hops to the chain, adding as many links as necessary.

4.4.1 One Hop Localization

First we will examine a single hop in more detail. Figure 4.3 shows the physical layout of a single hop. The solid lines represent the true values and the dotted lines represent measured ones. The heavily shaded circles are the true locations of the points, and the lightly shaded circles are the estimates generated by the equations. Estimates of the points are only shown for the first intermediate landmark, but the process is analogous for the others (which are shaded gray).

A single hop is made up of N *forward links* followed by N *back links*. Each forward link calculates an estimate of the position of an intermediate point (B_j). Each back link will yield one estimate of the location of A_{i+1} . Those N estimates are averaged together to calculate a single unified estimate for future calculations. The following three equations

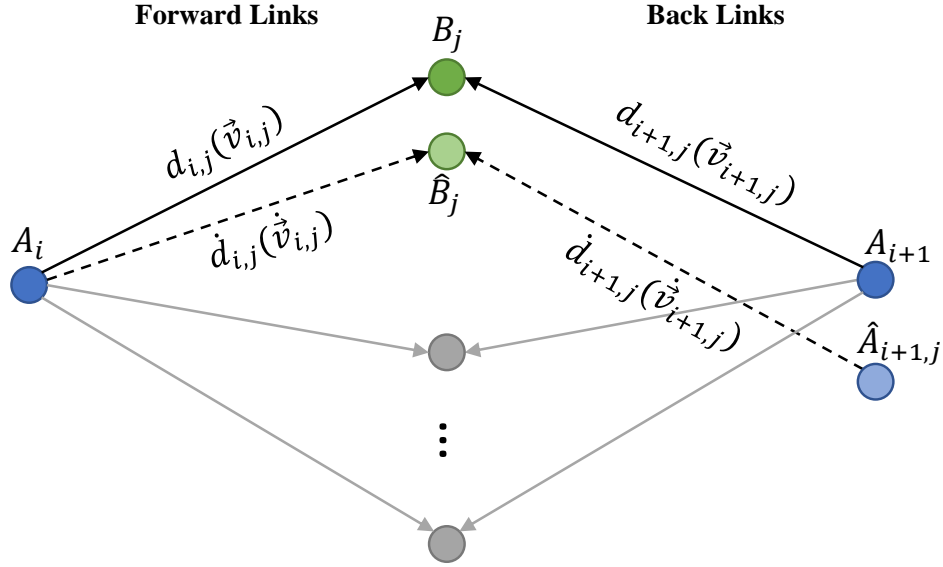


Figure 4.3: Relationship between the parameters for a single hop.

formalize the notion of a single hop mathematically.

$$\hat{B}_{i,j} = A_i + d_{i,j}(\vec{v}_{i,j}) \quad (4.4)$$

$$\hat{A}_{i+1,j}^- = \hat{B}_{i,j} - d_{i+1,j}(\vec{v}_{i+1,j}) \quad (4.5)$$

$$\hat{A}_{i+1} = \frac{\sum_{j=0}^N \hat{A}_{i+1,j}^-}{N} \quad (4.6)$$

Equation 4.4 represents a single forward link between A_i and B_j , where B_j is any one of the intermediate points used in the chain. $\vec{v}_{i,j}$ is defined as a unit vector in the direction from A_i to B_j , and $d_{i,j}$ is the distance between A_i and B_j . Values that are measured by the device are denoted \dot{d} and \vec{v} . To estimate the location of B_j , we traverse a distance of $\dot{d}_{i,j}$ along the direction vector starting from A_i .

In Equation 4.5, we follow a similar procedure. We start at A_{i+1} and look towards B_j , whose location estimate, $\widehat{B}_{i,j}$ we calculated with Equation 4.4. Here, $\vec{v}_{i+1,j}$ is the unit vector from A_{i+1} to B_j , and $d_{i+1,j}$ is the corresponding distance.

The final step, applied in Equation 4.6 but not shown in Figure 4.3, is to average the N estimates of A_{i+1} from each back link to create a single location estimate for A_{i+1} . Since both the forward and back link processes require one depth measurement and one direction measurement each, a single hop requires two of each for each intermediate point that is used. In other words, $2N$ depth measurements and $2N$ direction measurements are required for each hop.

Calculating Error

The amount of error that results from one hop can be determined algebraically. This is important because we will show in the next section how individual hops can be isolated from one another, meaning that the error at each point in the chain can be calculated irrespective of any other point in the chain.

Figure 4.3 shows the physical setup for an arbitrary hop. In the diagram, $A_{i+1,j}$ is the true location of the point A_{i+1} , calculated by going from A_i through B_j . Since we are referring to the ground truth here, the choice of which j to use is irrelevant—they will all yield the same result. The following equation represents the location of A_{i+1} mathematically.

$$A_{i+1,j} = A_i + d_{i,j}(\vec{v}_{i,j}) - d_{i+1,j}(\vec{v}_{i+1,j})$$

The next equation is the result of combining Equations 4.4 - 4.6. It represents the estimate of A_{i+1} given the various depth and direction measurements between A_i , B_j , and A_{i+1} . In this equation, d represents an actual distance and \hat{d} represents a measured distance (which likely does contain some error). Similarly, \vec{v} represents an actual direction vector and

\vec{v} represents a measured direction vector. As before, N is the number of intermediate points that are used.

$$\widehat{A}_{i+1} = \frac{\sum_{j=0}^N A_i + \dot{d}_{i,j}(\dot{\vec{v}}_{i,j}) - \dot{d}_{i+1,j}(\dot{\vec{v}}_{i+1,j})}{N}$$

Given that we know both where A_{i+1} really is and our estimate from our measurements, the error is simply:

$$e = \|\widehat{A}_{i+1} - A_{i+1,j}\|$$

Expanding for both \widehat{A}_{i+1} and $A_{i+1,j}$:

$$e = \left\| \left\{ A_i + \frac{\sum_{j=0}^N \dot{d}_{i,j}(\dot{\vec{v}}_{i,j}) - \dot{d}_{i+1,j}(\dot{\vec{v}}_{i+1,j})}{N} \right\} - \{A_i + d_{i,j}(\vec{v}_{i,j}) - d_{i+1,j}(\vec{v}_{i+1,j})\} \right\|$$

There is an A_i term that disappears after the subtraction:

$$e = \left\| \left\{ \frac{\sum_{j=0}^N \dot{d}_{i,j}(\dot{\vec{v}}_{i,j}) - \dot{d}_{i+1,j}(\dot{\vec{v}}_{i+1,j})}{N} \right\} - \{d_{i,j}(\vec{v}_{i,j}) - d_{i+1,j}(\vec{v}_{i+1,j})\} \right\|$$

Splitting the two halves into separate variables for readability:

$$\vec{X} = \frac{\sum_{j=0}^N \dot{d}_{i,j}(\dot{\vec{v}}_{i,j}) - \dot{d}_{i+1,j}(\dot{\vec{v}}_{i+1,j})}{N}$$

$$\vec{Y} = d_{i,j}(\vec{v}_{i,j}) - d_{i+1,j}(\vec{v}_{i+1,j})$$

$$e = \|\vec{X} - \vec{Y}\| = \sqrt{\|\vec{X}\|^2 + \|\vec{Y}\|^2 - 2(\vec{X} \cdot \vec{Y})} \quad (4.7)$$

In Equation 4.7, the second form of the equation is a result of applying elementary geometry and the law of cosines to the first. It helps to demonstrate that the error that arises from a single hop depends not only on the measured depth values and direction vectors, but also on their relationship with the true values.

4.4.2 Multi-hop Object Localization

An important concern with the chaining method is the accumulation of error between hops. Conceivably, if there was any error present in the estimate for point A_i , that error could propagate when estimating point A_{i+1} and every other point farther down the chain. In other words, one could expect a graph of the localization error after each successive hop to have a positive slope (though not necessarily a linear shape). We call such a graph the *error accumulation curve*.

In order for the chaining method to be generally applicable in real world scenarios, the slope of the error accumulation curve must be reduced as much as possible, ideally to 0. When that slope nears 0, each hop loses any sense of dependency on those that came before it in the chain. This implies that the error for any given hop comes entirely from the $2N$ depth measurements and the $2N$ direction measurements that were used in its calculation and not from any previous measurements used for other hops, which essentially guarantees that the chain can be extended indefinitely. The amount of error for an arbitrary hop will approach the theoretical value given in Equation 4.7 as the slope of the error accumulation curve approaches 0.

Since our goal is for the chaining method to be applicable for arbitrarily long sequences, it is vital that both the depth and rotational error from each measurement be minimized as much as possible. Based on the previous discussion in Section A, depth error can be modeled empirically. As such, it can be corrected easily using a device-dependent

calibration wherein the measured depth value, \hat{d} , is replaced with the estimated true depth value, $f(\hat{d})$ in the localization calculations.

Because rotational error is distributed normally, its effects can be countered with statistical averaging. Rather than using a single direction sample, we average M of them over a given time window. Given a large enough M , the rotational error degenerates to b , the user bias. While the user bias cannot be removed completely, it can be reduced significantly by presenting the user with a visual guide (e.g. crosshairs or a bulls-eye). Aids such as these will assist the user in looking at the desired point, thereby reducing their personal bias.

With these suggestions in mind, we will show (in Section 4.6) that it is possible to reduce the slope of the error accumulation curve by a sufficient amount to enable multi-hop object localization using the chaining method in real world scenarios.

4.5 Statistical Optimization

The chaining method has one additional potential weakness—it uses the depth and direction measurements from the device directly. Despite our best efforts to reduce the error, the measurements will still likely contain some noise. Rather than maintain an antagonistic relationship with it, is it possible to use that noise to improve the accuracy of our estimates? This question is the foundation of an optimization technique that we have created to supplement the standard chaining approach detailed in the last section.

At a high level, for a single hop, we require $2N$ depth measurements and $2N$ direction measurements to estimate the location of a given point (where N is still the number of intermediate points employed). If we have some understanding of the error range of each of those parameters, we can modify them slightly such that the N estimates of the location of the given point more closely agree with one another.

As described, this approach is a constrained optimization problem. We optimize

each parameter with the restriction that it stays within some defined error bounds. If an oracle were present, the goal function would adjust each parameter such that each calculated estimate equals the ground truth value, effectively eliminating all error from each parameter. However, since an oracle is not normally present, our goal function should instead be a quantity that can actually be measured under real world conditions.

One heuristic that we have found to work well in practice is to minimize the sum-squared distance (*SSD*) from the centroid for each location estimate. Intuitively, the idea is that we want all of our location estimates to come to a consensus as to what the correct answer is, whilst adhering to their various individual constraints. In order to do so, each measurement will be adjusted slightly until they are sufficiently aligned.

$$SSD = (\hat{A}_{i+1,j} - \bar{A}_{i+1})^2 \quad (4.8)$$

In Equation 4.8 we formalize the error metric that we use. Mathematically, the goal is to minimize *SSD*, where $\hat{A}_{i+1,j}$ is a single estimate of the location of point A_{i+1} using intermediate point j , and \bar{A}_{i+1} is the current centroid of all the estimates.

In general, the choice of the optimization technique that is used is inconsequential, as long as a sufficient local optimum can be found. We applied a simple gradient descent approach in our implementation that iterated until the *SSD* changed by less than a small preset ϵ or until a maximum number of iterations were performed.

4.6 Evaluation

4.6.1 Experimental Setup

Sensor Data Collection

We use a Google Project Tango tablet for our experiments. The Tango tablet provides depth measurements at a rate of approximately 5 Hz. It also provides pose estimates, including an orientation vector, at a rate of around 50 Hz. Although the tablet is capable of providing RGB color information, we do not make use of that data for any of our localization calculations. The Project Tango tablet has a specified depth range of 0.5 m to 4.0 m, but we found experimentally that it was difficult to get reliable readings farther than around 3.0 m in practice. Therefore, we set 3.0 m as the maximum depth range used in our experiments.

We built a custom Tango application to collect both depth and inertial data from the tablet. The application allows users to take measurements between the device and an object at the center of the scene. A single measurement consists of 10 depth samples (each of which is calculated using the method in Section 4.3) and a variable number of direction samples. In an effort to remove the effects of rotational error, all of the direction samples that fall within the measurement's time window are averaged to estimate the user's true direction at that moment. After a measurement is taken, the depth and rotation measurements are saved to a file so they can be processed offline later.

Using our data collection application is straightforward: when a user wants to localize an object, he or she only needs to point the tablet at that object. From the user's perspective, this process is quite similar to taking a photograph with a smart phone. The UI of our data collection application is shown in Figure 4.4. As shown in this figure, the UI forwards the live camera feed to the screen and superimposes a targeting reticle in the center. If the user uses the reticle to align his direction with the point of interest, it helps to further

reduce the affects of rotational error on the measurement that is taken by the application. The process of taking a single measurement typically takes only a couple seconds.

Laboratory Setup

All of our experiments were conducted in a laboratory setting, seen in Figure 4.4. We chose to use tripods to represent points of interest as they allowed us to quickly reconfigure the environment for different setups. We placed various tripods on a floor grid which allowed us to accurately measure ground truth. The Tango tablet was also mounted to a tripod in order to simulate a user.

Ground Truth

For each experiment, we either determined the ground truth manually or assigned it a priori. In the former case, we used laser levels to project the desired point on the tripod onto the floor grid. By measuring the offsets in the x and y axes, we were able to calculate the 2/3 of the direction vector between the user and the various points of interest. The last component of the direction vector, the height differential, was computed by measuring the heights of both objects and subtracting them. When we assigned the ground truth a priori, the tripods used as points of interest were physically moved to match the expected location before the experiment began.

4.6.2 Single Object Localization

Since any number of intermediate objects can be used to localize a single point, we first demonstrate the relationship between that number and localization accuracy. We conducted an experiment in which up to 5 different objects were used to localize the user. We analyzed the error that results from using each combination of those 5 objects, from using a single object to using all 5.

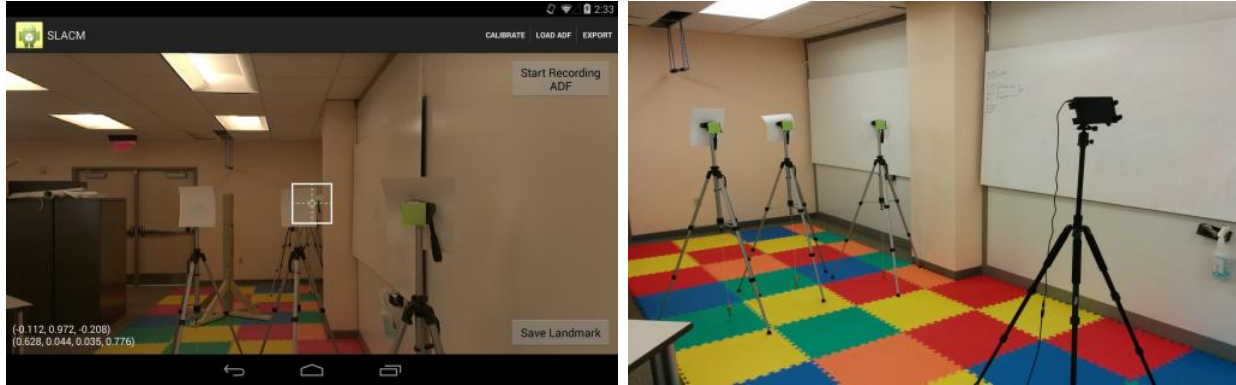


Figure 4.4: The laboratory setting of our experiments. Left: The UI of our Tango application. It provides a camera view for the user to point the tablet to the object to be positioned. Right: We use tripods to simulate users and objects in the scene. The user tripod (on the right) holds the Tango tablet, and others represent scene objects.

The results are tabulated in Figure 4.5. For each value of N , the corresponding bar represents the error distribution that arises when choosing N of the 5 objects for localization. In column 5, there is only a single data point, since there is only one way to choose 5 objects from a set of 5. Specifically, we have drawn a box and whisker plot that shows the minimum, Q1, Q2, Q3, and maximum error values that we determined from the experiment.

Note that when $N = 1$, the localization error shows a large variance. That variance decreases as N increases, which is to be expected, since with more objects used for localization, the more sure we should be that our calculated position is accurate. The lowest error for this particular experiment comes from combining the results of Objects 1 and 3, yielding an error of 3.9 cm, but unless the ground truth is known a priori, the user will not be able to know the ideal set of measurements. Given that it is typically intractable to determine the optimal set of objects to use, our recommendation is to simply average measurements from as many objects as is feasible given the circumstances. In Figure 4.5, the difference between the 3rd quartile and the 1st is about a centimeter for $N = 3$ and less for $N = 4$, so using $N = 3$ appears to be a good compromise between accuracy and user tedium.

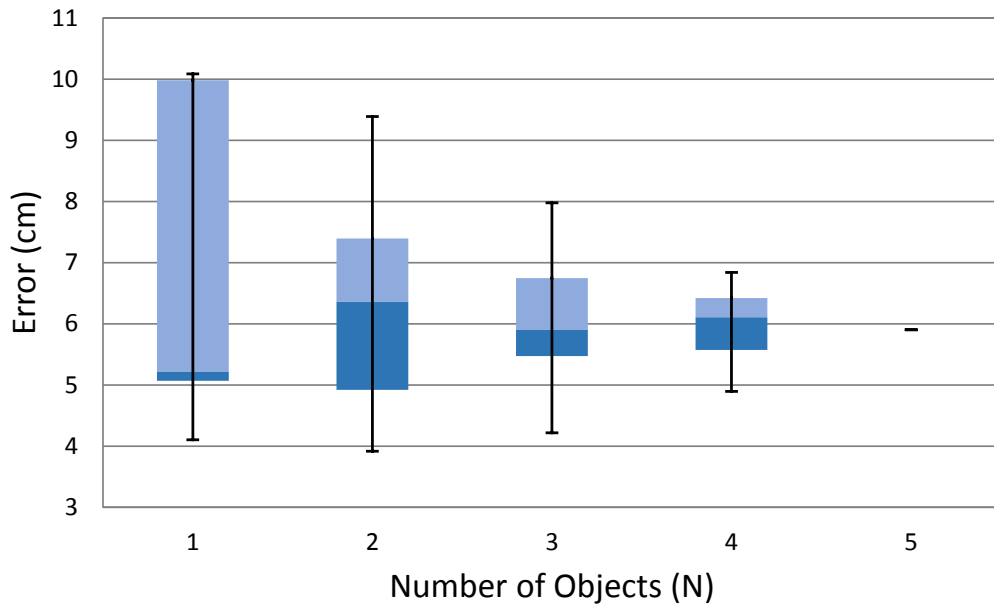


Figure 4.5: Using more objects to localize a single point tends to result in less variance.

4.6.3 Multi-hop Object Localization

While an object can be localized effectively using information from just a few objects, as seen in the previous section, the physical range is quite limited. In the case of the Tango tablet, it can be difficult to get depth measurements at a distance larger than about 3.0 m. In order to traverse greater distances, we will now evaluate the multi-hop chaining operation that was initially proposed in Section 4.4.

When sufficient care has been taken to reduce the effects of depth and rotational error, the slope of the error accumulation curve can be reduced very close to 0. At that point, the error that results from a single hop begins to lose any dependency on that of previous hops. This is useful because it effectively allows us to make as many hops as necessary without the need to worry about the accumulation of error along the way.

To demonstrate the lack of error accumulation, we performed an experiment in which the user was localized using the chaining approach. Only the initial location of the user was

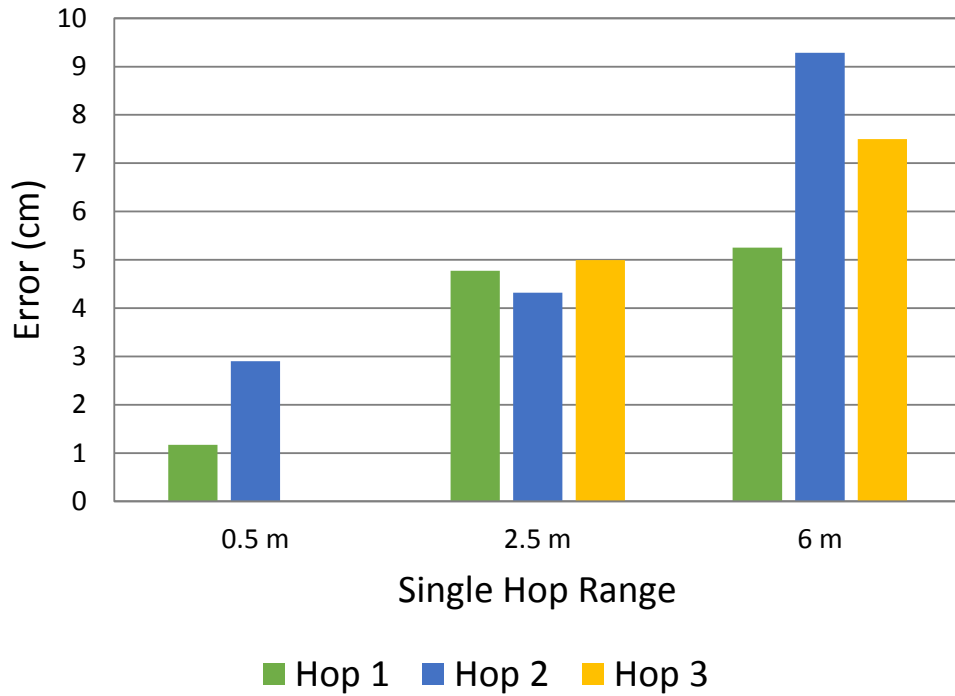


Figure 4.6: Error of several 3-hop chains with different ranges.

known. From that initial location, we localized 3 intermediate objects. The user moved to another location and we used those 3 intermediate objects to perform relocalization of the user. This sequence of operations constitutes a single hop. The process was then repeated with another set of intermediate objects and then once more for a total of 3 hops. This entire experiment was performed using 3 different ranges between hops, 0.5 m, 2.5 m, and 6.0 m, our assigned maximum. We were only able to perform two hops at the shortest range, so the final hop value is missing for that test.

Figure 4.6 shows the results of localizing the user after the first, second, and third hops under each set of conditions. The most important aspect to notice is that for both the 2.5 m and the 6.0 m cases, the error does not monotonically increase between the first hop and the last one. Unfortunately, there's not enough data to make a clear argument for the 0.5 case, so the pattern is not obvious at that range.

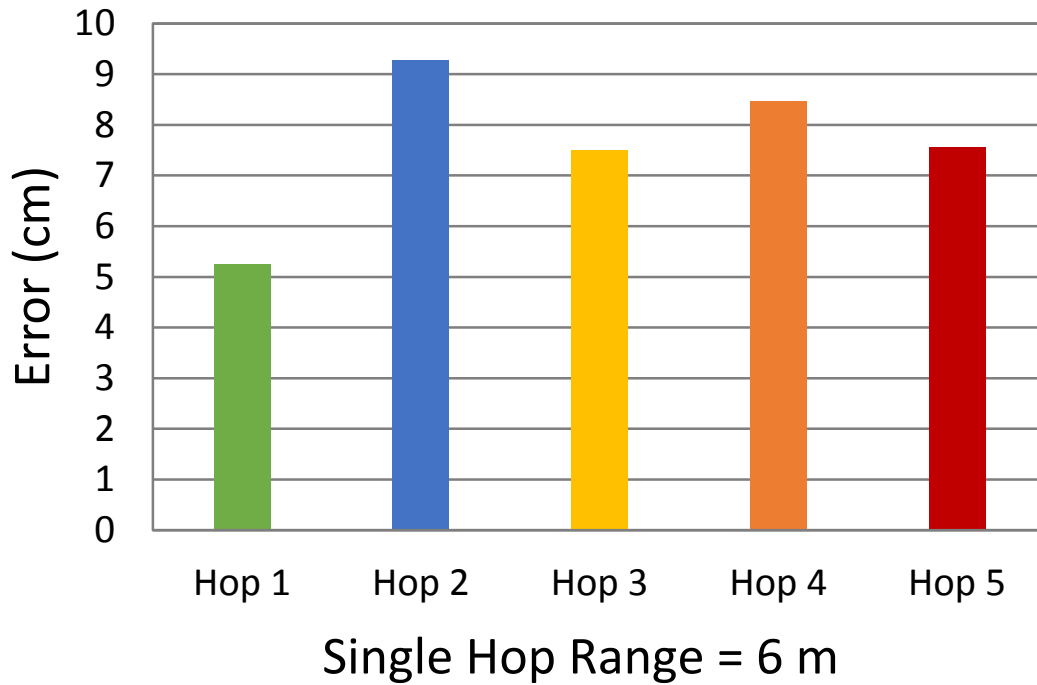


Figure 4.7: A long chain at long range. Note that the error does not monotonically increase between hops.

In the next experiment, we extended the number of hops to 5 for the long range (6.0 m per hop) test, giving a total range of roughly 30.0 m between the starting point and the final localization point. The results are shown in Figure 4.7. It can be seen the previous pattern has continued. The various error values are similar in magnitude, but there is not a clear upwards trend, which demonstrates that the error did not appreciably accumulate between hops.

4.6.4 Statistical Optimization

If there is a lot of uncertainty in the measurements, it is possible to improve on the chaining method by using the statistical optimization technique detailed in Section 4.5. We applied the technique to the same raw data that was used to generate Figure 4.6. The results are summarized in Figure 4.8.

In the short range case (0.5 m), applying optimization actually made the results appreciably worse. This is evidence that the error metric used for statistical optimization, SSD, is likely a poor fit when the range is severely limited. When the distance between hops was near the average value of 2.5 m, optimization provided a small benefit, but not necessarily a significant one. Statistical optimization is most useful when the distance between hops is large (6.0 m). Because increasing distance between objects causes the effects of rotational error to similarly increase, there is typically more uncertainty with those measurements than with the others. The optimization method is able to take advantage of that uncertainty in order to further reduce the error from one hop to the next, yielding better results than the chaining method alone in those circumstances.

As an intuition, for short- to mid-range applications, the chaining method provides results that are close to optimal already, and the optimization method is unable to exploit enough uncertainty to drastically improve the results further. The evident conclusion is to use the chaining approach alone if the distance between hops is fairly short (less than 3.0 m, for example) and to use optimization when the distance is larger, as the optimization method is more likely to appreciably increase the performance of the system.

4.7 Discussion and Limitations

DIPS can potentially enable a plethora of mobile applications that require accurate context localization. The most noticeable example is AR. For example, in supermarkets, navigation and augmented product information could be used to help shoppers find products that interest them. In shopping malls, augmented information such as discounts and featured products could also be displayed on AR headsets while users are looking around. Assisted living applications could also benefit from our technique. Critical contextual items, such as TV remotes, magazines, and glasses, could be localized immediately when a user intentionally

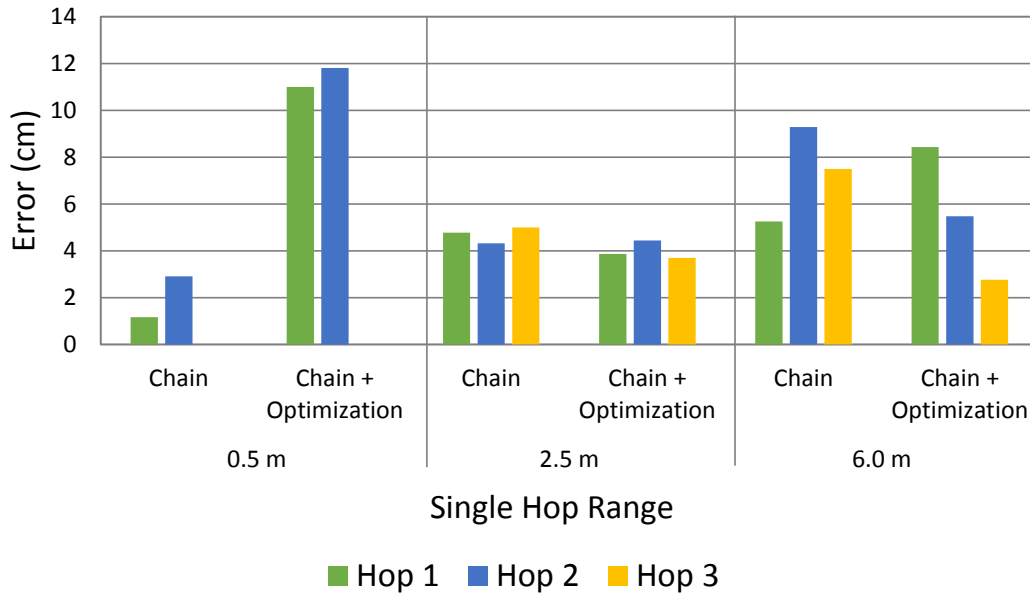


Figure 4.8: Statistical optimization tends to improve performance when the distance between objects is large.

or unintentionally gazes at them. When the user wants to find any of those items, the system could navigate the user to its precise location.

Despite the unique advantages our system provides, there are several limitations that should be mentioned. First, in our particular implementation, we used a Project Tango tablet that makes use of an IR-based depth sensor. Such sensors are known to operate suboptimally in an outdoor environment, meaning our implementation is currently only applicable to indoor areas. We do note, however, that our method does not have this restriction—just the implementation. The method would still be applicable if the IR-based depth sensor were replaced with another sensor better suited to outdoor environments. Second, our current system does require the user to actively gaze at an object in order to perform localization, which is an immediate drawback in terms of practicality. In the future, extensions that remove that requirement could be investigated, allowing the system to be used passively or opportunistically without any conscious cooperation from the user. Finally, the use of this

technology to enable augmented applications is not yet well-understood. Another area that could be further investigated in the future is the integration of accurate location data with AR platforms to enable novel mobile AR experiences.

4.8 Conclusion

In this chapter, we presented DIPS, an optical-based localization system for mobile platforms. The primary contributions of DIPS include a thorough theoretical analysis of the problem, a novel chaining approach capable of accurately localizing objects to within 10 cm at a range of 30 m or more, a statistical optimization to the chaining method that can further increase the accuracy of the system under certain circumstances, and a prototype implementation of the system using a Google Project Tango tablet evaluated under several real-world scenarios.

4.9 Future Work

Object positioning falls into the category of indoor localization techniques as it can be used to determine the location of the user at any given point, assuming one or more reference points are available and the user is willing to accept a small latency while the system performs localization. If continuous estimates of the user's position are desired, we could make further use of the inertial sensors on the device.

Since our system produces accurate estimates at low frequencies and inertial sensors can be used to produce less accurate estimates at high frequencies (due to sensor drift), we should be able to combine those estimates (e.g. using a Kalman or Particle Filter) in order to obtain reasonably accurate location estimates at any point in time.

5 Learning Resolution-independent Image Representations

Humans are well-known to be highly effective at comprehending continuous patterns within digital images. In this chapter, we present a collection of methods that enable analogous capabilities in deep neural networks. These methods train neural networks to represent images with continuous resolution-independent representations. They utilize an MCMC algorithm that directs attention during the learning phase to regions of the image that deviate from the current model. An encoding hypernetwork learns to generalize from a collection of images, such that it can effectively compute resolution-independent representations in constant time. These methods have immediate applications in super-resolution scaling of images, image compression, and secure image processing, and additionally suggest improved capabilities for image processing with neural networks in several future applications.

5.1 Introduction

The pixel values in a digital image may be viewed as samples drawn from a function of the form $f(x) = i$, where $x \in \mathbb{R}^2$ represents the image coordinates, $i \in \mathbb{R}^c$ is the corresponding intensity for that coordinate, and c is the number of channels of the image. A digital camera does not measure f directly, but samples it at uniform intervals. The discrete sequence $\{i_{0,0}, i_{0,1}, \dots, i_{H-1,W-1}\}$ of intensity values is only a representation of the underlying continuous image from which it is sampled.

Although generally useful for computer graphics, the discrete representation does have several important drawbacks. For example, in order to change the width or height of the image, the discrete representation must be resampled. In practice, the ability to do so directly is often limited (e.g. the digital camera does not have the ability, the subject

has moved, etc.), so new intensity values must be synthesized using the pixels that have already been captured, adding additional noise. The discrete representation also has a space requirement proportional to the number of samples taken. Larger images require more space, which limits their utility in practice.

A continuous representation of the image would not suffer from these drawbacks, but since f is not typically available, a learned approximation, \hat{f} , would have to be used instead. Such an approximation can be learned by examining the pixels of the discrete image representation as long as the underlying model has sufficient capacity.

A popular choice of learning model is the neural network, especially a multilayer perceptron (MLP). Assuming several conditions are met, MLPs have been shown to be universal function approximators [48]. As such, they possess the capability to accurately model f . We define $\hat{f}(\theta, x) = i$ to be an MLP that maps normalized image coordinate pairs x to a vector of intensity values i , where $i \in \mathbb{R}^c$. When fully trained, $\hat{f}(\theta, x) \approx f(x) \forall x \in \mathbb{R}^2$. In practice, we limit ourselves to the range $x \in [0, 1]^2$. Given θ , the intensity for any individual coordinate can be approximated by performing a feed-forward pass through \hat{f} . Similarly, an entire image can be constructed by feeding multiple coordinates in as a batch.

Importantly, if θ has been provided, \hat{f} can be used to generate discrete representations of f with arbitrary pixel sampling frequencies. In other words, images can be generated at any resolution. As a result, θ itself constitutes an alternative representation of the original image f , one that is invariant with respect to the sampling resolution.

The parameters θ of \hat{f} can be learned for any given image using traditional gradient descent-based techniques. A set of $N \langle x_i, f(x_i) \rangle$ tuples can be obtained directly from the pixels of the available image, where N is the number of pixels. This information can be used as training data for an ordinary regression model.

Interestingly, it is also possible to train another model, we call the **encoder**, to

predict a reasonable image encoding θ for a single image. The encoder is given as input all available intensity values from an original image and outputs the set of parameters to be used by \hat{f} . As the encoder will output the weights and biases of another network, it can be categorized as a type of hypernetwork [90]. The encoder is trained on a collection of images based on a reconstructive error metric utilizing \hat{f} . The encoder will learn to recognize features that are common to multiple images and associate them with the parameters of \hat{f} necessary to properly reconstruct the image that was provided as input, even providing reasonable encodings of images it has never seen.

In this work, we focus on two important questions: 1) How is θ calculated for a given image? 2) What is the utility of a sampling-independent image representation? As a response, our primary contributions are as follows:

- We show that θ can be learned using the discrete representation of an arbitrary image using traditional gradient-descent.
- We demonstrate an MCMC-based technique that yields improved reconstruction accuracy and lowers training time compared to standard batch processing of pixels.
- We present a deep convolutional encoder that is capable of generating reasonable θ values for unknown images using a single forward pass through the network. Our encoder generates the weights of \hat{f} directly, rather than requiring extensive training for each image.
- We demonstrate several practical applications of our work, including image resizing, compression, and security.

This chapter is outlined as follows: In section 5.2, we discuss the relevant works presented by others in this domain. In section 5.3, we detail the process by which θ can be

learned iteratively for a single image. In section 5.4, we show how a generalized encoder can be trained to output useful image representations directly. In section 5.5, we demonstrate several applications of a resolution-invariant image representation, including arbitrary scaling, compression, and security. In section 5.6, we evaluate our claims using single images and groups of related images. Section 5.7 concludes our work.

5.2 Related Works

Neural networks have been used for several state-of-the-art applications in image processing, including recognition [36, 37, 38], completion [41, 42], style transfer [39, 40], and generation, especially using Variational Autoencoders [43] and Generative Adversarial Networks [44, 45]. Many of these approaches create approximate image representations implicitly in order to accomplish their stated goals. In our work, we focus entirely on the problem of generating reasonable representations of images.

Several other works have put more focus on learning image representations, including PixelRNN [91], which uses a recurrent network to generate pixels one a time, and Ashmore et. al [92], who suggested learning image representations in order to capture state from either a single image or a sequence of images. Our approach differs from PixelRNN in that we can generate an entire image in parallel, as our model does not make use of recurrent connections (e.g. LSTM). Ashmore’s approach uses a type of autoencoder to learn image state, while our approach more closely resembles a hypernetwork.

5.2.1 Hypernetworks

A hypernetwork refers to a neural network capable of generating the weights for another neural network. For example, Ha et al. used hypernetworks to generate adaptive weights for recurrent neural networks [90]. Stanley used an approach called HyperNEAT to

make a neural network that was highly amenable for evolving images [93]. In our work, we train a hypernetwork to learn how to approximate resolution-independent representations of images from pixel values. More specifically, our hypernetwork learns from many images how to comprehend what is implied by the discrete sampling of pixels in digital images.

5.2.2 Super-resolution

Recently, the problem of resizing an arbitrary image, especially to increase the resolution, has been addressed by deep convolutional neural networks [94, 95]. For example, Dong et. al [94] demonstrated that super-resolution can be achieved for a single image by training a deep network to map between low and high resolution versions of that image. One application of our work is super-resolution, as an image can be reconstructed using any arbitrary resolution, including larger sizes. Our work differs in that a higher resolution version of the original image is not required for training. Our work also produces results a single pixel at a time, rather than one image at a time. This allows the networks to be significantly smaller and faster to evaluate.

Generative adversarial networks have also been applied to this problem. For example, Ledig et. al [96] have demonstrated that GANs can effectively generate the fine image texture details that are often missing from other super-resolution approaches, which indicates that they are capable of accurately approximating the original image. Our work differs in that we can train networks and produce scaled results using images of any arbitrary resolutions. As such, we are not constrained to any particular scale (e.g. 4x larger).

5.2.3 Compression

Although not our primary focus, one tangential application of our research is the ability to compress images with reasonable effectiveness. Several others have also used neural

networks to compress images, including Toderici et. al [97]. These approaches tend to rely on learning some form of mapping between the original image and the compressed version in order to maximize compression ratios. Our approach is conceptually much simpler, as θ directly corresponds to the compressed image representation. Due to the simplicity, however, our method is unlikely to outperform established SOA techniques in this particular field.

5.3 Learning a Representation for Single Images

5.3.1 Formulation

Training an MLP to fit to a single image f can be a relatively straightforward process when the problem has been well-formulated. The goal is to find a setting of the parameters of the network θ_f that minimizes some reconstruction error E with respect to the pixels of the original image. More formally,

$$\theta_f = \operatorname{argmin}_{\theta_f} (E[\hat{f}(x, \theta_f), f(x)]) \quad (5.1)$$

In this equation, E refers to some error metric, and x represents a vector in \mathbb{R}^2 containing the normalized image coordinates for a single pixel, scaled such that $x = (0, 0)$ and $x = (1, 1)$ refers to the top-left and bottom-right corners of the image, respectively. Both $\hat{f}(x, \theta_f)$ and $f(x)$ produce a value in \mathbb{R}^c , representing a c channel intensity (e.g. 3 for RGB, 1 for grayscale, etc.).

If E is differentiable, equation 5.1 can be evaluated using traditional gradient descent-based techniques. For each pixel, the appropriate image coordinate vector x is calculated and used as the input to the model. The pixel intensity values themselves are used as labels. If an image contains N pixel values, then at most N unique training samples are available for training \hat{f} .

5.3.2 Error Metrics

While the choice of error metric E is theoretically arbitrary, certain metrics work better than others in practice. For example, in our testing, variants of Sum-squared Error (SSE) seem to be highly susceptible to a local optimum in which the network learns to output a blank image (all intensities are either 0 or 1). Experimentally, we found that using Cross-Entropy Error does not have these limitations and indeed does perform well for most images. Therefore, the remainder of this paper assumes that Cross-Entropy Error is used for E .

5.3.3 Batching

As is common when training neural networks for classification or regression, equation 5.1 may also be formulated in terms of matrices in order to evaluate several image coordinates simultaneously:

$$\theta_f = \operatorname{argmin}_{\theta_f} (E[\hat{f}(X, \theta_f), f(X)]) \quad (5.2)$$

X then is a $B \times 2$ matrix, both $f(X)$ and $\hat{f}(X, \theta_f)$ are $B \times c$ matrices, and $B < N$ is the batch size. When batching is used, E is the mean Cross-Entropy Error across all pixels in the batch.

5.3.4 Model

The topology of \hat{f} controls the degree to which it is capable of approximating f . If the topology is too restrictive, \hat{f} will not be able to capture the fine details of the original image, resulting in a blurred reconstruction. Conversely, as we will discuss further in sections 5.4 and 5.5, it is desirable to have as small a representation of θ_f as possible. Ideally, the number of weights would be proportional to the entropy of the image in order to guarantee

the network has the capacity necessary to fully represent f , but a small fixed topology can be effective in practice. For example, a $2 \rightarrow 100 \rightarrow 50 \rightarrow c$ fully connected topology with tanh or relu nonlinearities is sufficient to represent many low resolution images.

The choice of nonlinearity has an affect on the quality of the reconstruction when training is not fully converged. A tanh nonlinearity tends to produce smoother images than a relu nonlinearity, for example. Upon convergence, however, the choice has demonstrated much less significance.

5.3.5 Training

\hat{f} can be trained using mini-batch gradient descent, where the batch size varies between 1 and N , where N is the number of pixels in the image. There are several different strategies that could be used to select the pixels to use in each mini-batch: A naïve approach would be to group pixels into blocks of a certain width and height. The pixels within a particular block would always be part of the same batch, and blocks could be chosen for training randomly. A slightly more sophisticated solution would allow for overlapping blocks by adjusting the horizontal and vertical strides from one block to the next. This approach tends to compromise between learning high-frequency and low-frequency image components based on the size of the blocks. As a result, it tends to converge slowly.

A related method is to choose pixels from the image randomly to fill each mini-batch. Random sampling allows the network to efficiently learn the low-frequency components of an image due to the potentially large spacial separation between the pixels in the mini-batch. As a result, this method is likely to converge more quickly than the block-based method at the beginning of training. However, since the separation between pixels is typically much larger than the block-based approach, learning high-frequency image components can be more difficult when using random sampling. This effect tends to slow progress once the

Algorithm 2 Applying the Metropolis-Hastings algorithm to pixel selection

```
function NEXT-BATCH-METROPOLIS( $f, \hat{f}, b_{prev}, \sigma, \gamma$ )  
   $b \leftarrow b_{prev}$   
  for  $i \in [0, len(b) - 1]$  do  
     $c_i \leftarrow b_i + \mathcal{N}(0, \sigma)$   
     $p(b_i) \leftarrow |f(b_i) - \hat{f}(b_i)|^\gamma$   
     $p(c_i) \leftarrow |f(c_i) - \hat{f}(c_i)|^\gamma$   
    if  $\mathcal{U}(0, 1) < \frac{p(b_i)}{p(c_i)}$  then  
       $b_i \leftarrow c_i$   
    end if  
  end for  
  return  $b$   
end function
```

low-frequency portions of the image have been learned.

A potential solution to this problem takes the image structure into account. Once low-frequency parts of the image have been learned, pixels that can already be reconstructed accurately should not be weighted the same as pixels that are far from their correct intensities. Training would be more efficient if those pixels with large reconstruction error were more likely to be placed in a batch than those with small reconstruction error. Many machine learning techniques based on boosting use a similar rationale.

Pixel weights can be calculated for an entire image by forward-propagating each pixel coordinate, calculating an appropriate reconstruction error for each pixel (e.g. $|y - \hat{y}|^\gamma$), and then normalizing those errors to form a valid probability distribution. This particular approach tends to be expensive in practice because the pixel weights need to be updated frequently, ideally after each mini-batch presentation, and calculating the pixel weights requires a forward propagation of the entire image.

We propose a significantly more efficient alternative approach based on the Metropolis-Hastings (MH) algorithm [98], a Markov Chain Monte Carlo (MCMC) approach often used to generate random samples from an arbitrary probability distribution. The first batch contains pixels chosen uniformly from the original image, but each batch afterwards is generated

using the process outlined in Algorithm 2. For each element in the batch, we generate a candidate pixel in the same neighborhood as the original. We then calculate reconstruction errors for both the original pixel and the candidate. (We use $\gamma = 0.125$ for our experiments.) These are used as estimates of the probability density function required by the MH algorithm. Finally, we apply the MH algorithm directly, choosing probabilistically whether or not to replace the original pixel with the candidate.

This algorithm tends to cause training to focus training on high-frequency portions of the image, such as edges. As those pixels are better represented in the training batches, the network is given more motivation to correctly reconstruct detailed portions of the image. Especially when combined with the random-pixel method to learn low-frequency portions of the image, this MCMC-based approach can lead to faster convergence throughout training and lower overall reconstructive error compared to either of the other methods alone.

5.4 Learning a Generic Encoder

The previous section presented an iterative process for training a network to generate a resolution-independent image representation. This section generalizes upon that capability by demonstrating that we can train an encoding hyper-network, e , to compute θ_f . This encoder offers significantly greater utility by computing θ_f in a single forward-pass, and learns to generalize effectively from multiple images. e maps from the pixels of the original image to a resolution-independent representation, θ_f . With this formulation, \hat{f} no longer needs to be trained directly. Instead it is used as part of an objective function to train the parameters of e , θ_e . θ_e then, represents the entire set of variables that need to be optimized.

Here, we show that e can be trained by examining numerous images, extracting common features and mapping them to image representations that can then be used by \hat{f} directly, without the need for additional training at inference time. As a result, if e

has already been sufficiently trained, θ_f can be calculated for an arbitrary image using a single forward pass through the encoder network. Compared to the approach outlined in the previous section, this approach is asymptotically faster. Evaluating e for a single input is a constant time operation, compared to the linear cost of fully training a network to accomplish the same goal, which is clearly an improvement.

In addition to the methods outlined in the previous section, additional procedures unique to this problem are also needed in order to generate high quality encodings across multiple images. In this section, we examine these additional procedures, as well as the architecture of the complete encoding hyper-network in detail.

5.4.1 Model Architecture

Deep neural networks, especially convolutional ones, have demonstrated that they are capable of extracting high-level features from images, especially for the purposes of image recognition, segmentation, and generation [36, 37, 38, 43, 44, 45]. The same properties are desirable in creating a generic encoder, as we desire to associate image features with appropriate image representations. As a result, we will use a deep convolutional network as the basis for our generic encoder.

The architecture of our model is shown in Figure 5.1. We use several layers of convolution, separated by rectified linear activation units and 2×2 max-pooling operations to extract and downsample important image features. We then attach two fully connected layers separated by additional relu activations to map between the extracted features and the appropriate image representation θ_f .

Adding a layer of activation units after the output layer would have the effect of enforcing constraints on the weights of the MLP \hat{f} . For example, a tanh activation layer would constrain all values of θ_f to fall in the range $[-1, 1]$, providing a form of regularization.

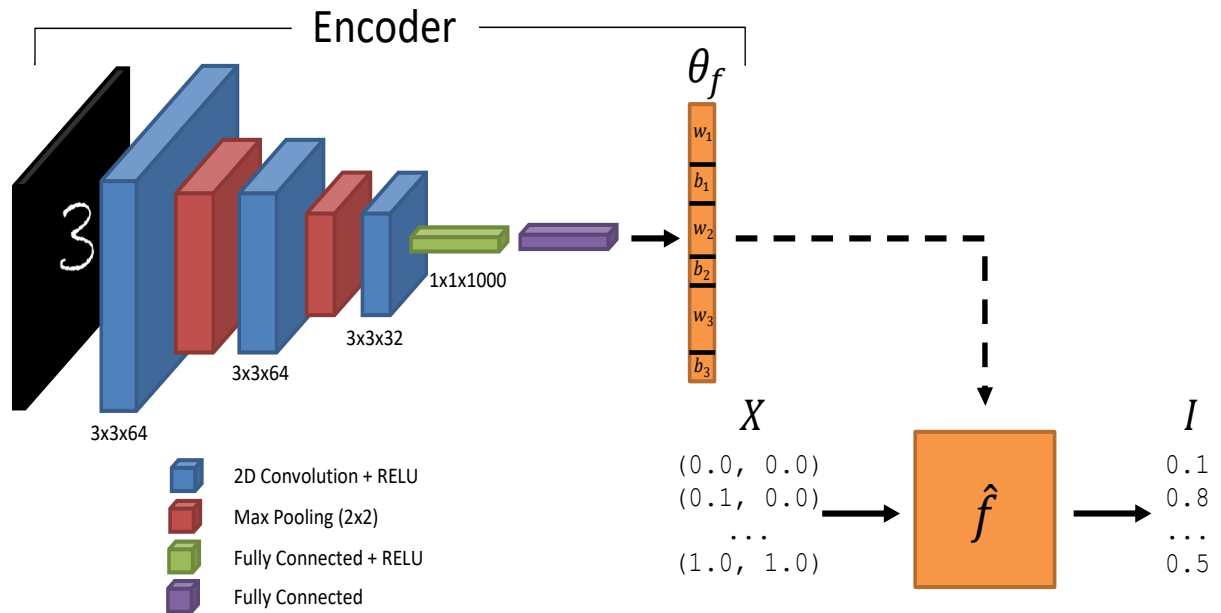


Figure 5.1: Architecture of the encoder. We use convolution and max pooling to identify important image features and fully connected layers to learn the mapping to θ_f .

However, in order to provide e with maximal flexibility to reconstruct images, we have chosen to omit such an activation layer in our model. Studying the effects of different activation layers is left as an area for future work.

5.4.2 Training

In order to train e directly, we would need to have access to a dataset mapping images to an appropriate representation, θ_f . While such a dataset could be generated given a large enough collection of images using the techniques outlined in Section 5.3, doing so would require training a network to completion for each image in that collection, which is not practical.

Instead, we use the function \hat{f} as an objective metric to guide training directly. We select an image from the training collection and forward-propagate it through e to obtain an initial image encoding θ_f . That encoding is split into individual vectors, each of which is reshaped to form the weight and bias matrices used by \hat{f} . \hat{f} is then evaluated using a mini-

batch of pixels selected from the chosen image, producing a $B \times c$ matrix of reconstructed pixels. As in Section 5.3, we calculate the gradient of some reconstruction error metric (e.g. Cross-Entropy) with respect to each of the components of θ_f , which is then backpropagated back into e in order to update the encoder's parameters θ_e . We repeat the process K times using the MCMC algorithm presented in Section 5.3 to select which pixels to use in each mini-batch. After the K iterations have passed, another image from the training set is selected, and the algorithm repeats until e produces encodings of sufficient quality.

There is an interesting interaction between the hyperparameter K and the choice of optimizer used during training. If e is trained for too long (or too well) on a single image, it can learn to produce encodings that do not generalize effectively. For example, e might learn to memorize a good encoding for the most recently presented image or to produce a uniform encoding for all images, introducing significant reconstruction error. We found that by using smaller values of K (e.g. $K = 10$) and by making use of slightly weaker optimizers (e.g. RMSProp [99] instead of ADAM [100]), we can counteract these types of issues in practice.

5.5 Applications

There are several practical applications for resolution-independent image representations, including, but not limited to, image resizing, compression, and security. In this section, we survey these applications.

5.5.1 Image Resizing

A straightforward application of our research is the ability to resample images to arbitrary resolutions. This is achieved by feeding a different set of relative pixel coordinates x' to \hat{f} than the model was initially trained on. If W_d and H_d represent the desired width and height in pixels, x' can be calculated as:

$$\begin{aligned}
h &= \left\lfloor \frac{i}{W_d} \right\rfloor \\
w &= i \bmod W_d \\
x'_i &= \left[\frac{w}{W_d - 1}, \frac{h}{H_d - 1} \right], \text{ where } 0 \leq i < W_d \times H_d
\end{aligned} \tag{5.3}$$

Indeed, we demonstrate in Section 5.6 that simple images such as MNIST digits can easily be rescaled roughly 20x larger, from 28 x 28 pixels to 512 x 512 pixels, with little perceptual loss in quality, while larger images can be comfortably be scaled by a lesser amount.

5.5.2 Compression

Another interesting application of this research is the ability to efficiently compress images, especially higher-resolution ones. Our generation network \hat{f} is designed to encode images using relatively few weights (e.g. 5,503 for the topology given in Section 5.3 with 3 channels). For the purposes of storage or transmission, only those weights need to be persisted, rather than each of the individual pixels of the original image. As a result, the resolution-independent encoding can often be smaller than the alternative representation.

For example, the famous “Lenna” image often used with Image Processing research is a 220 x 220 pixel RGB image, requiring 145,200 bytes uncompressed (220 x 220 x 3 bytes per pixel). The corresponding scale-independent representation would require 22,012 bytes uncompressed (5,503 weights x 4 bytes per weight), an 84% reduction in size or roughly a 7:1 compression ratio. For comparison, a standard JPEG compressed version of the same image with the highest quality level requires 47,145 bytes, which is a 67% reduction in size or a 3:1 compression ratio.

With many lossy compression algorithms, ours included, a tradeoff can be made

between file size and reconstruction quality. By limiting the topology of \hat{f} , we can simultaneously constrain the capacity of the neural network while introducing additional reconstruction error. In Section 5.6, we will demonstrate the results of a more complete compression test, showing how as the size of the network topology correlates positively to the reconstruction quality.

5.5.3 Security

A final application of our technique is relevant to information security, particularly as an additional layer of “security through obscurity”. Sensitive images that have been encoded to a resolution-independent format would not be able to be viewed by a malicious entity without understanding the significance of the values, similarly to how most binary formats cannot easily be read without understanding the file format in which they were saved.

Assuming an attacker did understand that the individual bits of an image’s resolution-independent representation could be interpreted as 4-byte floating point values, those numbers would still be meaningless without the associated network topology, which would not necessarily have to be encoded into the file format itself, and an understanding of what the inputs and outputs to the network represent.

As resolution-independent representations are not currently commonplace in many real-world scenarios, entities interested in preserving the privacy of their users could adopt our technique as an additional line of defense beyond other orthogonal approaches, such as strong encryption or multi-factor authentication.



Figure 5.2: A network has been trained on each individual image to generate a corresponding resolution-independent encoding. Images were then reconstructed at the original resolution.

5.6 Evaluation

In this section, we validate the claims that have been made so far in the paper. Firstly, we demonstrate that a resolution-independent image encoding can be learned for a single image using a small MLP. Then we demonstrate our results for training a single generic encoder that outputs reasonable image encodings directly. Next, we compare our resampling approach to several others in common use. Finally, we perform a comparison between the size of the resolution-independent encoding and reconstructive accuracy, showing that as the two are positively correlated.

For our evaluation, we will use examples from the MNIST database of handwritten digits, photo #6 from the Kodak dataset, and the famous “Lenna” photo often used in image processing. The resolutions of each (in pixels) are 28 x 28, 192 x 128, and 220 x 220, respectively. The MNIST images are greyscale, while the others are traditional RGB images.

5.6.1 Resolution-independent Image Encodings

We first demonstrate that the techniques discussed in Section 5.3 can be applied to learn a resolution-independent image encoding directly. For this experiment, a small network was trained on a single image from the MNIST dataset of handwritten digits using the ADAM optimizer and a learning rate of 0.001 for 50,000 iterations. This network was then used to reconstruct the image used for training. We repeated the process for the first

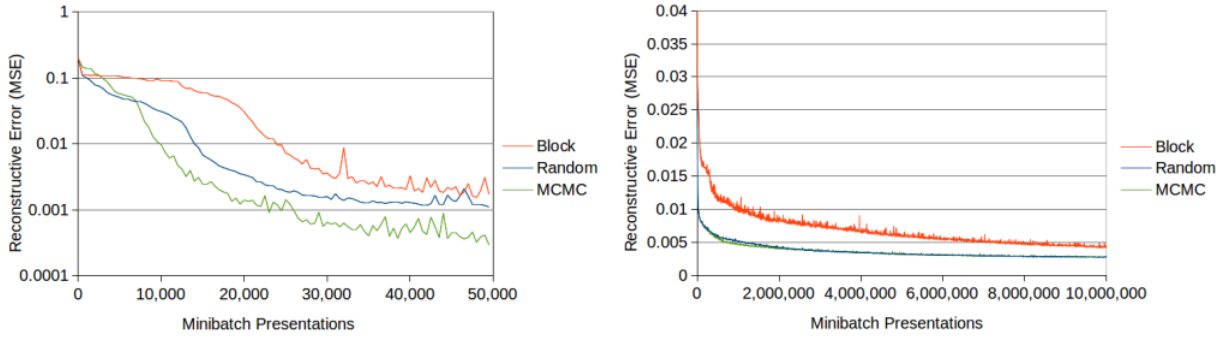
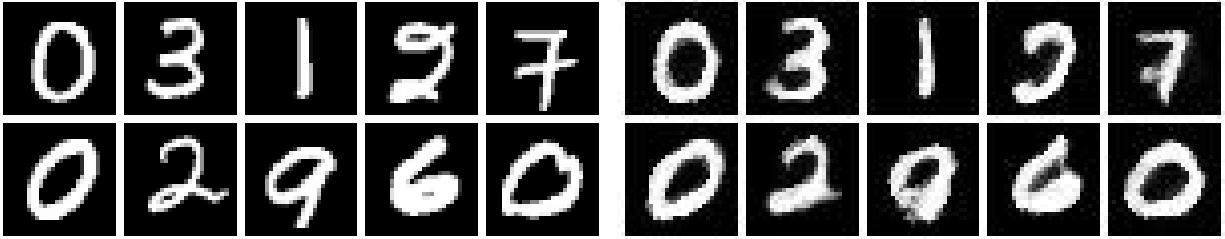


Figure 5.3: A comparison of three batching methods on two different images. Left: an arbitrary example from the MNIST dataset. Right: Lenna.

20 elements of the MNIST dataset to generate the entries in Figure 5.2. Notice that simple images can be reconstructed relatively easily and that most images show little perceptible error.

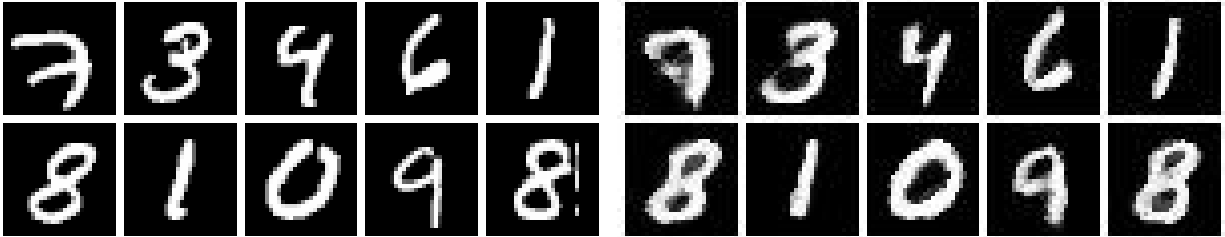
Next we show how the choice of batching method affects the convergence of training. In Section 5.3, three alternative methods were proposed: block training, random pixel sampling, and an MCMC-based approach. For this test, two individual models were created, one for a single MNIST image, and another for the Lenna image. Both networks were trained to convergence using each of the three approaches. For the MCMC-based approach, several iterations of random sampling were used at the beginning of training before switching to the MCMC algorithm as discussed in Section 5.3. The resulting training curves are given in Figure 5.3. Note that the figure on the left is displayed in log-scale to better visualize the differences between each method.

In both cases, block sampling proved to be the least effective of the three approaches, as it converged slower and had a higher resulting reconstruction error. The MCMC-based approach clearly outperformed random sampling for the MNIST test, but the two approaches produced virtually identical results for the Lenna test. We believe the MCMC method tends to perform better when there are steep color gradients as opposed to shallow ones, as demonstrated by its performance on the MNIST image.



(a) The first 10 training images.

(b) Reconstructions of the first 10 training images.



(c) The 10 testing images.

(d) Reconstructions of the 10 testing images

Figure 5.4: A generic encoder was trained on 10,000 images from the MNIST dataset. Reconstructions were produced without any additional training.

5.6.2 General Image Encoder

In this section, we evaluate the performance of our generic encoder. For this test, an encoder with the architecture given in Figure 5.1 was created and trained on 10,000 images from the MNIST dataset using the training procedure outlined in Section 5.4. That encoder was then used to recreate 10 additional images that were not part of the training set at their original resolutions, as well as the first 10 images in the training set. The results are given in Figure 5.4.

We see that the encoder was generally able to provide image encodings that allowed for good reconstructions of both the images that were seen previously and those that were not. Importantly, it seems that some of the imperfections present in several of the images were effectively corrected by the encoder. For example, the digit 8 in row 2 appears to have a ! symbol next to it in the original image, but that symbol was removed by the encoder. There were also holes in each of the two previous examples (the 0 and 9) that were



(a) NN Interpolation (b) Linear Interpolation (c) Cubic Interpolation (d) Sinc Interpolation (e) Our Approach

Figure 5.5: Various means of upscaling an element from the MNIST database. Images were increased in resolution from 28 x 28 pixels to 512 x 512 pixels.

filled during reconstruction. This demonstrates that the encoder does appear to have some understanding about the content of the images, rather than simply memorizing the encoding space.

5.6.3 Scaling

As mentioned in Section 5.5, one application of our work is the ability to resample an image at a higher resolutions. To demonstrate the utility, Figure 5.5 shows how various interpolation algorithms perform on one particular element of the MNIST dataset. For this experiment, the original image, which has a resolution of 28 x 28 pixels, was upscaled to 512 x 512 pixels (roughly 20x larger), using various common methods for image interpolation. Our method captures the unique features of the original image while avoiding unnecessary blurring.

5.6.4 Encoding Size and Reconstruction Error

For our last experiment, we compare how the size of θ affects reconstructive quality. For this test, we trained several networks to reproduce three images: an example from the MNIST database, photo #6 from the Kodak dataset, and the Lenna image. The size of the topology of \hat{f} was varied to produce larger (and more expressive) encodings. The

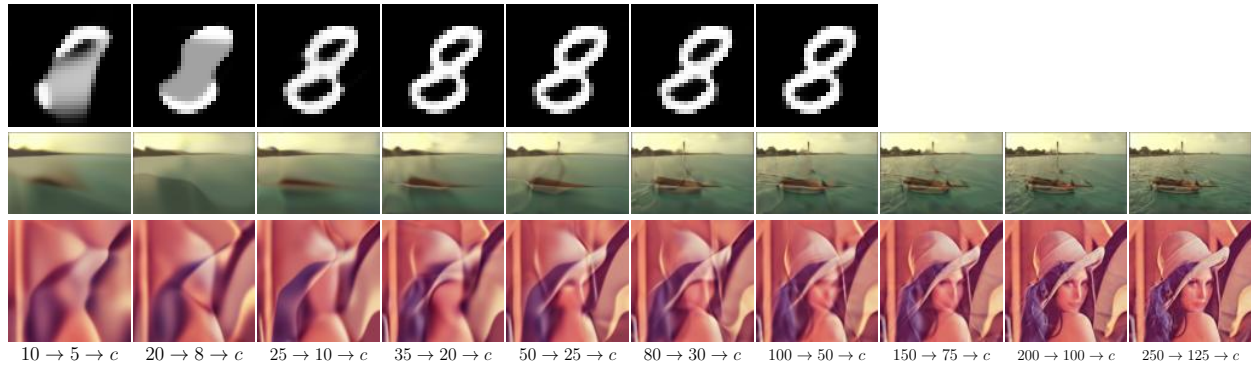


Figure 5.6: Larger topologies yield better image reconstructions. From top: an example from the MNIST dataset, Kodak #6, Lenna, the topology of \hat{f} using $\tanh()$ activations.

results are given in Figure 5.6. Intuitively, as the size of the encoding increases, so does the reconstructive power of the network. Relatively small topologies are sufficient to reconstruct images clearly.

5.7 Conclusion

In this chapter, we demonstrated a process for learning resolution-independent image encodings. We also demonstrated that a deep convolutional encoder can be trained to produce reasonable encodings for images, avoiding the cost of training a complete neural network for each individual image. We validated our claims with well-known standard datasets and images. These new methods offer capabilities in such areas as super-resolution scaling, image compression, secure image processing, and other image processing applications.

6 Summary

In this dissertation, we have explored methods for enhancing the context-awareness capabilities of modern computers, including mobile devices, tablets, wearables, and traditional computers. Advancements include proposed methods for fusing information from multiple logical sensors, localizing nearby objects using depth sensors, and building models to better understand the content of 2D images.

First we proposed a system called Unagi designed to incorporate multiple logical sensors into a single framework that allows context-aware application developers to easily test new ideas and create novel experiences. Unagi is responsible for collecting data, extracting features, and building personalized models for each individual user. We demonstrated the utility of the system with two applications: adaptive notification filtering and a network content prefetcher. We then thoroughly evaluated the system with respect to predictive accuracy, temporal delay, and power consumption.

While Unagi can significantly simplify the development of certain context-aware applications, it also faces several important limitations that may provide fruitful avenues for further research. For example, as the energy cost of accessing hardware sensors on a mobile device is generally not negligible, we cannot trivially extend Unagi's methodology to work with them. Hybrid sensing techniques may have the potential to collect highly relevant contextual information while avoiding many of these associated costs.

Next we discussed a set of techniques that can be used to accurately determine the location of objects near a user in 3D space using a mobile device equipped with both depth and inertial sensors. Using a novel chaining approach, we were able to locate objects farther away than the standard range of the depth sensor without compromising localization accu-

racy. Empirical testing showed our method was capable of localizing objects 30 m from the user with an error of less than 10 cm.

An important aspect of our DIPS project is that it is designed to obtain a single highly accurate location estimate, rather than a continuous stream of events. By combining information from other sources, such as inertial sensors, it is possible to use this work to create a more complete localization system that could be implemented using inexpensive consumer devices. Inexpensive localization, especially in an indoor environment, has long been a goal in mobile computing, so such a step could potentially be valuable for both research and for end users.

For our final topic, we demonstrated a set of techniques that allow a multi-layer perceptron (MLP) to learn resolution-invariant representations of 2D images, including the proposal of an MCMC-based technique to improve the selection of pixels for mini-batches used for training. We also showed that a deep convolutional encoder could be trained to output a resolution-independent representation in constant time, improving the utility considerably. Lastly, we discussed several potential applications of this research, including image resampling, image compression, and security.

A key challenge with work like this is the ability to “explain” the motivations behind actions taken by a neural network. For example, our resolution-independent image representation can reconstitute an image, but we can not currently alter the reconstruction in any semantically relevant way by adjusting individual parameters of the representation. For example, we cannot easily create a representation for the Paris skyline and then intentionally remove the Eiffel Tower in the reconstruction. Questions like these provide the motivation for a multitude of research topics in Computer Vision and Machine Learning that will prove invaluable for other fields as well.

6.1 Final Thoughts

Each of the individual topics we've discussed so far represents an important advancement in the field of context-aware computing.

Frameworks like Unagi are necessary to simplify the development of applications that can react appropriately to changes in the user or in the user's environment. These applications represent the realization of the efforts made by the research community to provide tangible benefits to our society. Without them, the research has diminished utility. Simplifying the process of creating context-aware applications then, has value to both end users and to researchers in the field.

Another important means to serve the research community is to explore uncommon data sources, such as the psychological or low-cost depth sensors we have addressed in this dissertation. Our goal with this sort of research is to better understand how the raw data can best be used to accomplish interesting tasks, such as determining how users are feeling or localizing objects near them efficiently. These experiments can be used to inspire other researchers to explore related methods, contributing to the natural growth of the field.

A final method of advancement comes from exploring other research domains and adapting topics, methods, and technologies to solve similar problems. Our work, for example, fuses Systems Design, Hardware and Software Sensing, Computer Vision, and Machine Learning with context-aware computing to advance our research. The perspectives offered by studying each of those domains have proven to be invaluable when deciding how to approach each of our research questions.

Future work in the field of context-aware computing will be heavily influenced by each of these three methods of advancement. As context-aware applications become more common and more powerful, a significant amount of engineering effort will be put into creating

high-quality experiences for users, necessitating useful programming interfaces. Additional data sources, particularly software-based ones that require almost no additional power consumption, will also be very important moving forward, as they tend to correlate very highly with user behaviors. Finally, collaboration between researchers in other fields will be invaluable in order to address new limitations as they are uncovered. New problems in one domain can often benefit from established solutions in another.

Bibliography

- [1] B. Schilit, N. Adams, and R. Want, "Context-aware computing applications," in *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*. IEEE, 1994, pp. 85–90.
- [2] L. Bao and S. S. Intille, "Activity recognition from user-annotated acceleration data," in *Pervasive Computing*, 2004.
- [3] D. Zhang, C. Chen, Z. Zhou, and B. Li, "Identifying logical location via gps-enabled mobile phone and wearable camera," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 26, no. 08, p. 1260007, 2012.
- [4] J. C. Hammer, "Enabling usage pattern-based logical status inference for mobile phones," Master's thesis, University of Arkansas, 2016.
- [5] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *International Journal of Ad Hoc and Ubiquitous Computing*, 2007.
- [6] C. Bettini, O. Brdiczka, K. Henriksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, "A survey of context modelling and reasoning techniques," *Pervasive and Mobile Computing*, 2010.
- [7] W. Z. Khan, Y. Xiang, M. Y. Aalsalem, and Q. Arshad, "Mobile Phone Sensing Systems: A Survey," *Commun. Surveys Tuts.*, 2013.
- [8] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell, "A survey of mobile phone sensing," *Communications Magazine, IEEE*, 2010.
- [9] H. Lu, J. Yang, Z. Liu, N. D. Lane, T. Choudhury, and A. T. Campbell, "The Jigsaw continuous sensing engine for mobile phone applications," in *Proc. of SenSys*, 2010.
- [10] Z. Yan, V. Subbaraju, D. Chakraborty, A. Misra, and K. Aberer, "Energy-efficient continuous activity recognition on mobile phones: An activity-adaptive approach," in *Proc. of ISWC*, 2012.
- [11] R. LiKamWa, Y. Liu, N. D. Lane, and L. Zhong, "MoodScope: building a mood sensor from smartphone usage patterns," in *Proc. of MobiSys*, 2013.
- [12] K. K. Rachuri, M. Musolesi, C. Mascolo, P. J. Rentfrow, C. Longworth, and A. Aucinas, "EmotionSense: a mobile phones based adaptive platform for experimental social psychology research," in *Proc. of UbiComp*, 2010.
- [13] Microsoft HoloLens, "<https://www.microsoft.com/microsoft-hololens/en-us>," microsoft HoloLens.

- [14] Google Glass, “<https://www.google.com/glass/start>,” google Glass.
- [15] Sony SmartEyeglass, “<https://developer.sony.com/devices/mobile-accessories/smarteyeglass/>,” augmented reality eyewear with a wired controller.
- [16] Laforge Optical, “<https://developer.sony.com/devices/mobile-accessories/smarteyeglass/>,” augmented reality Digital eyewear.
- [17] B. McNaughton, L. Chen, and E. Markus, “”dead reckoning”, landmark learning, and the sense of direction: a neurophysiological and computational hypothesis,” *Cognitive Neuroscience, Journal of*, vol. 3, no. 2, pp. 190–202, 1991.
- [18] M. S. Grewal, L. R. Weill, and A. P. Andrews, *Global positioning systems, inertial navigation, and integration*. John Wiley & Sons, 2007.
- [19] P. Bahl and V. N. Padmanabhan, “Radar: An in-building rf-based user location and tracking system,” in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2. Ieee, 2000, pp. 775–784.
- [20] J. C. Chen, K. Yao, and R. E. Hudson, “Source localization and beamforming,” *Signal Processing Magazine*, vol. 19, pp. 30–39, 2002.
- [21] Z. Sahinoglu, S. Gezici, and I. Guvenc, “Ultra-wideband positioning systems,” *NewYork: Cambridge University, Inc*, 2008.
- [22] D. Lymberopoulos, J. Liu, X. Yang, R. R. Choudhury, S. Sen, and V. Handziski, “Microsoft indoor localization competition: Experiences and lessons learned,” *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 18, no. 4, pp. 24–31, 2015.
- [23] R. Mautz and S. Tilch, “Survey of optical indoor positioning systems,” in *Indoor Positioning and Indoor Navigation (IPIN), 2011 International Conference on*. IEEE, 2011, pp. 1–7.
- [24] D. Murray and J. J. Little, “Using real-time stereo vision for mobile robot navigation,” *Autonomous Robots*, vol. 8, no. 2, pp. 161–171, 2000.
- [25] K. Pulli, A. Baksheev, K. Korniyakov, and V. Eruhimov, “Real-time computer vision with opencv,” *Communications of the ACM*, vol. 55, no. 6, pp. 61–69, 2012.
- [26] H. Du, P. Henry, X. Ren, M. Cheng, D. B. Goldman, S. M. Seitz, and D. Fox, “Interactive 3d modeling of indoor environments with a consumer depth camera,” in *Proceedings of the 13th international conference on Ubiquitous computing*. ACM, 2011, pp. 75–84.
- [27] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, “Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments,” *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 647–663, 2012.

- [28] AICON 3D Systems , “<http://www.aicon.de>,” aICON 3D Systems 2015.
- [29] Z. Zhang, “Microsoft kinect sensor and its effect,” *MultiMedia, IEEE*, vol. 19, no. 2, pp. 4–10, 2012.
- [30] Google, “<https://www.google.com/atap/project-tango>,” project Tango.
- [31] E. A. Ramirez, “An experimental study of mobile device localization,” Ph.D. dissertation, Massachusetts Institute of Technology, 2015.
- [32] S. R. Jimenez, B. Kline, D. A. Tsang, and S. J. Henderson, “Raven eye: A mobile computing solution for site exploitation,” *Industrial and Systems Engineering Review*, vol. 3, no. 2, pp. 117–123, 2015.
- [33] M. S. Nixon and A. S. Aguado, *Feature extraction & image processing for computer vision*. Academic Press, 2012.
- [34] P. V. Hough, “Machine analysis of bubble chamber pictures,” in *Conf. Proc.*, vol. 590914, 1959, pp. 554–558.
- [35] D. H. Ballard, “Generalizing the hough transform to detect arbitrary shapes,” in *Readings in computer vision*. Elsevier, 1987, pp. 714–725.
- [36] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, “Regularization of neural networks using dropconnect,” in *International Conference on Machine Learning*, 2013, pp. 1058–1066.
- [37] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [38] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [39] F. Luan, S. Paris, E. Shechtman, and K. Bala, “Deep photo style transfer,” *CoRR*, *abs/1703.07511*, 2017.
- [40] L. A. Gatys, A. S. Ecker, M. Bethge, A. Hertzmann, and E. Shechtman, “Controlling perceptual factors in neural style transfer,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [41] S. Iizuka, E. Simo-Serra, and H. Ishikawa, “Globally and locally consistent image completion,” *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 107, 2017.
- [42] C. Yang, X. Lu, Z. Lin, E. Shechtman, O. Wang, and H. Li, “High-resolution image inpainting using multi-scale neural patch synthesis,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, no. 2, 2017, p. 3.
- [43] D. J. Rezende, S. Mohamed, and D. Wierstra, “Stochastic backpropagation and approximate inference in deep generative models,” *arXiv preprint arXiv:1401.4082*, 2014.

- [44] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” *arXiv preprint arXiv:1406.2661*, 2014.
- [45] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein generative adversarial networks,” in *International Conference on Machine Learning*, 2017, pp. 214–223.
- [46] R. N. Bracewell and R. N. Bracewell, *The Fourier transform and its applications*. McGraw-Hill New York, 1986, vol. 31999.
- [47] N. Ahmed, T. Natarajan, and K. R. Rao, “Discrete cosine transform,” *IEEE transactions on Computers*, vol. 100, no. 1, pp. 90–93, 1974.
- [48] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.
- [49] A. Campbell and T. Choudhury, “From Smart to Cognitive Phones,” *Pervasive Computing, IEEE*, 2012.
- [50] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck, “Profiling Resource Usage for Mobile Applications: A Cross-layer Approach,” in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*. New York, NY, USA: ACM, 2011, pp. 321–334. [Online]. Available: <http://doi.acm.org/10.1145/1999995.2000026>
- [51] J. R. Kwapisz, G. M. Weiss, and S. A. Moore, “Activity Recognition Using Cell Phone Accelerometers,” *SIGKDD Explor. Newsl.*, 2011.
- [52] J. Lester, T. Choudhury, and G. Borriello, “A practical approach to recognizing physical activities,” in *Pervasive Computing*, 2006.
- [53] J. Yang, “Toward physical activity diary: motion recognition using simple acceleration features with mobile phones,” in *Proc. of the 1st international workshop on Interactive multimedia for consumer electronics*, 2009.
- [54] H. Lu, A. J. B. Brush, B. Priyantha, A. K. Karlson, and J. Liu, “SpeakerSense: energy efficient unobtrusive speaker identification on mobile phones,” in *Proc. of the 9th international conference on Pervasive computing*, 2011.
- [55] H. Lu, W. Pan, N. D. Lane, T. Choudhury, and A. T. Campbell, “SoundSense: scalable sound sensing for people-centric applications on mobile phones,” in *Proc. of MobiSys*, 2009.
- [56] Y. Xu, M. Lin, H. Lu, G. Cardone, N. Lane, Z. Chen, A. Campbell, and T. Choudhury, “Preference, context and communities: a multi-faceted approach to predicting smartphone app usage patterns,” in *Proc. of ISWC*, 2013.
- [57] K. Chintalapudi, A. Padmanabha Iyer, and V. N. Padmanabhan, “Indoor Localization Without the Pain,” in *Proc. of the Sixteenth Annual International Conference on Mobile Computing and Networking*, 2010.

- [58] H. Liu, Y. Gan, J. Yang, S. Sidhom, Y. Wang, Y. Chen, and F. Ye, "Push the limit of WiFi based localization for smartphones," in *Proc. of the 18th annual international conference on Mobile computing and networking*, 2012.
- [59] A. Thiagarajan, L. Ravindranath, H. Balakrishnan, S. Madden, and L. Girod, "Accurate, low-energy trajectory mapping for mobile devices," in *Proc. of the 8th USENIX conference on Networked systems design and implementation*, 2011.
- [60] J. Yang, A. Varshavsky, H. Liu, Y. Chen, and M. Gruteser, "Accuracy Characterization of Cell Tower Localization," in *Proc. of the 12th ACM International Conference on Ubiquitous Computing*, 2010.
- [61] S. Patel, H. Park, P. Bonato, L. Chan, and M. Rodgers, "A review of wearable sensors and systems with application in rehabilitation," *Journal of NeuroEngineering and Rehabilitation*, 2012.
- [62] T. Starner, S. Mann, B. Rhodes, J. Levine, J. Healey, D. Kirsch, R. W. Picard, and A. Pentland, "Augmented reality through wearable computing," *Presence: Teleoperators and Virtual Environments*, 1997.
- [63] T. Yan, D. Ganesan, and R. Manmatha, "Distributed image search in camera sensor networks," in *Proc. of the 6th ACM conference on Embedded network sensor systems*, 2008.
- [64] S. Nath, "ACE: exploiting correlation for energy-efficient and continuous context sensing," in *Proc. of MobiSys*, 2012.
- [65] S. Nirjon, R. Dickerson, J. Stankovic, G. Shen, and X. Jiang, "sMFCC: exploiting sparseness in speech for fast acoustic feature extraction on mobile devices – a feasibility study," in *Proc. of the 14th Workshop on Mobile Computing Systems and Applications*, 2013.
- [66] D. T. Nguyen, G. Zhou, X. Qi, G. Peng, J. Zhao, T. Nguyen, and D. Le, "Storage-aware smartphone energy savings," in *Proc. of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, 2013.
- [67] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, "Fine-grained Power Modeling for Smartphones Using System Call Tracing," in *Proceedings of the Sixth Conference on Computer Systems*. New York, NY, USA: ACM, 2011, pp. 153–168. [Online]. Available: <http://doi.acm.org/10.1145/1966445.1966460>
- [68] J. C. Hammer and T. Yan, "Exploiting Usage Statistics for Energy-efficient Logical Status Inference on Mobile Phones," in *Proceedings of the 2014 International Symposium on Wearable Computers*. Seattle, WA, USA: ACM, 2014.
- [69] E. Ertin, N. Stohs, S. Kumar, A. Raij, M. al'Absi, and S. Shah, "AutoSense: unobtrusively wearable sensor suite for inferring the onset, causality, and consequences of stress in the field," in *Proc. of the 9th ACM Conference on Embedded Networked Sensor Systems*, 2011.

- [70] H. Lu, D. Frauendorfer, M. Rabbi, M. S. Mast, G. T. Chittaranjan, A. T. Campbell, D. Gatica-Perez, and T. Choudhury, “StressSense: detecting stress in unconstrained acoustic environments using smartphones,” in *Proc. of the 2012 ACM Conference on Ubiquitous Computing*, 2012.
- [71] K. Plarre, A. Raij, S. M. Hossain, A. A. Ali, M. Nakajima, M. Al’absi, E. Ertin, T. Kamarck, S. Kumar, M. Scott, D. Siewiorek, A. Smailagic, and L. E. Wittmers, “Continuous inference of psychological stress from sensory measurements collected in the natural environment,” in *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, 2011.
- [72] A. Parate, M. Bohmer, D. Chu, D. Ganesan, and B. M. Marlin, “Practical prediction and prefetch for faster access to applications on mobile phones,” in *Proc. of UbiComp*, 2013.
- [73] C. Shin, J.-H. Hong, and A. K. Dey, “Understanding and prediction of mobile application usage for smart phones,” in *Proc. of the 2012 ACM Conference on Ubiquitous Computing*, 2012.
- [74] T. Yan, D. Chu, D. Ganesan, A. Kansal, and J. Liu, “Fast app launching for mobile devices using predictive user context,” in *Proc. of MobiSys*, 2012.
- [75] J. Huang, F. Qian, Z. M. Mao, S. Sen, and O. Spatscheck, “Screen-off Traffic Characterization and Optimization in 3G/4G Networks,” in *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*. New York, NY, USA: ACM, 2012, pp. 357–364. [Online]. Available: <http://doi.acm.org/10.1145/2398776.2398813>
- [76] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, “AppScope: application energy metering framework for android smartphones using kernel activity monitoring,” in *Proc. of the 2012 USENIX conference on Annual Technical Conference*, 2012.
- [77] N. Aharony, W. Pan, C. Ip, I. Khayal, and A. Pentland, “Social fMRI: Investigating and shaping social mechanisms in the real world,” *Pervasive Mob. Comput.*, 2011.
- [78] J. C. Hammer and T. Yan, “Poster: A Virtual Sensing Framework for Mobile Phones,” in *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*. New York, NY, USA: ACM, 2014, pp. 371–371. [Online]. Available: <http://doi.acm.org/10.1145/2594368.2601457>
- [79] Q. Xu, J. Erman, A. Gerber, Z. Mao, J. Pang, and S. Venkataraman, “Identifying Diverse Usage Behaviors of Smartphone Apps,” in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*. New York, NY, USA: ACM, 2011, pp. 329–344. [Online]. Available: <http://doi.acm.org/10.1145/2068816.2068847>
- [80] A. Kansal, S. Saponas, A. J. B. Brush, K. S. McKinley, T. Mytkowicz, and R. Ziola, “The Latency, Accuracy, and Battery (LAB) Abstraction: Programmer Productivity and Energy Efficiency for Continuous Mobile Context Sensing,” in *Proceedings of*

the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages and Applications. New York, NY, USA: ACM, 2013, pp. 661–676. [Online]. Available: <http://doi.acm.org/10.1145/2509136.2509541>

- [81] G. H. John, R. Kohavi, K. Pfleger, and others, “Irrelevant Features and the Subset Selection Problem.” in *ICML*, 1994, pp. 121–129.
- [82] H. Markram, “The human brain project,” *Scientific American*, vol. 306, no. 6, pp. 50–55, 2012.
- [83] R. Caruana and V. R. de Sa, “Benefitting from the Variables That Variable Selection Discards,” *J. Mach. Learn. Res.*, vol. 3, pp. 1245–1264, 2003. [Online]. Available: <http://dl.acm.org/citation.cfm?id=944919.944972>
- [84] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, “Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments,” in *In ISER*, 2010.
- [85] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, “Kinectfusion: Real-time dense surface mapping and tracking,” in *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*. IEEE, 2011, pp. 127–136.
- [86] P. Tanskanen, K. Kolev, L. Meier, F. Camposeco, O. Saurer, and M. Pollefeys, “Live metric 3d reconstruction on mobile phones,” in *Computer Vision (ICCV), 2013 IEEE International Conference on*. IEEE, 2013, pp. 65–72.
- [87] K. Khoshelham and S. O. Elberink, “Accuracy and resolution of kinect depth data for indoor mapping applications,” *Sensors*, vol. 12, pp. 1437–1454, 2012.
- [88] M. Zollhöfer, M. Nießner, S. Izadi, C. Rehmann, C. Zach, M. Fisher, C. Wu, A. Fitzgibbon, C. Loop, C. Theobalt *et al.*, “Real-time non-rigid reconstruction using an rgb-d camera,” *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 156, 2014.
- [89] Google Project Tango Documentation, “<https://developers.google.com/project-tango/overview/concepts>.”
- [90] D. Ha, A. Dai, and Q. V. Le, “Hypernetworks,” *arXiv preprint arXiv:1609.09106*, 2016.
- [91] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu, “Pixel recurrent neural networks,” *arXiv preprint arXiv:1601.06759*, 2016.
- [92] S. C. Ashmore and M. S. Gashler, “Practical techniques for using neural networks to estimate state from images,” in *Machine Learning and Applications (ICMLA), 2016 15th IEEE International Conference on*. IEEE, 2016, pp. 916–919.
- [93] K. O. Stanley, D. B. D’Ambrosio, and J. Gauci, “A hypercube-based encoding for evolving large-scale neural networks,” *Artificial life*, vol. 15, no. 2, pp. 185–212, 2009.

- [94] C. Dong, C. C. Loy, K. He, and X. Tang, “Image super-resolution using deep convolutional networks,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 2, pp. 295–307, 2016.
- [95] J. Kim, J. Kwon Lee, and K. Mu Lee, “Accurate image super-resolution using very deep convolutional networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1646–1654.
- [96] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang *et al.*, “Photo-realistic single image super-resolution using a generative adversarial network,” *arXiv preprint*, 2016.
- [97] G. Toderici, D. Vincent, N. Johnston, S. J. Hwang, D. Minnen, J. Shor, and M. Covell, “Full resolution image compression with recurrent neural networks,” in *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE, 2017, pp. 5435–5443.
- [98] W. K. Hastings, “Monte carlo sampling methods using markov chains and their applications,” *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970.
- [99] T. Tieleman and G. Hinton, “Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude,” COURSERA: Neural Networks for Machine Learning, 2012.
- [100] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

A Appendix

A.1 List of Published Papers

1. Jon C. Hammer, Michael S. Gashler, “Learning Resolution-independent Image Representations”, Pending Review for ICCI*CC, 2018.
2. Jon C. Hammer, “Enabling Usage Pattern-based Logical Status Inference for Mobile Phones”, *University of Arkansas*, 2016.
3. Jon C. Hammer, Tingxin Yan, “Inferring Mobile User Status with Usage Cues”, *Computer Magazine*, 48 (6), 34-40, 2015.
4. Jon C. Hammer, Tingxin Yan, “Exploiting Usage Statistics for Energy-efficient Logical Status Inference on Mobile Phones”, *In Proceedings of ACM ISWC 2014. Seattle, WA*, 2014.
5. Jon C. Hammer, Tingxin Yan, “A Virtual Sensing Framework for Mobile Phones”, *In Proceedings of ACM MobiSys (Poster)*, 2014.

A.2 IRB Approval



Office of Research Compliance
Institutional Review Board

September 10, 2014

MEMORANDUM

TO: Tingxin Yan
Jon Hammer

FROM: Ro Windwalker
IRB Coordinator

RE: PROJECT CONTINUATION

IRB Protocol #: 13-09-068

Protocol Title: *Context-Aware Mobile Computing and Participatory Sensing*

Review Type: EXEMPT EXPEDITED FULL IRB

Previous Approval Period: Start Date: 10/01/2013 Expiration Date: 09/30/2014

New Expiration Date: 09/30/2015

Your request to extend the referenced protocol has been approved by the IRB. If at the end of this period you wish to continue the project, you must submit a request using the form *Continuing Review for IRB Approved Projects*, prior to the expiration date. Failure to obtain approval for a continuation on or prior to this new expiration date will result in termination of the protocol and you will be required to submit a new protocol to the IRB before continuing the project. Data collected past the protocol expiration date may need to be eliminated from the dataset should you wish to publish. Only data collected under a currently approved protocol can be certified by the IRB for any purpose.

This protocol has been approved for 60 total participants. If you wish to make *any* modifications in the approved protocol, including enrolling more than this number, you must seek approval *prior* to implementing those changes. All modifications should be requested in writing (email is acceptable) and must provide sufficient detail to assess the impact of the change.

If you have questions or need any assistance from the IRB, please contact me at 210 Administration Building, 5-2208, or irb@uark.edu.

210 Administration Building • 1 University of Arkansas • Fayetteville, AR 72701
Voice (479) 575-2208 • Fax (479) 575-3846 • Email irb@uark.edu

The University of Arkansas is an equal opportunity/affirmative action institution.