In-Material Processing of High Bandwidth Sensor Measurements using Modular Neural Networks

by

Dana Hughes

B.S., Colorado State University, 2000

M.S., University of Missouri—Rolla, 2003

M.S., College of Charleston, 2012

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

2018

This thesis entitled:
In-Material Processing of High Bandwidth Sensor Measurements using Modular Neural Networks
written by Dana Hughes
has been approved for the Department of Computer Science

Dr. Nikolaus Correll

Dr. Christoffer Heckman

Dr. Sriram Sankaranarayanan

Dr. Richard Han

Dr. Kurt Maute

Date

The final copy of this thesis has been examined by the signatories, and we find that both the content and the form meet acceptable presentation standards of scholarly work in the above mentioned discipline.

Hughes, Dana (Ph.D., Computer Science)

In-Material Processing of High Bandwidth Sensor Measurements using Modular Neural Networks

Thesis directed by Dr. Nikolaus Correll

Robotic materials are a novel class of materials that tightly integrate sensing, computing, and actuation into an engineered material or composite to allow the behavior of the material to be defined algorithmically. Robotic materials are constructed using an embedded network of computing nodes based on small, inexpensive microcontrollers. Examples of such materials include morphable airfoils which change shape in response to flight conditions or mission parameters, robotic skins with rich tactile sensing capabilities that recognize texture or touch gestures, clothing with tightly integrated sensing to assist with or augment the wearer's perception of the environment, or materials with dynamic camouflage capabilities.

In this thesis, I develop a framework for in-material processing which tightly couples modularized deep neural networks and high-bandwidth sensors using a network of embedded, material-scale components. This framework enables materials to learn multiple desired responses to stimuli, avoiding the need for accurate modeling of the dynamics of the material and stimuli.

I utilize a modular neural network design consisting of convolutional (CNN) and long short-term memory (LSTM) layers implemented in each node in the material as a computational approach for robotic materials. This network architecture allows for nodes in the material to process local sensor values, maintain local state information, and communicate with nodes in a local neighborhood in the materials. A multiobjective optimization approach is employed to automatically design the neural network architectures which maximizes the performance of the network while ensuring hardware budgets, such as memory requirements, are maintained. A communication network design is also developed to allow network modules to learn a communication protocol that limits communication to a desired rate, ensuring in-network bandwidth constraints are maintained.

I demonstrate the suitability of this computational model for robotic materials using ex-

amples in several domains. An RF-based e-textile gesture input device capable of distinguishing between user control gestures is used to control arbitrary external devices. A tire with embedded piezoelectric sensing capabilites for use in high-performance autonomous vehicles performs state-of-the-art identification of terrains driven on. Two robotic skins are presented—one which is capable of detecting and localizing contact, and identifying the texture of the contacting objects; and a second which assists with avoiding collisions with obstacles and identifies affective touch gestures performed by a human collaborator. Finally, a distributed approach to human activity recognition is presented whose activity identification performance is comparable to a centralized approach, but can be implemented on hardware designed for wearable applications, as opposed to a GPU-enabled device. The examples shown demonstrate that robotic materials can perform significant in-material processing; are loosely coupled from a host system, communicating a minimal number of low-bandwidth events to the host; and can exhibit multifunctional behavior that is analyzed for safety or performance considerations.

Dedication

This thesis is dedicated to members of my family whose support has been critical to its completion.

First, to my mother, Nancy Wilkinson, who impressed upon me from a young age the importance of my education, and to my father, Einar Hughes, who was always eager to hear about my current research.

Second, to my wife Missy Hughes, who was willing to support me during our time in Colorado, and who probably is the only person who is more excited than I am that this thesis is completed.

Finally, to my son Oliver Hughes. His appearance into this world provided the final push to complete this thesis, hope that this work provides some small improvement to his world, and inspiration for our future endeavors.

Acknowledgements

I would like to extend heartfelt thanks to all my mentors who helped me and guided my research during my time here. To my advisor, Dr. Nikolaus Correll, who has provided me with valuable guidance for both my research and future career goals. You helped me integrate my research interests into a cohesive thesis and helped me recognize the importance and scope of my work. As I completed this thesis, you pushed me to apply for and work towards postdoctoral and faculty positions at the institutions of the highest caliber—your confidence in my abilities has been inspiring and makes me realize the extent of my capabilities. To Dr. Christoffer Heckman, whose interest in and support of my research has resulted in a deep understanding of the complexities of autonomous vehicles, and direct application of my research in this field. To Dr. Sriram Sankaranarayanan, for his discussions regarding how to most effectively present my work in the future. Finally, to Drs. Richard Han and Kurt Maute, who have provided guidance and interesting paths to take my research in the future.

I would also like to thank my colleagues in Dr. Correll's lab for providing a convivial work environment. I would like to specifically thank Nicholas Farrow for his help with hardware development, and Dr. Halley Profita, for introducing me to the fascinating field of e-Textiles and wearable computing. I would like to thank my several coauthors—Alon Krauthammer, Sarah Radzihovsky, John Lammie and Sarah Aguasvivas Manzano—who helped write such high-quality papers.

Contents

Chapter

Tables

Table

Figures

Figure

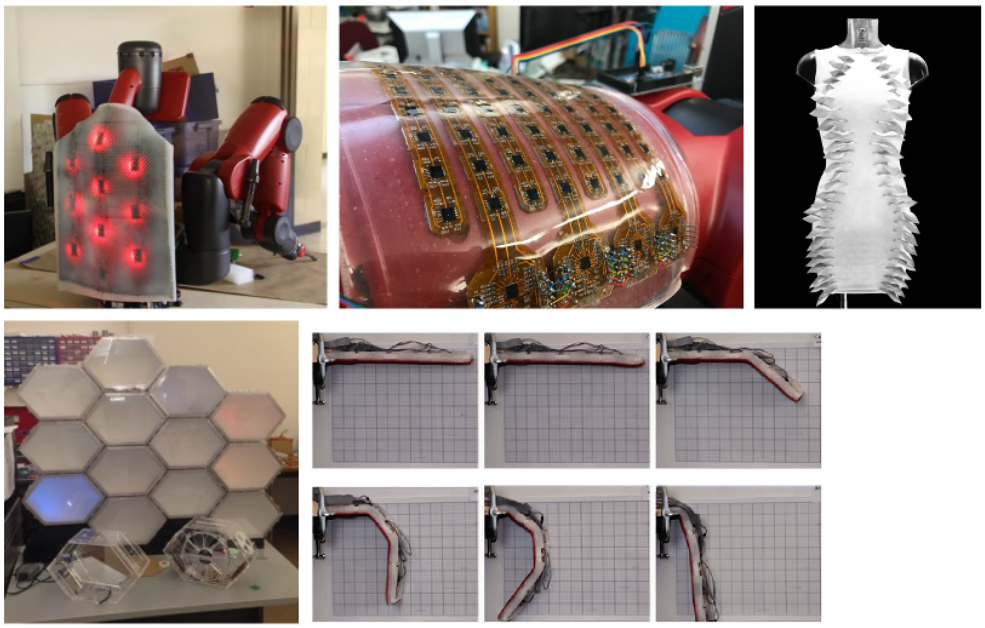
Chapter 1

Introduction

Robotic materials is a new class of materials that tightly integrates sensing, actuation, communication and computation elements into engineered composites or materials [91]. Robotic materials have the potential of creating multi-functional materials whose behaviors can be defined algorithmically, rather than simply demonstrating a stimuli-response behavior. These materials are inspired by the functionality of natural systems, such as the camouflage skin of cuttlefish, bird wings which morph to change flight properties, or multiple sensing modalities present in human skin. Several examples of such materials have been demonstrated, as shown in Figure 1.1: robotic skins that perform contact localization and texture identification [53, 54] and gesture recognition and collision avoidance [60]; an assistive garment localizes sound and directs the wearer's attention toward the source [108]; an interactive architectural façade [16]; and variable stiffness beam capable of changing shape [89, 90].

Several challenges associated with the development of robotic materials have been identified in recent literature [15, 17, 92]. Processing of high-bandwidth sensor signals is cited as a common challenge, as well as a motivation for robotic materials: the information collected by a large number of sensors, multiple sensing modalities, and/or sensors with high sampling rates rapidly exceeds communication bandwidth when sensor signals are processed by a central system. Regardless of application, a successful robotic material requires extracting low-bandwidth patterns from local sensor data, whether to aggregate and communicate directly to a computing sink, or as a part of a distributed algorithm for controlling local acutators. In many cases, machine learning will need to

Figure 1.1: Example Robotic Materials: robotic skin capable of detecting and localizing contact and identifying textures (top left, [54]); robotic skin capable for obstacle avoidance and gesture recognition (top center [60]); assistive garment for sound localization (top right [108]); amorphous architectural façade (bottom left [16]); morphable beam with variable stiffness control (bottom right [89]).



be leveraged to extract desired information, such as determining gesture from spatiotemporal data in a pressure-sensitive robotic skin [15, 92].

A second challenge to robotic materials is the need to develop distributed computing and control algorithms to achieve the desired behavior in the material. Inspiration has been drawn from other domains, such as swarm robotics, amorphous computing, and sensor networks. One source of inspiration, which is of particular interest for this thesis, is organization and emergent computing capabilities of neurons in biological systems. From a computating perspective, this biological motivation has resulted in the development of artificial neural networks, which have been shown to be effective in a variety of tasks. The advancement of this field from shallow to deep neural networks over the last decade has resulted in significant progress in several domains, particularly in machine vision, speech recognition, and natural language processing [82].

Several examples exist in biology where neural organization is exploited to perform significant processing or control locally. For instance, in vertibrates, stimulation of densely packed optical nerves on the retina is processed by rapidly generating neural codes in the retina [40], and various receptive fields in the visual cortex respond to spatial patterns projected on the eye [52]; this neural architecture is the basis of convolutional neural networks used image processing [36]. The neural organization in octopi has evolved in such a way that each arm and optic lobe of the octopus contains the same number of neurons as the central brain; this morphology allows for local sensing and control in each arm, solving the challenges associated with controlling an appendage with infinite degrees of freedom [46]. A final example is demonstrated by the tactile capabilities of the human hand—peripheral neurons on the human fingertip detect the orientation of edges [109], and generate contact events for the brain which define subgoals of manipulation tasks and triggers corrective actions when mismatches occur [65].

Deep learning methods have resulted in computational models composed of multiple processing layers, each layer capable of learning more abstract representations of the provided input [82]. Specifically, advances over the last few years have resulted in architectures capable of processing complex spatial and temporal information [81], able to model and respond to sequential informa-

tion [85], as well as learning control policies through reinforcement learning [84]. Given the success of neural organization in biology, as well as the success of deep neural networks for a variety of perception and control tasks, it is natural to consider deep learning architectures as a computational basis for robotic materials.

1.1 Thesis Overview

In this thesis, I propose that modularlized deep neural networks consisting of convolutional and recurrent layers can be effectively integrated into robotic materials as an approach for performing in-material processing of high-bandwidth sensor measurements. This approach allows a material to extract high-level information from spatiotemporal stimuli using a distributed set of sensing and computing nodes, with minimal degredation of results when compared to a centralized approach. In addition, this approach allows for use of a network of small, inexpensive microcontrollers, as opposed to expensive computing elements with high-bandwidth communication buses. Ultimately, I develop a modular CNN-LSTM architecture as a model for performing computation in robotic materials, and demonstrate the utility of this model.

1.2 Main Contributions

This work demonstrates that deep learning approaches are applicable for processing high-bandwidth sensor measurements in robotic materials, and are suitable for perception tasks for a variety of robotic materials. Specifically, modular CNN and CNN-LSTM networks are presented which allows training and implementing a large, distributed network with identical building blocks in a scalable way, enabling large-scale robotic materials.

The neural network architectures implemented on each node in a robotic material is constrained by limits on computing, memory, and communication bandwidth by the selected microcontrollers. This thesis provides two approaches to mitigate these issues. First, a multi-objective optimization algorithm is used to evolve local convolutional neural network (CNN) architectures which jointly minimize or maximize desired properties of the architecture. The typical objectives

involve minimizing the number of network parameters (and thus memory requirements) while maximizing the performance (e.g., classification accuracy). Second, an approach to learning sparse communication protocols between connected nodes is provided, which mitigates the possibility of exceeding communication bandwidth in a robotic material, and minimizes energy consumption due to communication.

From an applications perspective, this thesis demonstrates the ability to perform in-material perception for several tasks, providing components for or advancing perception in several domains. Robotic skins capable of affective touch recognition and collision avoidance have been developed for human-robot interaction tasks. A novel e-textile input device and distributed activity recognition is demonstrated for wearable computing. Finally, a novel terrain sensitive tire has been developed for autonomous vehicles.

1.3 Thesis Outline

This thesis is the synthesis of my published work associated with robotic materials, and is broadly organized into three main categories: background and motivation; CNN models, memory considerations, and examples of robotic materials with a single node; and modular CNN-LSTM models, communication considerations, and examples of robotic materials with multiple nodes. Chapter 2 provides a background on both robotic materials and deep learning [56].

Chapters 3 and 4 considers the case where single sensing and computing nodes are used to enable robotic materials. Chapter 3 introduces and provides motivation for using convolutional neural network to perform signal processing in the single-node case. Motivation and desired properties are discussed, and an algorithm for evolving CNN architecture that can be implemented on material-scale components (i.e., small microcontrollers and MEMS sensors and actuators) is provided. Chapter 4 discusses three experiments where in-material processing of high-bandwidth sensing is performed using a single computing node. Finally, this chapter demonstrates how the features extracted from the CNN architectures can be exploited to perform ancillary functions in the material, demonstrates a robotic material capable of changing behavior based on context, as

well as analyzing the *global* behavior of the material-host system.

Chapters 5, 6, and 7 considers the case where multiple computing nodes are used in a robotic material. Chapter 5 describes a multifunctional robotic skin which utilizes knowledge of the dynamics of the skin and an amorphous algorithm to perform various tasks—the purpose of this chapter is to determine the desired properties of a scalable, multi-node processing model, and to contrast using an amorphous computing model to a modular CNN-LSTM model. Chapter 6 presents the modular CNN-LSTM model, a scalable computing model for use in robotic materials. A network architecture suitable for learning a communication protocol with limits communication rate is described—as with the algorithm described in Chapter 3, the communication network aims to limit communication in the network, ensuring bandwidth limits are not exceeded and minimizing energy consumption associated with communication. Chapter 7 demonstrates the utility of the modular CNN-LSTM architecture for multinode robotic material applications, using distributed human activity recognition as an example [55].

A summary of contributions of this thesis and potential future work is discussed in Chapter 8.

Chapter 2

Background

2.1 Robotic Materials

Robotic materials stems from the growing interest over the last few decades to extend the functionality of various engineered materials, such as composites, cement-based materials, polymers, and textiles, beyond that of a purely structural material. The ability of creating such materials has been enabled by several advances in material science, miniaturization of electronics and microcontrollers, and manufacturing capabilities. From a material science perspective, a class of materials (referred to in the literature as *functional materials* or *smart materials*) has emerged to describe materials which respond to some external stimuli, such as piezoelectric materials, shape memory alloys and polymers, and chromic materials. From a manufacturing perspective, microelectromechanical systems (MEMS) have been developed, resulting in a suite of sensors and actuators, including accelerometers, gyroscopes, pressure sensors, LEDs, ultrasonic transducers, and microphones, which are on the millimeter scale. Microcontrollers and microprocessors have also become available on a similar scale; combined with the sensing and actuation capabilities of MEMS sensors and functional materials, it is possible to design small nodes capable of performing sensing, actuation and computation in engineered materials.

The concept of distributed MEMS [7] evolved from fabrications techniques which enable miniaturization and batch fabrication of sensors and actuators. As the physical size and cost of sensors and actuators decrease, systems with thousands or millions of units became feasible. Photolithographic processes provide a means to manufacture mechanical components in MEMS

in conjunction with microelectronic components, which provide a means to control the resulting sensor and actuator arrays. "Smart dust" is a conceptual application for such manufacturing capabilities [136], where cubic-millimeter modes form the basis of a massively distributed sensor network.

Programmable matter is a concept based on a similar concept of millimeter-scale units [39]. Small claytronic atoms (catoms) are units which are capable of moving in three dimensions, adhering to other catoms, communicating with other catoms, and computing state information. Ensembles of catoms are able to model 3D scenes, creating items of arbitrary shape.

Amorphous computing is a similar concept to programmable matter, though individual elements are influenced by biological cells [2]. One main concept in amorphous computing is that there is no underlying structure to the individual cells, and communication is not viewed as discrete events. Rather, messages are diffused throughout the material, and individual cells respond to concentrations of received messages. In this way, programs are designed such that state propagates through the material similar to wave propagation in physics, and based on markers within cells and local response rules, pattern formation is possible.

2.1.1 Algorithmic Considerations

There are several necessary considerations associated with algorithms implemented in a robotic material [91]. Algorithms must scale as the material grows in size. As it is desired to keep the cost of individual nodes to a minimum, algorithms must be able to run on limited computing capabilities and memory of the selected microcontroller. Furthermore, algorithms must be robust to failure of individual nodes. These considerations have many implications with regard to the use of machine learning approaches with robotic materials.

Scalability imples that any algorithm cannot have full access to all sensor data or all actuators in the system. One approach is to ensure that nodes have limited support over the sensors, i.e., a learned task relies only on information gathered by nodes in a local neighborhood. For many applications, it is reasonable to assume that information gathered from a local neighborhood of

nodes would be sufficient for the task of interest. For example, detecting the location of the source of a vibration requires only a small number of sensing nodes [54]. The intensity of vibration due to an impact decreases as distance increases from the location of impact. Consequently, nodes outside a sufficiently large enough radius from the source would not detect the impact, and only nodes within the radius need be used to calculate the location of the impact.

2.2 Neural Networks and Deep Learning

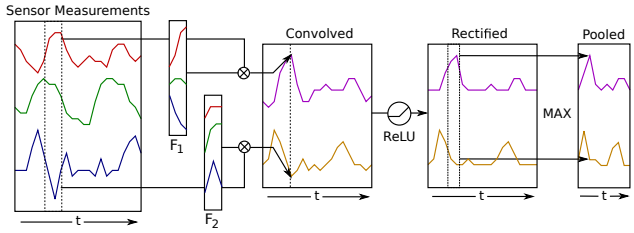
Recent trends in neural networks (i.e., so-called "deep learning" approaches) have demonstrated several desired properties for processing in robotic materials. From the perspective of signal processing and perception, convolutional neural networks have been shown to be well suited for learning suitable features for classificaiton of spatial [78, 81], temporal [81], and spatiotemporal [130, 133] data. Long-term temporal dependencies are readily mapped using recurrent networks, specifically long short-term memory [48]. Convolutional neural networks generate layers of *feature maps*, with nodes in earlier layers activating in response to simple, local features and nodes in higher layers responding to more abstract concepts (e.g., faces in image recognition) [144]. These feature maps are learned directly from training data; thus, the features learn to contribute to the discrimination between different training examples.

From a computational perspective, neural networks have the advantage of being *universal appoximators* [51, 50], that is, a feedforward network can approximate a measurable function to a desired degree of accuracy. Similarly, the computational capabilities of recurrent neural networks have been explored. Recurrent networks can be constructed to represent arbitrary finite state machines [99]. Additionally, [120] demonstrates that recurrent networks are equivalent to Turing machines, and therefore can be used to implement any computable function. While these observations do not provide information on the size of the networks needed to implement an arbitrary algorithm, they do demonstrate that neural networks can theoretically be trained to perform an arbitrary behavior in a robotic material.

2.2.0.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) are feed-forward networks which utilize a weight sharing scheme to allow a network to learn local features which are invariant to translation and scaling. This is achieved through the use of convolutional layers, which convolve a set of kernels with an input to generate a feature map, and pooling layers, which reduces the size of a layer by subsampling small regions of the input. Examples of these operations is shown in Figure 2.1 for a 1D case and Figure 2.2 for a 2D case. The primary advantage of convolutional networks is the use of weight sharing: sets of weights are replicated among multiple connections. This allows the network to detect local features in the input, as well as reducing the total number of paramters in the model. Additionally, features suitable to a particular task are automatically generated during training, avoiding the need to hand-engineer features for the task.

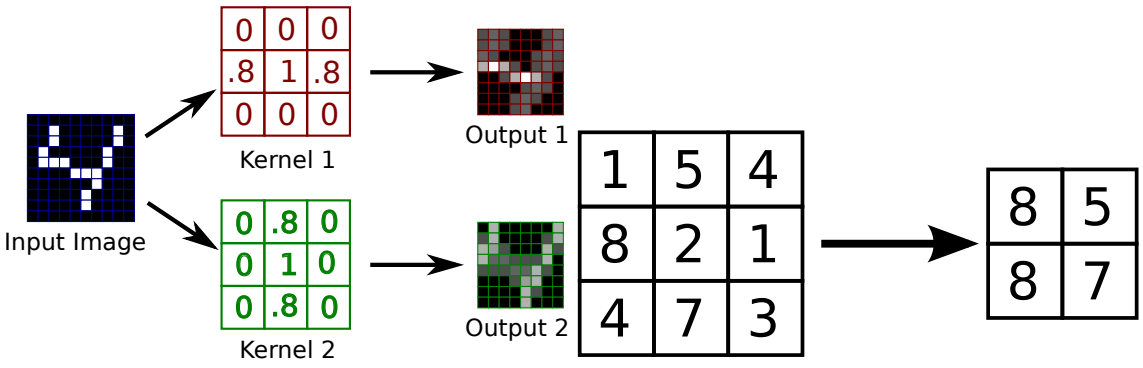
Figure 2.1: Example of convolutional and pooling operations for a 1D convolutional neural network.



2.2.0.2 Recurrent Neural Networks

Generally, feedforward neural networks are not suitable for sequential or time-series data, due to the fixed input size of the network. When such networks are used for time series data, windows are typically extracted from measured signals and assumed to be quasi-static in nature. However, these models are unable to capture effects requiring a significant delay between the input of the signal and the desired effect. Recurrent neural networks consist of networks which contain cycles, which are more suitable for sequential or time series models. Recurrent connections allow the network to mainatin and update an interal state at each step in the sequence, providing a

Figure 2.2: Example of convolutional and pooling operations for a 2D convolutional neural network.



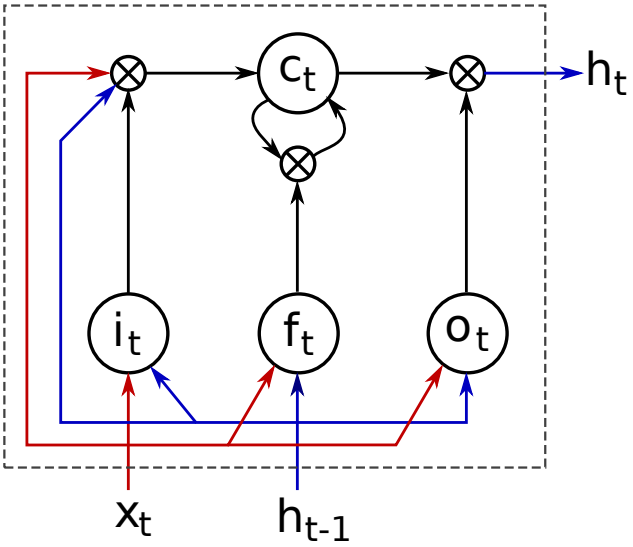
suitable architecture to handle temporal aspects of the signal.

A major limitation to recurrent neural networks is the inability to learn tasks where there exists a long time delay between the effect from some input signal and the corresponding output. The network is unable to learn with large time delays due to the nature of backpropagating the error signal through time during training [47]. Essentially, as the error is propagated backward through time, it either vanishes or blows up, depending on the recurrent weights. A promising solution to this problem is to incorporate Long Short Term Memory (LSTM) [48]. A LSTM unit, shown in Figure 2.3, can store values using a recurrent neuron and perform operations on the value of this neuron using several gate neurons. These values may then be read, written to, or forgotten, based on the activation of input, output or forget gates.

2.2.0.3 Notation

For the sake of brevity and consistency, a shorthand notation for the architecture of a network is used. Each layer is represented as a tuple, with the first element being a letter representing the layer type (input, I, convolutional, C; pooling, P; fully connected, FC; LSTM, $LSTM$; or softmax, S). For fully connected, LSTM or softmax layers, the second element of the tuple is the number of units in the layer. For an input layer, the second element represents the dimensionality of the input to the network, with the last value of the dimensionality representing the number of measurement

Figure 2.3: Long short-term memory cell



channels of the input. For convolutional layers, the second element is the dimensionality of each kernel, the third element is the number of kernels, and, if present, the fourth element is the stride. For a pooling layer, the second and third elements represent the pooling and stride dimensions, respectively.

As an example, [(I,(30,15,3)),(C,(8,4),5),(P,(2,2),(2,2)),(LSTM,20),(FC,10),(S,3)] represents a network with input with 3 channels and dimensionality of 30 x 15, followed by a convolutional layer consisting of 5 kernels with dimensions 8 x 4, followed by a 2 x 2 pooling layer with stride 2 in each dimension, followed by an LSTM layer with 20 LSTM cells, a fully connected layer with 10 neurons, and a softmax layer with 3 neurons. Unless otherwise noted, a rectified linear actuation function (i.e., ReLU, $y = max(0, x)$) is applied to the output of each convolutional, fully connected and LSTM layer, and max-pooling operations are assumed for each pooling layer. Networks were implemented and trained using TensorFlow [1].

Chapter 3

Convolutional Neural Networks for Signal Processing in Materials

In many applications, the spatial dimensions of a particular robotic material precludes the need for multiple computing nodes in the material. However, benefits to processing sensor signals in-material remain, and are specifically useful in certain applications.

Sampling a number of sensors in-material can mitigate the need for high-bandwidth communication channels to be supplied for each sensor. Relying instead on an embedded microcontroller to sample and process the signals replaces these communication channels with a single low-bandwidth channel. As an example, tires with embedded piezoelectric sensors can measure vibrations generated in the tire due to interaction with the terrain. Transferring signals from the tire to the chassis of the vehicle, however, requires either a slip ring, or a wireless communication channel with sufficient bandwidth to transmit the sensor samples. The former approach adds an additional expensive component to the vehicle, while the latter is limited by the bandwidth capabilities of the wireless transmitter, and still requires a microcontroller to sample sensors and generate packets for transmission.

An additional benefit to in-material processing is that the material can then be considered a modular component to a system as a whole, resulting in a loose coupling between the component build using the robotic material and the host system employing the component. For instance, one or more robotic skin patches capable of gesture recogntion can be applied to a variety of robots with different morphologies. Rather than tightly coupling the sensing capabilities of the skin to the host robot, a set of individual patches would simply provide the host robot with gesture information

when a contact event occured. Adding or removing skin patches, or updating or modifying the behavior of the skin patches, would be trivial in such a scenario.

The benefits of using convolutional neural networks for processing high-bandwidth sensor signals—automatically learning local features of interest, reducing model parameters, and invariance to translation and scale—provides motivation for utilizing CNNs as a primary model for processing in such materials. The primary constraint to such a model stems from the limitations of program memory on a selected microcontroller, which limits the total number of parameters available to define the network. Additional constraints may include limits on the responsiveness of the network, i.e., a maximum acceptable time to process a provided input.

This chapter provides an approach to evolving CNN architectures which jointly optimize model performance (e.g., maximizing accuracy) and memory requirements (i.e., minimizing parameter count), using a multiobjective evolutionary algorithm. The efficacy of this approach is demonstrated with two applications, handwritten digit classification and terrain classification.

3.1 Convolutional Neural Network Optimization

The large number of hyperparameters associated with CNNs (e.g., number of layers, size and number of kernels, etc.) makes hyperparameter optimization difficult. Grid search [79] is an early approach which is easily implemented, though infeasible for a large number of hyperparameters. Random Search [5] has been shown to be a more efficient approach capable of handling a larger number of hyperparameters. Bayesian optimization [123] updates a Guassian Process model after each architecture evaluation, and allows for selecting hyperparameters based on expected improvement or reducing uncertainty. Other approaches using surrogate models for the expected architecture performance include tree-structred Parzen estimators [6] and sequential model based optimization (SMBO) methods [64]. A major drawback to these approaches is the need for a fixed number of hyperparameters; for deep neural networks, this implies a fixed number of layers.

Evolutionary approaches to constructing and training neural networks has been extensively explored over the last several decades [32]. For a fixed architecture, genetic algorithms can be

easily used to learn network parameters, providing an alternative to backpropagation for training. Various approaches to evolving network architectures have been explored. Architectures can be directly encoded, such as using genotypes to represent a connection matrix between a fixed number of neurons [21], or indirectly, such as using genotypes to represent strings in a graph generating grammar [74]. NeuroEvolution of Augmenting Topologies (NEAT) [124] is of particular interest, as it is the basis for several recent approaches to evolving CNNs. Starting with a minimal architecture, new architectures can be evolved through mutation (adding nodes or connections, or disabling connections), and by performing crossover between two architectures.

While genetic algorithms have been used to evolve both the structure and parameters of shallow neural networks, evolutionary approaches to learning deep neural networks have only recently been explored. The weights of a five-layer neural network are evolved in [22]; the GA-evolved network performed better than networks evolved using backpropagation, and sparsity is encouraged by mutating weights to zero. Co-evolutionary approaches are explored in [132], creating networks with similar improvement in performance.

Discovering convolutional neural network architectures has only been recently explored. An evolutionary approach using only mutation operations is presented in [112]. Using an initial population of simple, poorly performing individuals, the approach was able to evolve architectures which performed competitively with hand-designed architectures on the CIFAR dataset. Two extensions to the NEAT algorithm, DeepNEAT and CoDeepNEAT, are presented in [93]. DeepNEAT evolves architectures by forming connections between various layers in an network, as well as evolving the attributes of each layer. CoDeepNEAT co-evolves small network modules with a blueprint for the network, encouraging repetitive structure in the resulting architecture. Similarly, the EXACT algorithm [26] generates network architectures by mutating layer connections and parameters. This approach allows for only convolutional layers, and explores distributing the evaluation of architectures on volunteer computers. Finally, the EDEN algorithm performs mutation operations on a feed-forward CNN [27]. The algorithm bounds the number of layers and layer size (number of filters, units or embeddings) to allow it to run on a single GPU in a reasonable time.

3.2 Algorithm Details

This chapter extends the idea of evolving CNN architectures by utilizing an multi-objective evolutionary algorithm to allow additional attributes (e.g., parameter count) to be included as design objectives.

3.2.1 Nondominated Genetic Algorithm-II

Genetic algorithms are an optimization technique inspired by biological evolution [139]. Given a task which requires the minimization (or maximization) of a scalar objective function, a population of potential solutions, each encoded as a **genotype**, is created. The objective of each solution is calculated. A new population is generated by randomly selecting pairs of individuals, and performing **crossover** and **mutation** operations to generate new individuals. This process is repeated until a suitable solution is found, or a defined number of generations is reached. Selection of individuals is performed such that superior individuals are selected more often than poor individuals. Combined with the crossover operation, this selection strategy exploits portions of the genotype which generate high-quality solutions. The mutation operation ensures that the algorithm doesn't converge to a local optimum.

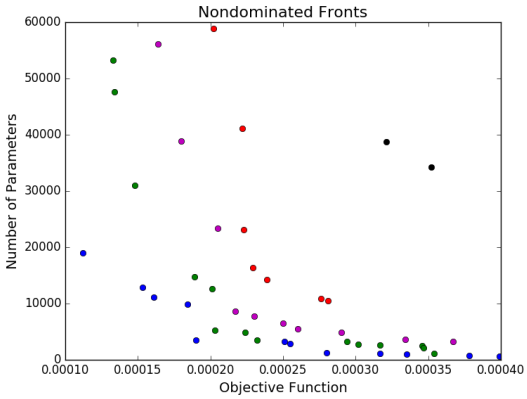
Multiobjective evolutionary algorithsm (MOEAs) extend the concept of genetic algorithms by finding solutions to problems with multiple objective functions. Rather than finding a single optimal solution, MOEAs find a set of solutions which are Pareto optimal, that is, a set of solutions where no individual is **dominated** by another. For the case where objectives are to be minimized, an individual, p, is said to dominate another individual, q, if there is no individual objective in q which is less than the same objective in p, and there exists at least one objective in p which is less than in q.

We base our approach to finding a family of Pareto-optimal network architectures on the Nondominated Sorting Genetic Algorithm II (NSGA-II) [25]. This approach has several desirable advantages over other MOEAs: it is inherently an elistist algorithm, so good solutions are main-

tained once found, and diversity is preserved in a population on a nondominated front without the need for additional parameters.

A core idea to the NSGA-II algorithm is to calculate a set of nondominated fronts at each iteration. A nondominated front consists of individual which are dominated by individuals in fronts of lower rank. An example of this is shown in Figure 3.1 for a problem with two objectives. As opposed to the conventional selection process for genetic algorithms, NSGA-II first generates a population of children and merges this with the current population. The joint population is then sort based primarily on the ranking of individuals, followed by local density of individuals within its front. The next generation is selected from better half of individuals in the joint population. This has the effect of naturally maintaining elite individuals, ensuring quality solutions are not discarded.

Figure 3.1: Nondominated fronts for neural network architectures which are jointly minimizing classification error and parameter count. The Pareto front consists of blue points; green, purple, red, and black points indicate additional nondominated fronts.



3.2.2 CNN Representation

The convolutional and pooling layers of CNNs have the effect of reducing the dimensionality of the previous layer. Given an input dimension of i and a kernel or pool width w and stride s, the

dimension of the output, o, is given as

$$o = \left\lfloor \frac{i - w}{s} \right\rfloor + 1 \tag{3.1}$$

assuming no zero padding [28]. From an implementation standpoint, we assume the width of a kernel cannot exceed the input dimension.

CNNs are directly encoded on structured, variable length genotypes. Genotypes consist of a sequence of layers, with the output of layer i being used as the input to layer $i + 1$. We allow three types of layers: convolutional, pooling and fully connected. Each layer contains layer-specific parameters, which are summarized in Table 3.1.

Table 3.1: Available genotype layers and associated parameters.

Layer Type	**Parameters**
Convolutional	kernel dimensions, kernel stride, activation function
Pooling	pooling dimensions, pooling stride
Fully Connected	number of neurons, activation function

The parameters of genotypes are constrained to ensure that the generated CNN is valid—the dimensions and stride of convolutional and pooling layers cannot take values such that the output dimensionality of a layer is less than one, as given by Equation 3.1. Furthermore, all convolutional and pooling layers must preceed any fully connected layers in the network. Finally, we do not allow two sequential pooling layers, as two such layers could be reduced to a single pooling layer.

3.2.3 Genotype Generation

We generate genotypes by iteratively adding layers to the end of the genotype, starting with an initial input layer. Convolutional layers are added with a probability $p_{conv}^{(g)}$, a pooling layer is added after with a probability $p_{pool}^{(g)}$. Convolutional and pooling layers can be added until such a layer becomes infeasible, i.e., the output dimensionality of the previous layer is one. Fully connected layers are added after convolutional and pooling layers in a similar manner, with a fully connected

layer being added with probability $p_{fc}^{(g)}$.

Parameters for each type of layer are randomly selected from a modified Poisson distribution. A parameter is assigned a value n with probability

$$P(n) = \frac{\lambda^n e^{-\lambda}}{n!} + n_{min} \tag{3.2}$$

where n_{min} is the minimum value a parameter can take, and $\lambda + n_{min}$ the expected value of n. Selected values which exceed the maximum possible value for a parameter are truncated to the maximum possible value. We allow separate values for λ and n_{min} for each parameter. We use a Poisson distribution to avoid generating genotypes with excessively large kernel or pooling dimensions or strides.

3.2.4 Crossover

Crossover is performed on two parent genotypes to generate to child genotypes. The output dimensionality and the minimum input dimensionality of each layer in each parent genotype is calculated (the minimum input dimensionality of fully connected layers is one). All valid crossover points (i, j) between the two genotypes are determined, where a crossover point is valid if the output dimensionality of layer i of the first genotype is smaller than the minimum input dimensionality of layer j of the second genotype. If a valid crossover point cannot be performed, children are generated by performing a mutation operation on each parent.

3.2.5 Mutation

Two mutation operations are defined: **addition** of a layer and **modification** of a layer. The **addition** operation involves inserting a randomly generated layer at a random position in the genotype, and **modification** involves randomizing one of the parameters of a random layer. The same constraints apply for the **addition** operation as with generation and crossover operation—layers cannot be added if it results in two sequential pooling layers, only fully connected or output layers can follow a fully connected layer, and layer parameters cannot result in an invalid architecture.

Added layers are generated in the same manner as during initial genotype generation. Layers are modified by selecting a parameter at random, and randomly adding or subtracting a random value selected using the modified Poisson distribution in Equation 3.2. A layer removal mutation is not provided, as networks are crossover operations are capable of generating architectures with fewer layers; experiments including a removal operation resulted in biasing architectures towards shallow architectures.

3.2.6 Genotype Evaluation

As network architectures vary greatly, the number of training epochs required to reach convergence is also likely to vary between architectures. An automated stopping criteria is desired, in order to ensure that training doesn't end early for complex networks that require a large number of training epochs, and excess training epochs are not used for very simple networks that converge quickly.

An automated stopping criteria is presented in [94], based on identifying steady state in a signal. The steady-state identification algorithm estimates the variance of a signal using two approaches. The first variance is calculated as the sum of the squared differences of the data from the mean, and the second is the sum of the squared differences of successive measurements. When a signal is at steady state, the ratio of these two values are near unity; a large ratio indicates that the signal is not at steady state. The algorithm is computationally inexpensive, and requires three filter parameters.

The stopping criteria is applied to the validation cost calculated after each training epoch. Training is considered complete when the variance ratio is below 1.5 for 10 training epochs.

3.3 Experimental Results

The proposed algorithm was used to generate CNN architectures for two datasets: MNIST [83], and data collected from a terrain-sensitive tire (see Chapter 4). The MNIST dataset consists of 60,000 images of hand-written digits, of which 5,000 were removed for a validation set to evaluate

individual genotypes. The tire data consists of 24,472 measurements extracted from a tire driving on four simulated terrains; 20% of this data was retained for validation.

Each experiment used the same set of parameters for the NSGA-II algorithm, which is summarized in Table 3.2. The values of λ and n_{min} were the same for both generating and mutating genotypes. $p_{crossover}$ and p_{mutate} are the rate at which crossover and mutation operations are performed, respectively. If crossover is not performed, mutation is automatically performed to ensure unique genotypes are generated.

Table 3.2: Parameters used for the NSGA-II algorithm for both experiments.

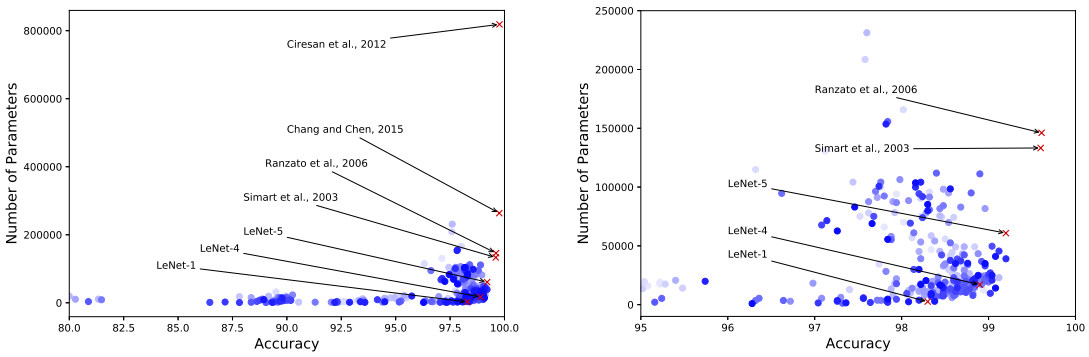
Population Size	20
Number of Generations	20
$p_{conv}^{(g)}$	0.5
$p_{pool}^{(g)}$	0.5
$p_{fc}^{(g)}$	0.1
$p_{crossover}$	0.5
p_{mutate}	0.25
λ - convolution	3
n_{min} - convolution	2
λ - pooling	0
n_{min} - pooling	1
λ - fully connected	10
n_{min} - fully connected	5

3.3.1 MNIST

The first experiment evolved architectures to perform classification of handwritten digits, using the MNIST dataset. Figure 3.2 shows the accuracy and parameter count of each evolved individual (in blue). The darkness of each point indicates which generation the individual was generated—lighter points were generated earlier in the experiment, while later individuals are darker. Additionally, results of hand-designed architectures in the literature are also presented as red points, namely LeNet-1, LeNet-4, and LeNet-5 [83], and architectures in Simard et al., 2003 [121], Ranzato et al., 2007 [110], Cireşan et al., 2012 [12], and Chang and Chen, 2015 [11].

Table 3.3 summarizes the accuracy and parameter count of these architectures.

Figure 3.2: Network architectures evolved for the MNIST dataset (left), and architectures with error rates less than 5% (right). Darker blue points indicate individuals from later generations, red points indicate hand-designed architectures from the literature.



Comparing the evolved results with the networks in [83], the algorithm presented is able to generate architectures which are superior in at least one attribute—a network was evolved that was more accurate than LeNet-4 while requiring fewer parameters (99.08% accuracy, 14,000 parameters), while a network as accurate as LeNet-5 requires ∼33% fewer parameters (99.2% accuracy, 39,002 parameters). Generally, the high accuracy models require models with a large number of features, either due to a significant increase in the number of feature maps at one or more layers [121, 110], using so-called "network in networks" [11] after each convolutional layer, or by using multiple networks [12]. Additionally, each reference cited augmented the data with some sort of distortion, which generally improves accuracy by ∼0.2%, while the approach used here was trained only on the original data.

While the algorithm presented here cannot produce models with comparable accuracy as the state-of-the-art models [12, 11], it should be noted that the results from the state-of-the-art models perform nearly perfectly on the MNIST dataset; the model in [12] requires over three times the number of parameters in [11], with an improvement in accuracy of only 0.01% (i.e., one test example). Additionally, the models in these references utilize architectures which cannot

Table 3.3: Accuracy and parameter count of CNNs trained to classify handwritten digits in the literature.

	Accuracy	**Parameter Count**
LeNet-1 [83]	98.3%	2,578
LeNet-4 [83]	98.9%	17,000
LeNet-5 [83]	99.2%	60,840
Simard et al., 2003 [121]	99.6%	133,260
Ranzato et al., 2006 [110]	99.61%	146,104
Cireşan et al., 2012 [12]	99.77%	818,650
Chang and Chen, 2015 [11]	99.76%	263,658

be modeled by the algorithm presented here, due to specialized network components (e.g., multi-column CNNs), or are not likely to be consistently evolved (e.g., network-in-networks). However, the approach used here does generate superior architectures to those in [83], which contains network components available to the algorithm presented here.

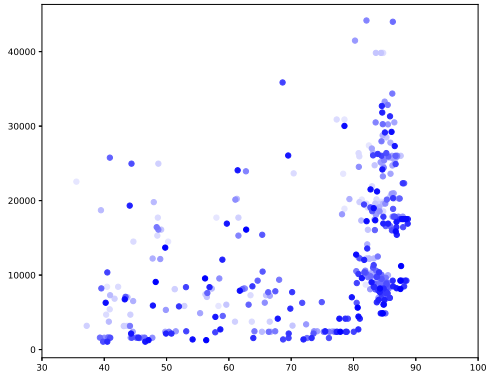
3.3.2 Terrain Data

The second experiment evolved architectures to classify terrains, using a dataset collected from a terrain sensitive tire. Figure 3.3 shows the accuracy and parameter count of each evolved individual. The algorithm evolved several efficient solutions with accuracies near 89%, such as an architecture with an accuracy of 88.3% and 9,252 parameters, and a slightly more accurate model with accuracy of 88.7% and 17,518 parameters. Interestingly, the former model achieves the low parameter count by using a pooling layer as the first layer following the input, hinting that the dataset could be downsampled without affecting model performance.

3.4 Discussion

The multiobjective optimization algorithm used in this chapter has been demonstrated the ability to generate CNN architectures which are optimal in a Pareto sense, primarily jointly minimizing the objective function and the parameter count of the network. The effectiveness of the

Figure 3.3: Network architectures evolved for the terrain classification dataset.



algorithm was demonstrated with two datasets, the MNIST handwritten corpus, and a set of terrain measurements collected by a terrain-sensitive tire. As shown in the MNIST results, networks can be generated that are as accurate as hand-engineered networks, but with significantly fewer parameters.

The MNIST dataset has been well studied since its initial publication, and many recent techniques, such as using a committee of networks, have been employed to increase the classification accuracy to the current state-of-the-art. The algorithm presented here could possibly be modified to incorporate such techniques, either by adding new types of layers to the genotype, or by evolving a connection matrix to allow, for example, skip connections or multi-column CNNs. The goal of this chapter for this thesis, however, is to provide an automated approach to discoving an efficient network architecture for the desired task. Additionally, it should be noted that this approach may be considered optional. A hand-engineered network might be easily developed which performs sufficiently for the desired task, and whose parameter count is well below the maximum available for a particular microcontroller.

Chapter 4

Single Node Robotic Material Experiments

The previous chapter provides motivation for performing in-material processing using convolutional neural networks for high-bandwidth sensing application, and an approach to generating network architectures which can be implemented on material-scale microcontrollers. This chapter provides example applications in three domains for robotic materials: a wearable gesture input device [61, 62], a smart tire capable of identifying terrain [20], and affective touch recognition and collision avoidance using robotic skins capable of pressure and proximity sensing [58, 60]. For each example, a brief background a motivation is provided, followed by the design of the prototype used and theory of operation (where applicable for each), and experimental results. Performances using alternative sensing modalities or machine learning are also provided for the skin and tire, to compare the classification abilities of a robotic material to current approaches available in the literature. Finally, memory requirements of each model is provided, to show that each model can be implemented on material-scale hardware. Details of each project can be found in the referenced papers.

The wearable gesture device and smart tire are both robotic material applications where the resulting component is decoupled from the remaining system and accessed via wireless communication, allowing the components to operate independently and be accessed as needed by the host system. In the case of the wearable gesture device, this involves pairing the component to another device via Bluetooth, while the smart tire maintains a TCP server, providing terrain estimates when queried by an arbitrary client.

The robotic skin application demonstrates a case where it is desired for the robotic material to perform multiple functions. The important considerations with this application is ensuring that a common feature set can be used to suitably perform the desired functions, the material can autonomously transition between dicrete states of operation, and the material communicates only events of interest to the host system.

4.1 e-Textile Input Device

An e-textile input device, referred to as "Switchback," is a device created as an example robotic material [61, 62]. Like robotic materials, wearable computing devices are often concerned with tight integration of sensing, computing, and acuation elements with the underlying fabric [107]. Specifically, conductive thread has allowed for creating circuit traces directly on fabric, as well a creating resistive and capacitive elements, switches, and buttons [38]. Switchback is designed to detect and identify a set of user touch gestures (here, tapping and bidirectional swiping), and does not require direct contact with skin to operate, allowing for waterproofing or covering with fabric.

4.1.1 Swatch Design

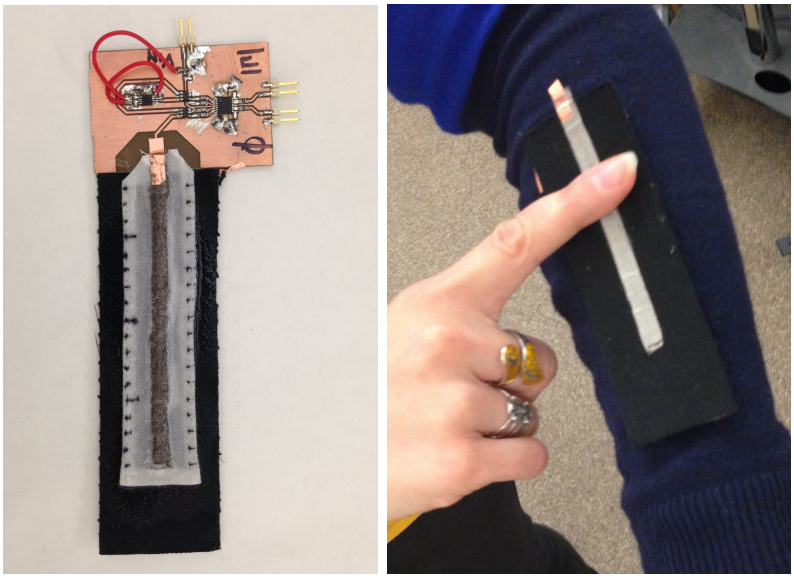
Switchback, shown in Figure 4.1 combines a simple microwave reflectometer circuit with a fabric-based microstrip quarter-wave short-circuit stub. The reflectometer circuit requires a one-sided microwave circuit with a footprint of roughly 10 cm^2; the reverse side of the circuit can contain a microcontroller, power regulation and Bluetooth or WiFi components to process measured signals and communicate to an external device. The reflectometer circuit consists of a Voltage Controlled Oscillator (VCO), two directional couplers, a gain-phase detector [1] . The gain-phase detector has an operating frequency range of 100MHz–2.7GHz.

The e-textile swatch is a microstrip stub consists of a 2-mm layer of iron-on adhesive denim as a dielectric layer, a ground plane constructed from Rip-Stop conductive metalized nylon fabric,[2]

[1] Analog Devices AD-8302: LF–2.7 GHz RF/IF Gain and Phase Detector

and a 6.8 cm x 0.635 cm conductive strip of Rip-Stop conductive fabric. The length of the strip corresponds to one quarter of the wavelength of a 900MHz RF signal on the microstrip. The conductive strip was shorted at one end using 117 / 17 2-ply conductive thread,[3] and an ultraminiature coax (UMC) line was attached to the other and the ground plane as a measurement port.

Figure 4.1: Left: e-Textile swatch with reflectometer circuit. Right: Example swipe gesture of swatch on the forearm of a shirt sleeve.



4.1.2 Theory of Operation

The reflectometer circuit measures the reflection coefficient at the measurement port of the microstrip stub by injecting a 900MHz signal into the measurement port. Some or all of this signal is reflected back into the circuit. A directional coupler routes the reflected signal into one input of the gain-phase detector. A second directional coupler routes a small portion of the signal generated by the VCO into the other input of the gain-phase detector for use as a reference signal. The gain-phase detector generates two voltages: one related to the ratio of the magnitudes of the reflected signal, and a second related to the phase difference between the two signals.

[2] http://www.sparkfun/products/10056
[3] http://www.sparkfun.com/products/retires/8554

The reflection coefficient is a function of the input impedance of the microstrip stub, which is a function of the characteristic impedance, propagation constant, and length of the stub. Covering the microstrip with a dielectric material changes the effective permittivity of the microstrip line where touched. This results in a change of the characteristic impedance and propagation constant, which results in a change of the reflection coefficient. The permittivity of human skin, compared to free space, is very high (ϵ_r $44.5 - j18.8$) [41], while the permittivity of denim is much lower ($\epsilon_r = 1.67$) [118]. Using these two values, the transmission line properties of a microstrip when touched and untouched can be calculated using a conformal mapping method [128]. Table 4.1 provides the effective permittivity, characteristic impedance and propagation constant of the microstrip swatch when untouched and touched by an adult human finger (1.6 cm–2.0 cm in width).

Table 4.1: Effective permittivity, characteristic impedance, and propagation constant of an e-textile microstrip when touched and untouched.

Fingertip Width	ϵ_{eff}	$Z_0(\Omega)$	$\gamma(rad/m)$
Untouched	1.50	48.56	$0.0 + j23.10$
1.6 cm	$9.54 - j2.57$	$18.84 + j2.50$	$7.78 + j58.78$
1.8 cm	$10.04 - j2.62$	$18.28 + j2.62$	$8.60 + j60.38$
2.0 cm	$10.40 - j3.23$	$17.90 + j2.72$	$9.34 + j61.54$

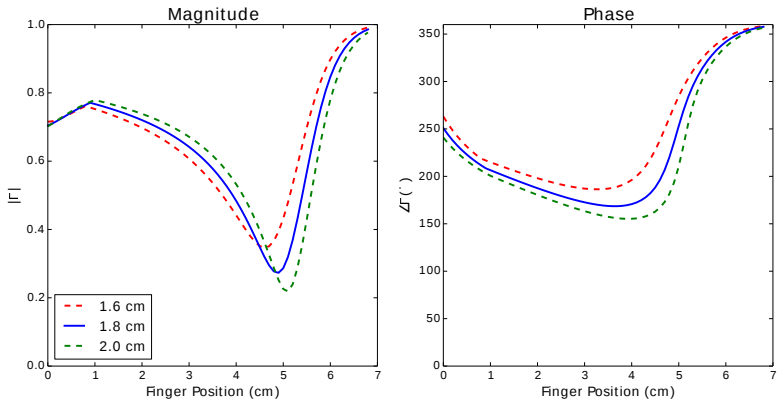
Using the parameters in Table 4.1, the magnitude and phase of reflection coefficient at the measurement port can be calculated as a function of the position of a finger on the line. These measurements are given in Figure 4.2, showing a significant change in reflection coefficient as the finger moves down the line.

4.1.3 Gesture Classification

The reflectometer was sampled at a rate of 30Hz using a development board[4] with an Atmel Mega328P microcontroller, operating at 16MHz with 32kB program memory and 2kB internal RAM. Gesture classification was performed using a convolutional neural network implemented in

[4] Arduino Pro Mini 328: http://www.sparkfun.com/products/11113

Figure 4.2: Magnitude (left) and phase (right) of reflection coefficient of a quarterwave e-Textile microstrip stub as a function of contact position.



program memory, and a Bluetooth modem[5] transmits the final classification to an arbitrary device. The CNN architecture used to classify gestures was [(I,(50,1)), (C,10,5), (P,4,4), (C,5,3), (FC,12), (S,3)]; a sigmoid activation function was used in place of a rectified linear activation function.

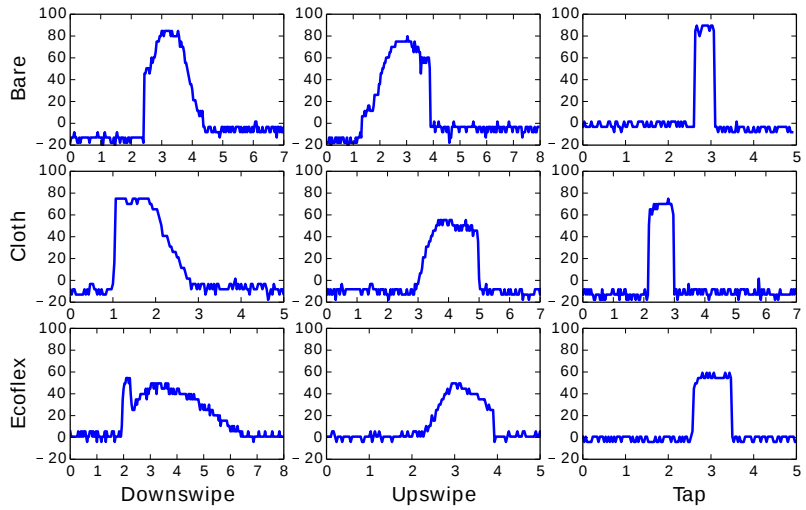
4.1.4 Experimental Results

A preliminary data set was collected by having a single subject perform three gestures—down-swiping (moving the finger from the short-end of the stub to the measurement port), up-swiping, and tapping. Additionally, gestures were collected with the swatch covered with fabric, as well as embedded in EcoFlex silicone rubber for weatherproofing. Figure 4.3 shows characteristic examples of each of these gestures, and shows little variation between the three configurations in the gesture signals.

From the data set, 60 tap gestures, 32 up-swipe gestures and 26 down-swipe gestures were collected and hand-labeled. The onset and conclusion of each gesture was determined when the measurement exceeded the baseline (non-contact) voltage. The samples were augmented by adding uniform noise with a range of 20% of the total measurement range, resulting in a total of 120

[5] BlueSMIiRF Silver: https://www.sparkfun.com/products/12577

Figure 4.3: Characteristic gesture signals for down-swiping, up-swiping, and tapping, for three configurations of the swatch.



samples for each gesture. The CNN was trained and evaluated using 10-fold cross-validation, with a final overall model accuracy of 96.11%. The confusion matrix of the classifier is given in Table 4.2.

The total number of parameters required for the model is 256, which requires only 1kB of program memory on a microcontroller to implement using 32-bit values. Consequently, a microcontroller which is much smaller, less expensive and consumes less power (e.g., AVR ATtiny44) could be used for this controller.

4.2 Terrain Sensitive Tire

Prototype terrain-sensitive tires were designed for use with "Ninja Car," a highly agile autonomous vehile, shown in Figure 4.4 (left). The vehicle is designed to map and navigate in unknown environments, and uses a simulation-in-the-loop model predictive controller to generate control signals. The physics simulator used by the controller accurately predicts the dynamics of the vehicle, allowing control signals to be generated for highly agile driving [96]. The ability to determine the terrain driven on allows more accurate parameters to be used by the physics simulator, improving

Table 4.2: Confusion matrix of the SwitchBack classifier.

		Predicted		
		Tap	Upswipe	Downswipe
Actual	Tap	**94.17%**	2.50%	3.33%
	Upswipe	0.83%	**98.33%**	0.83%
	Downswipe	3.33%	0.83%	**95.83%**

the performance of the controller.

Figure 4.4: Autonomous vehicle with terrain sensitive tire inset (left). An individual piezoelectric sensor (center). Location of a single sensor mounted on the interior of the tire (right).

4.2.1 Data Collection and Preprocessing

Two prototype tires were created, and datasets were collected for each. Piezoelectric sensors (Measurement Specialties, Inc. LDT0-028K) are bonded to the interior surface of the tire using polyurethane rubber (Smooth-On PMC-780); Figure 4.4 (right) shows the positioning of the sensor with a portion of the tire cut away. The piezoelectric sensors are sampled using a microcontroller board attached to the face of the wheel.

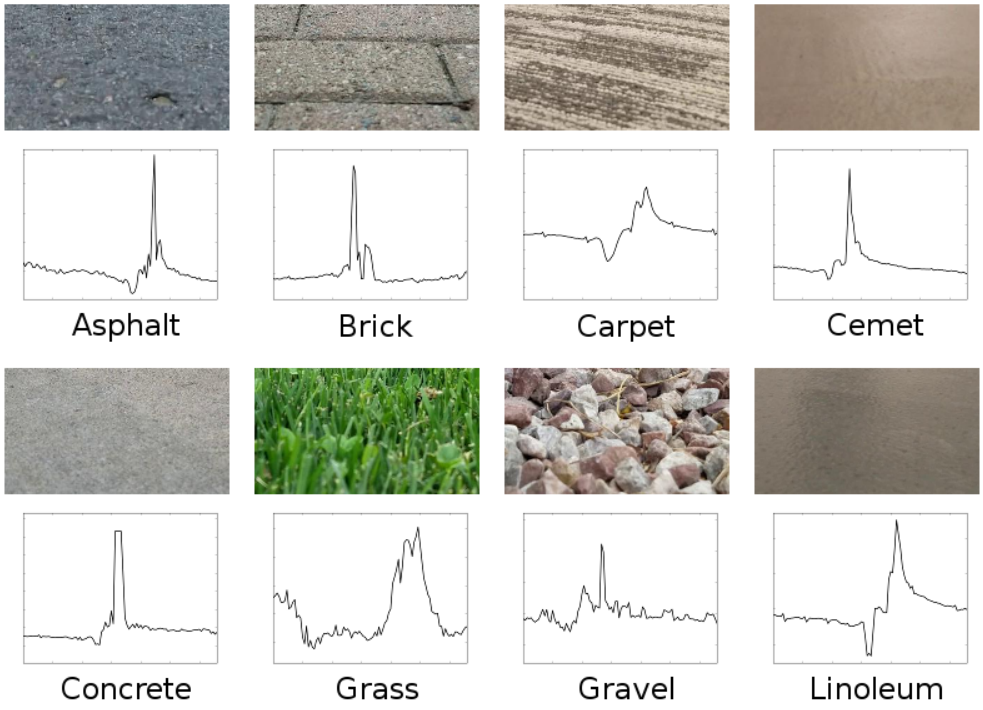
4.2.1.1 Indoor and Outdoor Terrains

The first prototype tire created consisted of ten piezoelectric sensors located around the radius of the tire, ensuring that at least one sensor is located at the point where the tire contacts

the ground. These sensors were sampled at a rate of 750Hz using a Teensy 3.2 development board, with raw data stored locally on a microSD card.

Data was collected while driving on eight different indoor and outdoor terrains: asphale, brick, carpet, cement (indoor), concrete (outdoor), grass, gravel, and linoleum. Additionally, data was collected while the car was suspended in the air, as the car is expected to be occasionally airborne and knowledge of this state is useful for the controller. The car was driven on each terrain for approximately 20 minutes each, for a total of three hours of driving time. Figure 4.5 shows images of each terrain, as well as a window of measurements from one of the sensors embedded in the tire. Measurement data was normalized so that each channel had zero mean and unit standard deviation. Windows of 100 samples (133.33 ms) were extracted every 50 samples (66.67 ms).

Figure 4.5: Examples of the indoor and outdoor terrains used for classification experiments, as well as example sensor signals for each terrain.



4.2.1.2 Simulated Terrains

A second prototype tire was created with four piezoelectric sensors located around the radius of the tire. These sensors were sampled at a rate of 2kHz using an ESP32 development board. This board contains on-chip WiFi and Bluetooth LE components, allowing for data to be collected wirelessly, as well as sending terrain estimates wirelessly to the vehile. The vehicle was driven on four simulated terrains with this tire: carpet, foam, grass, and plywood. The car was driven for ten minutes on each terrain. Windows of 100 ms (200 samples) were extracted every 50 ms, for a total of 24,472 samples.

4.2.2 Terrain Classification Results

4.2.2.1 Indoor and Outdoor Terrains

A classifier was built to identify terrain from a window of tire msensor measurements using a CNN with the architecture [(I,(200,10)), (C,20,10), (P,2,2), (C,10,15), (P,2,2), (C,5,20), (P,2,2), C(3,25), (P,2,2), (FC,15), (S,9)]. The model contained a total of 10,979 parameters.

The accuracy of the CNN classifier, as well as the number of parameters, is provided in Table 4.3. Additionally, the results available in the literature for vibration-based [137], visual-based [70], and audio-based [135] are provided for comparison.

Finally, a confusion matrix for the classifier is provided in Table 4.4. Accuracy for each class if relatively high (near 99%), with a slight decrease from the average for gravel and linoleum.

4.2.2.2 Simulated Terrains

A classifier evolved in Chapter 3 was selected to classifiy the simulated terrain data. The selected architecture achieved as classificaiton accuracy of 88.7% on the validation data using 17,518 parameters. The CNN architecture used was [(I,(200,4)), (C,23,20,2), (P,2,1), (C,29,19,3), (P,2,1), (C,1,15), (C,1,24), (C,1,19), (C1,36), (S,4)]. The last four convolutional layers (with kernel width of 1) create an architecture similar to the "network-in-networks" architecture in [11]. Once trained,

Table 4.3: Accuracy and classifier size (number of parameters) for each sensing modality.

	Accuracy	Model Size
Tire	**98.93 ± 0.17**	10,979
Valada et al. [135]	97.36 ± 0.12	
Weiss et al. [137] — Feature Set	94.71 ± 0.22	
Weiss et al. [137] — PSD	93.90 ± 0.37	
Weiss et al. [137] — FFT	93.40 ± 0.33	
Weiss et al. [137] — Combined	95.11 ± 0.31	
Khan et al. [70] — SURF	99.2	
Khan et al. [70] — Daisy	79.2	
Khan et al. [70] — LBP	96.9	
Khan et al. [70] — LTP	98.1	
Khan et al. [70] — LATP	97.2	

Table 4.4: Confusion matrix of the classifier trained on indoor and outdoor terrains.

	Predicted Class								
	A	B	C	D	E	F	G	H	I
A	**99.82**	0.01	0.00	0.01	0.00	0.00	0.07	0.06	0.02
B	0.00	**98.99**	0.66	0.01	0.07	0.00	0.09	0.07	0.12
C	0.00	0.86	**98.87**	0.04	0.01	0.00	0.01	0.12	0.09
D	0.00	0.00	0.01	**98.78**	0.27	0.00	0.27	0.01	0.66
E	0.00	0.03	0.02	0.13	**99.65**	0.00	0.01	0.08	0.08
F	0.00	0.00	0.02	0.00	0.02	**99.95**	0.00	0.01	0.00
G	0.03	0.11	0.00	0.31	0.04	0.00	**98.46**	0.14	0.91
H	0.15	0.13	0.36	0.08	0.34	0.12	0.37	**97.56**	0.87
I	0.02	0.12	0.09	0.99	0.21	0.00	0.78	0.26	**97.53**

(A) Air (B) Asphalt (C) Brick (D) Carpet (E) Cement
(F) Concrete (G) Grass (H) Gravel (I) Linoleum

this architecture produces an overall classification accuracy of 88.26%. Table 4.5 show the confusion matrix for the classifier. Confusion between cement and grass is observed, with a much higher classification accuracy for foam and wood.

Table 4.5: Confusion matrix of the classifier trained on simulated terrains.

	Predicted Class			
	A	B	C	D
(A) Cement	**81.63**	1.91	13.43	3.03
(B) Foam	2.31	**93.32**	3.46	0.91
(C) Grass	13.88	2.26	**82.34**	1.53
(D) Wood	1.80	1.14	1.55	**95.03**

4.3 Proximity Sensitive Skin

A modular robotic skin with pressure and proximity sensing capabilities was developed to assist a robot with collision avoidance and tactile interaction with a human companion. Modular tactile sensitive skin cells have been utilized for full-body tactile sensing, resulting in several applications in manipulation, exploration, navigation, and human-robot interaction [19, 31, 68, 67]. Using full-body tactile sensing is useful for conveying emotion and intent, which is especially useful given the increase in therapeutic and compantion robots [143, 125, 8, 31], and collaborative human-robot teams [13].

The main purpose of this experiment is to demonstrate a robotic material capable of performe multiple functions, namely, a robotic skin with the ability to detect approaching objects, discriminate between an obstacle and the hand of a collaborator, and identify touch gestures. The approach used to achieve this involves having the skin transition between behavior states based on a low-dimensional feature vector extracted from individual frames. Communicated events and skin behavior can then be defined based on the current state of the skin. Additionally, the behavior of the system as a whole can be analyzed, allowing important attributes to be extracted, such as the maximum velocity the robot can safely move without inadvertantly colliding with an obstacle.

An interesting interpretation of convolutional neural networks is that each layer represents feature maps of increasing abstraction. The penultimate layer can be considered a low-dimensional feature vector summarizing the input to the system. As demonstrated in [111], CNNs trained on one

set of images can be effectively utilized to perform classification on other types of images by replacing the final classification layer with an SVM, and simply training the SVM on the features generated by the pre-trained CNN. The features generated by CNNs trained to perform classification in robotic materials can be similarly exploited to generate more complex behavior in the material by using them as common features for a set of models trained for different tasks.

4.3.1 Prototype Skin

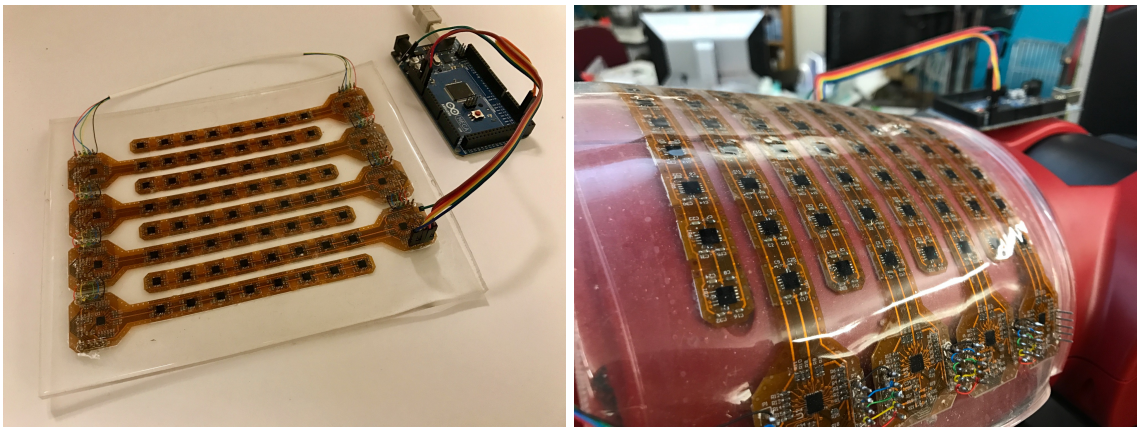
A pressure and proximity sensitive skin patch was created for this investigation, similar to the pressure sensitive skins described for the affective touch classification task in [67]. Figure 4.6, left, shows a skin consisting of an 8x8 array of proximity and ambient light sensors (Vishay Semiconductors VCNL4010 [6]). Each sensor combines an infrared emitter and PIN photodiode to perform proximity and ambient light measurement. The skin is calable of detecting both the proximity of an approaching object, as well as the amount of force applied to the skin once the object makes contact. Details on the operation of an individual sensor is given in [104] and [60]. The microcontroller samples frames of measurements at a rate of 20 Hz.

The skin is embedded in a thin film of PDMS (Dow Corning Sylgard 184), with a final overall width and height of 10.8 cm x 10.8 cm, and a thickness of 5.0 mm. A collocated microcontroller (Atmel ATmega2560) samples the sensor array, and performs in-skin processing to allow the skin to detect and avoid collisions with arbitrary obstacles, recognize whether an approaching object is an obstacle or a human collaborator, and classify six social touch gestures made by a collaborator, similar to the affective touch example in Chapter 4.

Calibration and preprocessing of the sensor is performed on the microcontroller. The microcontroller maintains a "baseline" value for each sensor (the average measurement in the absence of an object), as well as a "contact" values (the value of the sensor at the point where contact is made). Sensors values were scaled such that the baseline value and the maximum sensor value were mapped to values of -1 and 1, respectively.

[6] http://www.vishay.com/docs/83462/vcnl4010.pdf

Figure 4.6: Left: Pressure- and proximity-sensitive skin with an 8x8 array of taxels. Right: Skin mounted on the forearm of a Baxter robot.



4.3.2 Algorithmic Approach

The skin is designed to perform three main functions: detect an approaching object, discriminate between an obstacle or hand (from a human collaborator), and identify affective touch gestures performed on the skin. The skin extracts a set of 15 frame-level features per measurement frame: mean, standard deviation, median, mode and range of values, as well as geometric moments up to the 4^{th} order [131]. Gesture-level features were extracted by computing the mean, standard deviation, median, and range of each frame-level feature in a window of measurements, resulting in 60 features per frame.

4.3.2.1 Approach Detection

The skin determines an obstacle or hand is approaching whenever any sensor value is above some threshold of its baseline value in a particular frame. The threshold for sensor i, $T_{prox}^{(i)}$ is calculated such that the false-positive detection rate of the entire skin patch is at 1% or below.

This is determined for each sensor by finding the value k such that the following equation is satisfied

$$\begin{aligned} T_{prox}^{(i)} &= \mu^{(i)} + k\sigma^{(i)} \\ \text{s.t.} \quad P\left(\bigcap_{i=1}^{N}\{s^{(i)} > T_{prox}^{(i)}\}\right) &< 0.01 \end{aligned} \tag{4.1}$$

where $s^{(i)}$ is the value of sensor i, and $\mu^{(i)}$ and $\sigma^{(i)}$ are the mean and standard deviation of the value of sensor i with no object present.

4.3.2.2 Hand / Obstacle Discrimination

Discriminating an approaching object as a hand or obstacle is critical for the performance of the skin, as it determines if contact should be allowed to be made or avoided. The skin needs to estimate the class of an object (hand or obstacle) at each frame, and update this estimate as future frames become available. While possible to train a binary classifier for such a task, this approach is not ideal. Primarily, the obstacle dataset would need to include all obstacles expected to be in the environment, the set of which may not be known during training.

The task of distriminating between an obstacle and hand is better suited for novelty detection [87], where a model is trained to identify the likelihood of a measurement belonging to a particular class. For this skin, this involves training a model on frames collected from gesture captures before contact is made, and identifying measurements below a certain likelihood as belonging to an obstacle. Two approaches to novelty detection were considered: Mahalanobis squared distance (MSD) [140], and one-class SVMs using Radial Basis Function (RBF) kernels [119].

Both models generate a distance measurement given the frame-level features of a measurement. The frame is identified as a hand or obstacle depending on if this distance falls inside or outside a given decision boundary. Given a distance measurement, x, and a decision boundary, D_T, the probability that a hand is approaching is

$$P(H|x) = e^{-0.693x/D_T} \tag{4.2}$$

Multiple frames will be available to determine if the approaching object is a hand or obstacle, allowing Bayesian updates of aninitial estimate as each frame is observed. The sequence of predictions can be treated as Bernoulli process due to the presence of a hand with some probability distribution θ. The conjugate distribution of θ is simply the Beta distribution,

$$p(\theta; \alpha, \beta) = \frac{\theta^{\alpha-1}(1-\theta)^{\beta-1}}{B(\alpha, \beta)} \tag{4.3}$$

where $B(\alpha, \beta)$ is the Beta function, and the parameters α and β are calculated from N frames as the total number of frames classified as a hand or obstacle, respectively. The probability of an approaching object being a hand, given a history of N frames, is given simply as

$$P(H|x_0 \dots x_N) = \frac{\alpha}{\alpha + \beta} \tag{4.4}$$

Details of the derivation of this is given in [60].

4.3.2.3 Gesture Classification

Gesture recogntion was performed on the extracted gesture-level features. A random forest classifier, implemented in Scikit [105], was trained and evaluated using 10-fold cross-valication and cross-subject validation, providing insight on the behavior of the classifier when presented with gestures from known and unknown users.

4.3.3 Experimental Results

Gesture data was collected from nine participants; the gestures used matched those used in the HAART dataset [8]. Each participant performed each gesture for 8 seconds a total of five times. Examples of two types of gestures are given in Figure 4.7. Additionally, obstacle data was collected using nine items, shown in Figure 4.8; obstacles were moved over a range of 0.0 cm to $\sim$5.0 cm over the skin, varying position and orientation at random.

Figure 4.7: Sequence of measurements from tapping (top) and rubbing (bottom). The five frames in the top show a single tap over 250 ms, the bottom row shows one back- and forth motion exting over 1s.

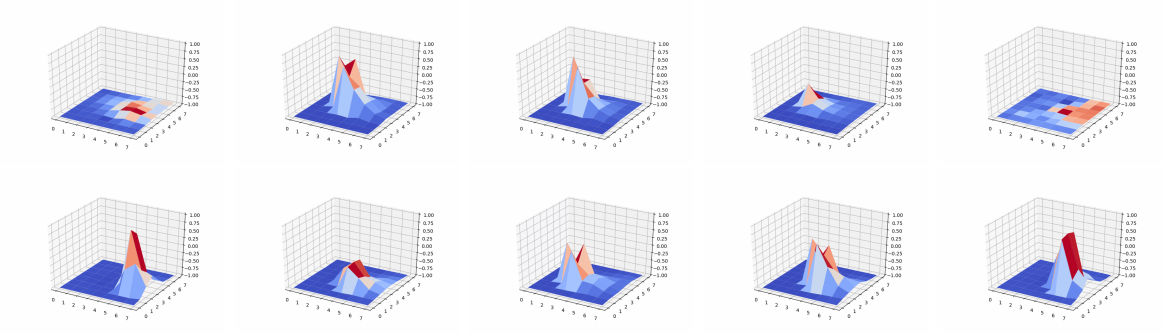


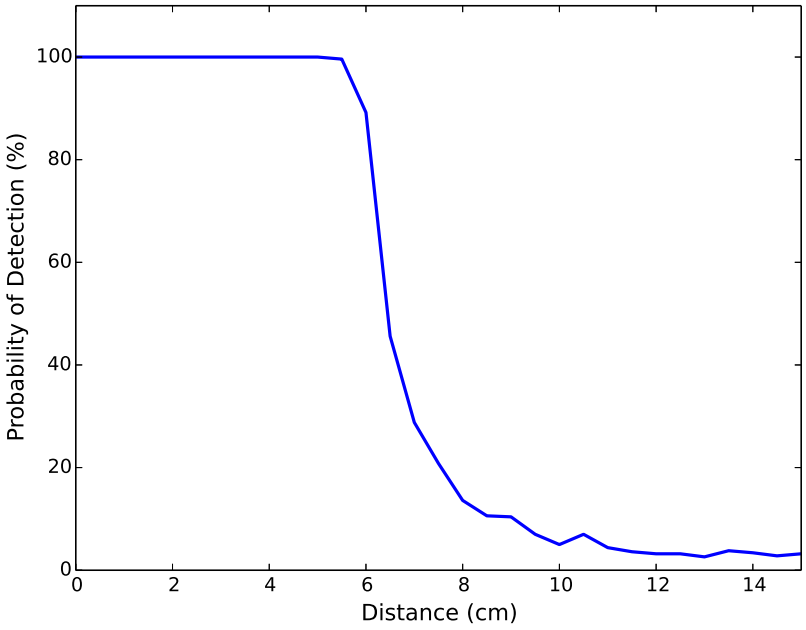
Figure 4.8: Objects used for collision recogntion testing: wooden plate, brick, PVC pipe, wine glass, plastic chain, foam balls, ball, screwdriver, and 2x2 wooden stick.

4.3.3.1 Approach Detection

To validate the ability of the skin to detect approaching objects, a set of measurements were collected where an MDF panel was located parallel to the surface of the skin, and moved between 0.0 cm and 15.0 cm, with measurements at every 0.5 cm increment. The panel was large enough to fully cover the skin, ensuring each sensor detected it. 500 frames were collected for each position of the MDF, with an additional 500 frames with no object present for use as a baseline.

Figure 4.9 shows the probability of detecting the approaching panel as a function of distance. Up to 5 cm, 100% of the frames correctly detected the panel; this dropped to ∼50% at 6.5 cm. Above ∼11 cm, the panel could not be reliably detected.

Figure 4.9: Probability of detecting an object as a function of distance to the skin.



4.3.3.2 Hand / Obstacle Discrimination

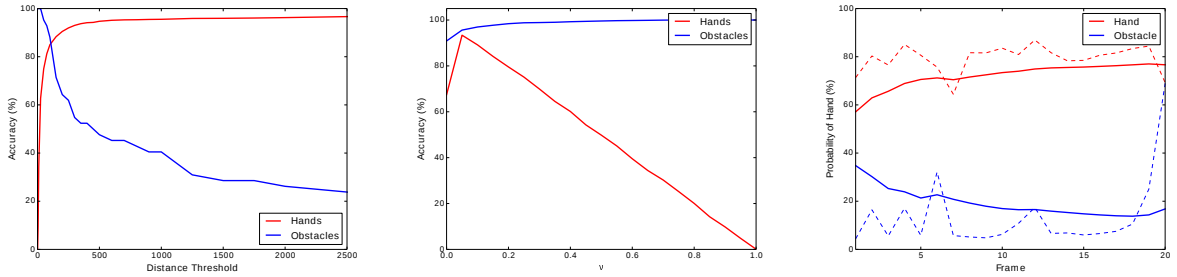
Training frames for hand and obstacle discrimination were extracted from the gesture data by selecting frames where at least one sensor exceeds T_{prox}, and no sensor exceeds the contact value. A total of 17,023 frames were collected from the gesture data; 4,721 frames of obstacle data was collected from the obstacle dataset. 90% of the gesture frames (15,320 frames) were used to train a MSD and one-class SVM discriminator; 10% of the gesture frames (1,702 frames) was retained for evaluation.

Figure 4.10a shows the accuracy of labeling hands and obstacles using an MSD discriminator with respect to the selected decision boundary. At a boundary of 250, the discrimination accuracies are roughly equivalent—hands are correctly identified 91.4% of the time, obstacles 91.8%. Figure 4.10b show the discimination accuracy of the one-class SVM as a function of ν, a parameter that provides an upper bound on training error, with a second parameter, γ (the width of the RBF kernel), set to 5e-8. A value of 0.05 was found to be optimal, providing an accuracy of 93.4% for hands and 95.6% for obstacles.

Figure 4.10c shows the benefit of performing Bayesian updates on the discrimination estimates. The probability of each individual frame being generated by an approaching hand or obstacle is given as dashed lines; the probability generated from the Bayesian update is given as solid lines. Essentially, performing Bayesian updates in this manner has the effect of smoothing individual probabilities, ensuring that no single frame results in a misidentification of the approaching object.

4.3.3.3 Gesture Classification

Two cases were considered for gesture classification: classifying using features extracted from an entire 8 second capture, and classifying using 1 second (20 sample) windows drawn from each capture. Four ranges of measurements were used to generate features. “Full measurements” consists of features extracted from the full calibrated range of the sensors. “Pressure only” and “proximity only” consisted of measurements in the pressure or proximity region—measurements above or below



(a) Mahalanobis Squared Distance (b) One-Class SVM (c) Bayesian Update

Figure 4.10: Per-frame discrimination accuracy using (a) MSD, and (b) One-Class SVM. (c) Demonstration of smoothing predictions over a sequence of MSD predictions using Bayesian updates with a hand (red) or obstacle (blue) approaching.

a sensor'c contact value were ctruncated to the contact value. "Pressure + proximity" concatenated features from the "pressure only" and "proximity only" data.

Figure 4.11 gives the evaluation of random forest classifiers trained on each set of data using each evaluation approach (cross-fold validation vs. cross-subject validation; features from full gestures vs. features from 10 second windows). Using only pressure measurement, accuracy drops considerably when compared to other cases. Using only proximity data performs only slightly worse when compared with a classifier trained on the full range of measurements, indiciating the advantage of incorporating proximity information into measurements for gesture recogntion. Cross-subject validation performed worse than cross-fold validation; this is to be expected, as cross-subject validation requires a model to generalize to data from users not previously encountered.

The confusion matrix for the windowed, full measurement dataset using cross-validation is given in Table 4.6. As with the affective touch example in Chapter 4, most confusion occurs between *tickle* and *scratch* pairs, and *rub* and *stroke* pairs.

4.4 System-Level Analysis

The tasks presented in Section 4.3.2 can be organized in such a manner to allow the skin to transistion between tasks in an appropriate manner. Figure 4.12 shows a state machine capable

Figure 4.11: Classification accuracy of a random forest classifier.

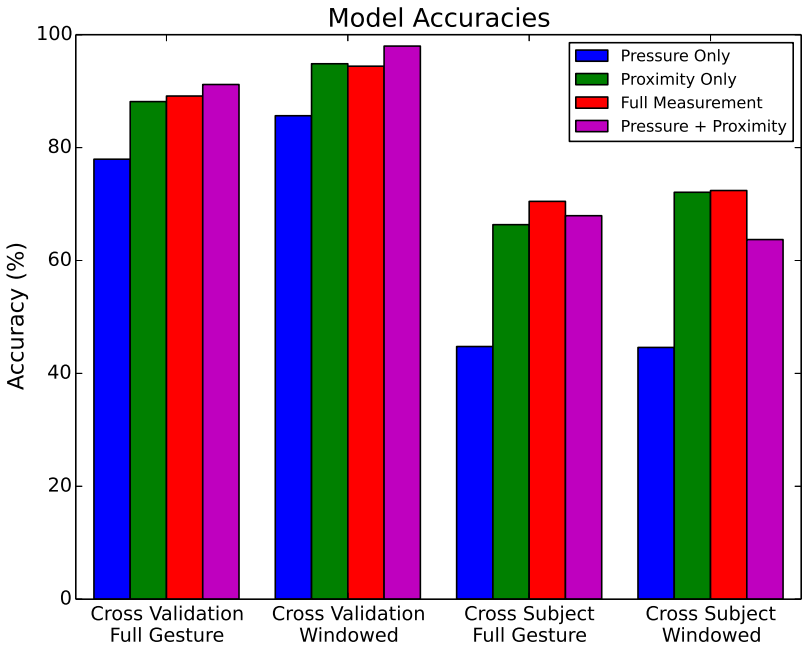


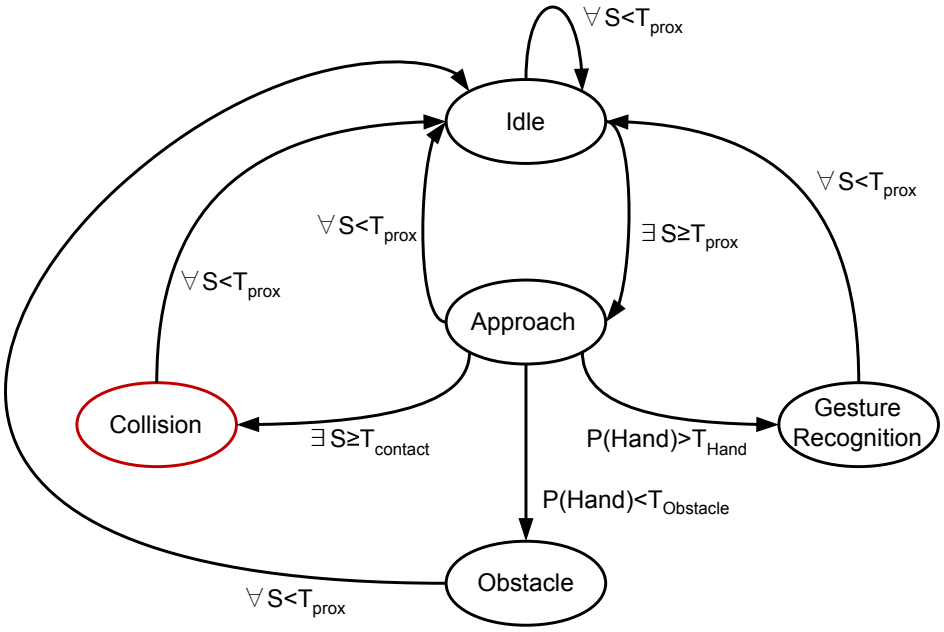
Table 4.6: Confusion matrix of gestures classified using features from the windowed, full measurement data.

		Predicted						
		a	b	c	d	e	f	g
Actual	(a) NoTouch	**100**	0.00	0.00	0.00	0.00	0.00	0.00
	(b) Press	0.00	**97.8**	0.85	0.85	0.12	0.36	0.00
	(c) Pat	0.00	0.41	**97.4**	0.68	0.55	0.82	0.14
	(d) Rub	0.00	0.66	2.08	**89.1**	2.08	4.38	1.75
	(e) Scratch	0.00	0.00	0.44	1.33	**93.9**	1.66	2.65
	(f) Stroke	0.00	0.00	1.13	5.13	3.63	**88.5**	1.63
	(g) Tickle	0.00	0.25	0.13	0.75	6.52	0.38	**92.0**

of governing the behavior of the skin. The skin is **Idle** until an approaching object is detected (Section 4.3.2.1). While approaching (**Approach**), the skin needs to determine if the object is a hand or obstacle (Section 4.3.2.2). If the object is a hand, the skin should identify (**Gesture Recognition**) which gesture is being performed (Section 4.3.3.3). Otherwise, the skin should

inform the robot that an obstacle is approaching (**Obstacle**), so that the robot can alter its current trajectory. If the skin fails to discriminate between a hand and obstacle prior to contact, then the system collides (**Collision**) with the object, which is an undesired state. Whenevery the skin no longer senses and object, it returns to the **Idle** state.

Figure 4.12: State machine representation of the behavior of the skin with S representing sensor data. $T_{Contact}$ is a fixed value for each sensor, T_{Prox} is determined empirically, and T_{Hand} and $T_{Obstacle}$ are user-defined thresholds.



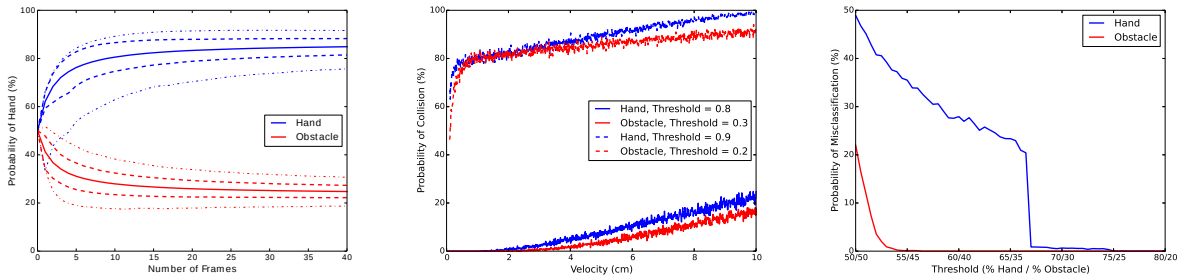
From the experiments, the stochastic properties of the transistion between each state in the state machine is well understood; it is possible to perform Monte Carlo experiments to determine the expected probability of collisions, allowing for the tuning of tresholds for identifying a hand or obstacle, T_{Hand} and $T_{Obstacle}$, until a suitable margin of safety is achieved.

Figure 4.13a shows the results of 10,000 simulations of discriminating between hands and obstacles using the MSD discriminator with Bayesian updates. Setting T_{Hand} to above 80% and $T_{Obstacle}$ to below 30% ensures that a majority of the instances are correctly classified; more liberal thresholds of 60% and 40% provide faster and more inclusive discrimination, though with a small percentage of hands misidentified as obstacles in the first few frames.

Given T_{Hand} and $T_{Obstacle}$, it is possible to determine the robustness of the skin to collisions with respect to the velocity of an approachin hand or obstacle. Figure 4.13b shows the collision probability with respect to velocity for the 80%/30% threshold, as well as a more strict 90%/20% threshold, for approach velocities in the range of 0.1 cm/sec to 20 cm/sec. This graph demonstrates that, for the less strict thresholds, velocities up to ∼2.0 cm/sec are safe, and collision rate increases to ∼20% at 10 cm/sec. Using a more liberal threshold of 60%/40% results in no collision, regardless of velocity, though hands were misclassified as obstacles an average of 4.8% of the time.

The probability of misclassifying a hand or obstacle as a function of the two thresholds is shown in Figure 4.13c. This graph demonstrates that hand threshold can be set as low as 55%, and obstacle threshold set to 30%, with almost no misclassification of hands or obstacles.

Figure 4.13: Results of the Monte Carlo simulations: (a) Probability of classifying objects as a hand—mean (solid), one standard deviation (dashed) and 1^{st} and 99^{th} percentiles; (b) Probability of a collision with respect to approaching object velocity, for midlevel and strict thresholds; (c) Probability of misclassifying hands and obstacles as a function of threshold.



(a) Bayesian Update (b) Collision (c) Misclassification

4.5 Discussion

The examples provided here demonstrate that in-material processing of sensing signal is possible using CNN models implemented on material-scale microcontrollers. In most cases, the needed microcontroller is expected to be located on a printed circuit board with additional components, e.g., the reflectometer circuit on Switchback. In this particular case, the added microcontroller does

not require a larger PC board to be incorporated—the microcontroller and related components can be located on the back side of the reflectometer circuit. Recently released low-cost system-on-a-chip microcontrollers with incorporated Wi-Fi and Bluetooth capabilities have become available—the Espressif Systems ESP32 used by the prototype terrain-sensitive tire is an example of such a chip. The incorporated wireless communication greatly simplifies computing nodes, as additional communication components are not required.

The networks implemented in these examples perform tasks well. Each example material has been shown to be able to discriminate between several classes of stimuli with accuracies near or above 90%. With regards to the first prototype tire, the trained model outperformed visual, vibration and audio based models. This is encouraging, as it implies that the tight coupling of the sensing with the tire material plays a more imporant role than the methods used to perform classification.

The ability to incorporate multiple functions is desirable for robotic materials, as this allows the material to exhibit behavior which is context sensitive. In this chapter, a robotic skin is designed to perform several functions (object detection, hand/obstacle discrimination, and gesture recognition), as well as adapting the behavoir of the host robot based on environmental conditions. The functions utilize the same high-level features extracted from individual frames. The dimensionality of this feature set makes adding additional functionality feasible—as the dimensionality increases, additional model parameters (e.g., for the MSD discriminator) would also increase. A common feature set with lower dimensionality would allow more functions to be incorporated without exceeding the momory capacity of the microcontroller, when compared with a feature set with higher dimensionality or multiple feature sets.

A final and significant result from this chapter is the use of the state machine to model context-sensitive behavior of the skin, as demonstrated by the robotic skin example. The transition probabilties of the state machine are readily extracted from the properties of the skin (e.g., probability of falsely detecting an object), and requires only two user-defined thresholds (T_{Hand} and $T_{Obstacle}$). Once provided, this model can be leveraged to perform holistic analysis of the skin-

robot system, determining attributes such as safe operation speed, and likelihood of inadvertantly avoiding contact by a human collaborator.

Chapter 5

Amorphous Computing for Robotic Materials

In general, robotic materials contain multiple nodes distributed in the material. Computation is performed in each node by processing local sensor signals, and communicating with nodes in a local neighborhood to determine the material response. Initial perspectives on the computational aspect of robotic materials consisted of designing local controllers based on a fundamental understanding of the underlying material dynamics—material was either described as a set of discrete lumped element models, or a continuous model described a set of distributed parameters [91]. For certain applications where the dynamics of the material are well understood, a distributed or amorphous algorithm may be relatively simple to derive for perception or control. This chapter describes an amorphous robotic skin designed to detect and localize contact due to material rubbed against the skin, and identify the texture of the contacting material, using an fully distributed, amorphous approach [53, 54]. The purpose of this chapter is to compare an amorphous approach to the modular CNN-LSTM model described in Chapter 6. Specifically, I identify key behavioral requirements of individual nodes, advantages and limitations to an amorphous approach, and conditions in which the modular CNN-LSTM model would be preferred.

5.1 Skin Design and Manufacturing

A large number of robotic skins have been designed with a variety of sensing modalities. Pressure detection is a primary sensing modality [138, 146, 9, 23, 45, 98, 75, 126]; less common sensing modalities include shear detection [95], temperature [126], vibration [116], and proximity

or "pre-touch" [86]. The skin presented in this chapter is inspired by the Pacinian corpuscles found in glabrous human skin. The Pacinian corpuscles are mechanoreceptors with a very wide receptive field that respond to high-frequency vibrations, and are the primary means of perception of various textures in human skin [66, 4]. The choice to design vibration-sensitive skin is especially relevent from a robotic materials perspective: propagation of vibrations through the skin allows for extracting information of interest from a sparse set of sensors, whereas pressure detection requires a dense array of sensors to detect pressure distribution.

The prototype skin, shown in Figure 5.1, consists of a network of ten sensor nodes embedded into silicone rubber (Ecoflex^{TM} Supersoft 0030). The final dimensions of the skin are 61 cm x 43 cm, with a final thickness of $\sim$1 cm.

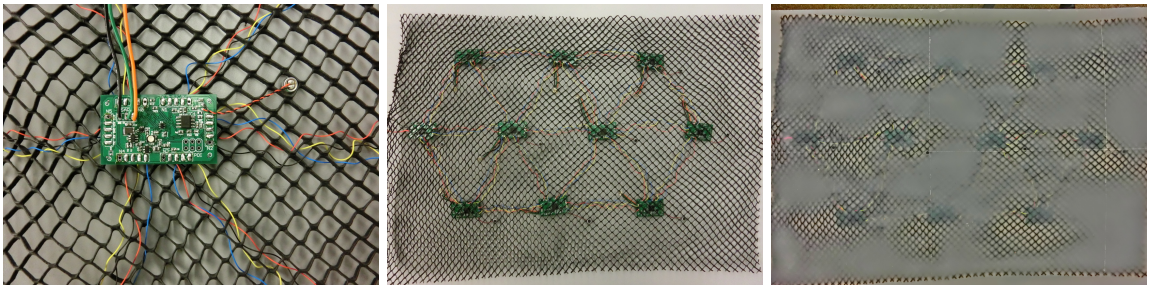
Figure 5.1: Amorphous skin mounted on the back of a Baxter robot.

Each sensor node in the skin contains an Atmel ATxmega128A3U microcontroler, an omnidirectional microphone with a sensitivity of -45dB and signal-to-noise ratio of 58dB, and an amplification circuit. The microcontroller samples the amplified microphone signal at a rate of 1kHz using a 12-bit analog–digital converter, and maintains a circular buffer of 256 samples. The

microcontroller contains six hardware serial ports (USART) capable of communicating at a maximum rate of 115kbps. Figure 5.2, left, shows an individual sensing node.

A network of ten sensor nodes are connected in an equilateral triangular lattice to create a network, shown in Figure 5.2, center. Connections between nodes consist of four wires for power, ground, and serial communication. Nodes are spaced 15 cm apart in the network. The network wires are woven into a neoprene mesh, which is then embedded into silicone rubber. The surface of the skin is textured by lining the form in which the skin is made with 60-grit aluminum oxide sandpaper. The textured surface is designed to mimic the surface of the human fingertip: the grit size of the sandpaper roughly corresponds to the distance of ridges in the fingertip [49].

Figure 5.2: Left: close-up of an individual sensor node. Middle: ten element sensor network, woben into a neoprene support mesh. Right: sensor node network embedded into EcoFlex^{TM} silicone rubber.



5.2 Skin Operation and Dynamics

The operation of the skin involves local monitoring of vibrations at each node, which are generated when the skin is tapped or rubbed. When a node's circular buffer is filled, the spectrum of the signal is computed using the Fast Fourier Transform (FFT), resulting in a 128 bin spectral measurement being generated at a rate of $\sim$4 Hz.

5.2.1 Transient Signal Detection

The skin is expected to operate while attached to a robot, as well as in environments where background noise is present. To distinguish between these potential sources of vibrations and signals generated due to contact, it is necessary to maintain an estimate of the background noise at each node. This is computed in a similar manner to that presented in [88]. Each node maintains an estimate of the spectrum of the *ambient* signal, $A(f,t)$, at each time step when the spectrum of the measured signal, $S(f,t)$ is computed. The ambient signal is updated using slow averaging

$$A(f,t) = \alpha S(f,t) + (1-\alpha)A(f,t-1) \tag{5.1}$$

where α is a smoothing constant representing the rate at which the ambient signal responds to recent changes in background noise; high values results in an ambient signal sensitive to recent events, while lower values result in an estimation which responds more slowly to changes in background noise. A value of 0.05 is used as a smoothing constant, as suggested in [88].

Any signal generated due to contact is assumed to generate a momentary increase in the measured signal at nodes near the source of contact. This *transient* signal is assumed to be statistically independent of the ambient signal, so that the measured signal is simply the sum of the ambient and transient signals. At each time step, the transient spectrum, $T(f,t)$ is extracted from the signal

$$T(f,t) = max(S(f,t) - A(f,t), 0) \tag{5.2}$$

where the *max* operation simply ensures the transient spectrum is non-negative.

5.2.2 Vibration Propagation

As the skin is of constant thickness, it is trivial to model the propagation of vibrations through the skin. In practice, the dimensions of the skin will generally be larger in terms of signal wavelength, and vibration intensity will decrease rapidly as a function of distance. For simplicity,

the skin is considered a thin vibrating plate infinite in extent. Under this model, the sound intensity of a vibration propagating through the skin is given as

$$I(r, \omega) = \frac{I_0(F, \omega)}{r} \tag{5.3}$$

where $I_0(F, \omega)$ is the intensity of the vibration due to a source with displacement force F and frequency ω, and $I(r, \omega)$ is the intensity of the signal a distance r from the source. I_0 is dependent on the mechanical properties of the skin (i.e., thickness, density, and rigidity), which are constant for our skin. Details on the derivation of this equation are available in [54].

This equation is validated by pressing a small vibration motor (1 cm x 2 mm) against the skin at various distances from a sensor node. The peak energy of the motor vibration occurs at a frequency of 150 Hz. For each distance, 15 measurements were made. Figure 5.3 shows the measured signal, as well as the intensity equation given in Equation 5.3 at a frequency of 150 Hz. The unknown value of $I_0(F, \omega)$ is determined using a least-square fit of the average measurements to Equation 5.3. This demonstrates that the propagation model is sufficiently accurate to use for a localization algorithm.

5.3 Algorithmic Approach

The skin is designed to continually monitor vibrations generated in the skin, and provide high-level information to the host robot only when events of interest occur. Each node is assumed to have a unique ID, as well as know its physical location in the skin.

5.3.1 Finite State Machine

Each node is modeled as a finite state machine, as shown in Figure 5.4. The purpose of modeling node behavior as a state machine is to ensure that every node can operate independently of other nodes, and thus ensuring global behavior which is robust to individual node failure.

The behavior of each state is detailed below:

Figure 5.3: Amplitude of the vibration intensity due to a vibration motor as a function of distance between the motor and sensor node.

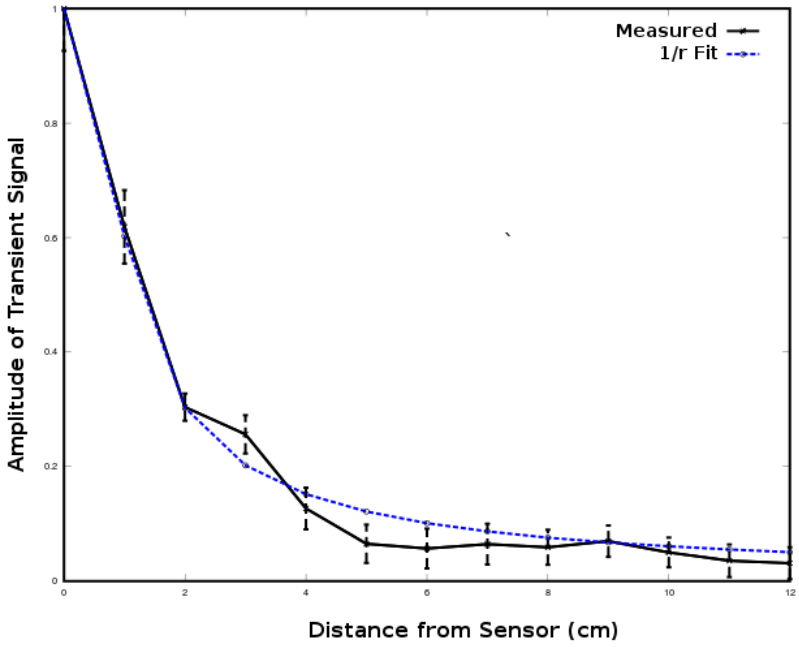
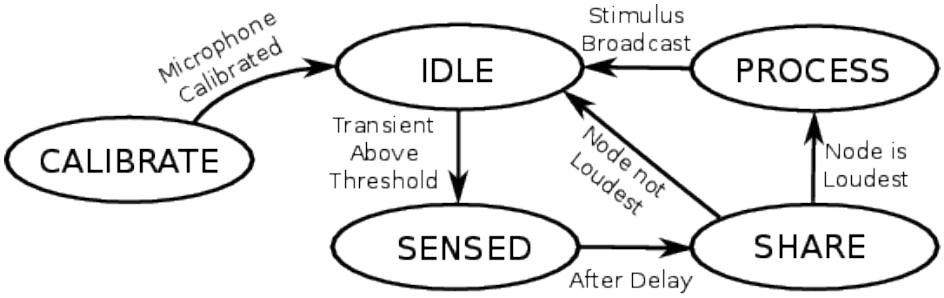


Figure 5.4: Finite state machine model of individual sensor nodes.



- **CALIBRATION.** When initially powered, nodes begin in the **CALIBRATION** state. In this state, the node initializes the signal buffer and allocates memory for the various spectral windows. Additionally, the initial level of the ambient spectrum is computed using a number of valid measurement windows. Once calibrated, the node enters the **IDLE** state.

- **IDLE.** While in the **IDLE** state, nodes sample the microphone, compute the spectrum of the signal, and update the ambient and transient spectrums. The node remains in this state until the the transient spectrum exceeds a predefined threshold at one or more frequencies. At this time, a signal is considered to be detected, and the node enters the **SENSED** state.

- **SENSED.** The purpose of the **SENSED** state is to ensure that vibrations have propagated through the skin, and the transient spectrum recorded by nodes capable of detecting the signal. While in the **SENSED** state, the node waits for a random amount of time between 5 and 25 milliseconds. This delay ensures that neighboring nodes have time to process their local measurements, and helps balance communication load in the network by ensuring that a group of node do not simultaneously attempt communication. Once this delay has finished, the node enters the **SHARE** state.

- **SHARE.** When a node enters the **SHARE** state, it broadcasts a packet containing its ID, location and transient spectrum. Additionally, while in this state, the node receives similar packets from neighboring nodes. A node maintains a table containing its own information and neighboring information, inserting an entry into the table when a packet with a previously unseen ID is received. After 50 milliseconds, it is assumed that the node will have recieved packets from nodes which have detected the vibration. At this point, the node determines if its total transient energy (i.e., the sum of the transient spectrum) exceeds that of all other nodes. If so, the node enters the **PROCESS** state, otherwise, it returns to the **IDLE** state.

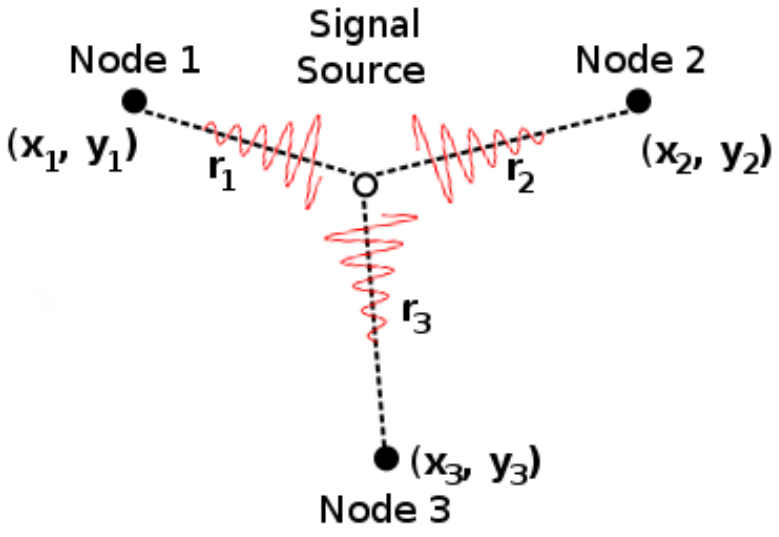
- **PROCESS.** In the **PROCESS** state, a node estimates the contact location (described in Subsection 5.3.2) as well as the texture of the material making contact (described in Subsection 5.3.3). Once this information has been calculated, the node broadcasts a packet containing the estimated contact location and material class, and returns to the **IDLE** state.

Ensuring each node remains in one of the above states ensures robust and efficient operation of the skin. As there is no explicit requirement for neighboring nodes to be in a particular state, a node's operation is robust to the introduction or failure of neighboring nodes. While in the **IDLE** state, nodes ignore packets received by neighboring nodes entering the **SHARE** state; thus, the network uses only the nodes which have detected a contact event, automatically limiting communication of packets to the neighborhood of nodes capable of using such information. Finally, the **SHARE** state automatically elects a single node (theoretically, the node physically closest to the contact location) to perform localization and classification, allowing all other nodes to resume monitoring the skin for contact events.

5.3.2 Contact Localization

A vibration generated due to contact with the skin will propagate through the skin, and will be detected by sensor nodes where the sound intensity is sufficiently high. Figure 5.5 illustrates this with three sensor nodes.

Figure 5.5: Propagation of sound to sensor nodes from an arbitrary source location.



Sound intensity propagates through the skin according to Equation 5.3. One approach to

determining the location of the source is to determine the source location, (x, y), which minimizes the norm of the residuals between the measured sound intensity and that predicted by Equation 5.3. However, this requires either knowledge of the sound intensity at the source (I_0), or including this as independent variables. Alternatively, this variable can be removed entirely by considering measurements at pair of nodes. The source location can then be determined by the following optimization problem

$$x, y = \underset{x,y}{\operatorname{argmin}} \sum_{i=2}^{N} \left\| I_1^2 \left((x - x_1)^2 + (y - y_1)^2 \right) - I_i^2 \left((x - x_i)^2 + (y - y_i)^2 \right) \right\| \tag{5.4}$$

where N is the number of sensor nodes that detected the signal. A detailed derivation of this equation is given in [54]. Using the L2-norm, in the above equation, gradient descent can be used to determine the location of the source; the mean location of the sensor nodes that detected the signal is a reasonable choice for the initial guess of (x, y).

5.3.3 Texture Identification

In addition to determining the source location, identifying the texture of the material used to make contact is also desired. Several machine learning approaches are suitable for this task, though models with a minimal number of parameters and operations is ideal, given the constraint of computing resources. One very simple approach is to use logistic regression—given the transient spectrum, X, the likelihood that the spectrum was produced by texture t is given by

$$y_t(X) = \phi \left(b + \sum_{N} w_i X_i \right) \tag{5.5}$$

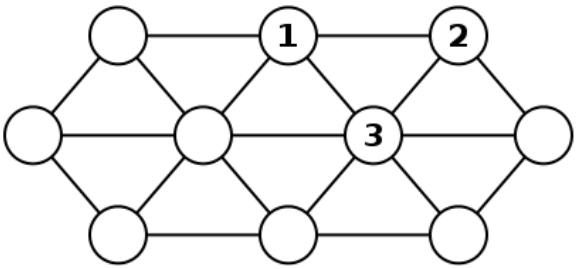
where ϕ is the sigmoid function, X_i is the value of bin i of the transient spectrum, and w_i and b are model parameters learned from a training set.

5.4 Experimental Results

5.4.1 Contact Localization

Localization was validated with an experiment involving a 15 cm x 13 cm region of the skin, shown in Figure 5.6. A vibration motor was placed on a grid with 1 cm intervals, and the intensity of the vibration signal was measured at the three labeled sensor nodes in the figure. Figure 5.7 shows the intensity of the peak bin of the transient spectrum as a function of source position.

Figure 5.6: Sensor nodes used for localization experiment.



The source position is estimated using Equation 5.4. Figure 5.8 shows the residual error for each measurement point. The dashed blue triangle represents the region of interest for this calculation. Outside this area, other nodes are expected to participate in the localization estimates, likely improving the estimates in these regions. The mean magnitude of the residual is 3.55 cm, with a standard deviation of 1.96 cm.

5.4.2 Texture Identification

A texture recognition experiment was also performed using 15 textures, as shown in Figure 5.9 and listed in Table 5.1. For each texture, 100 samples were collected by rubbing a sample of each texture near a sensor node, and recording the transient signal measured my the node. The texture sample was rubbed within a 3 cm region of the microphone, and the transient spectrum was sampled every 5 seconds.

Figure 5.7: Amplitude of the transient signal for nodes 1 (left), 2 (center), and 3 (right), due to signal source at various locations in the region of interest.

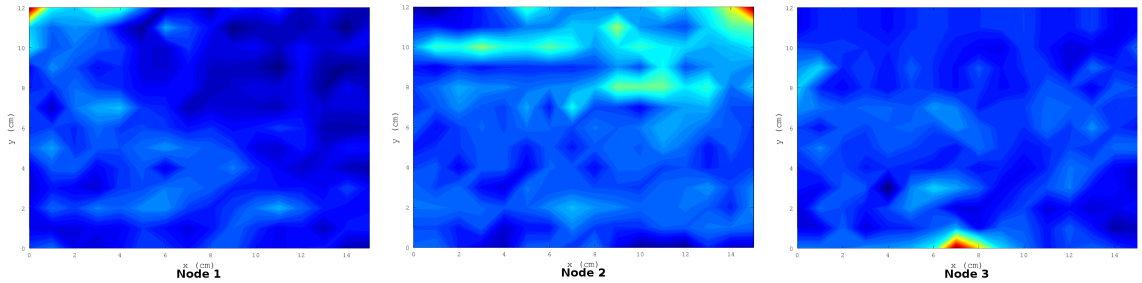
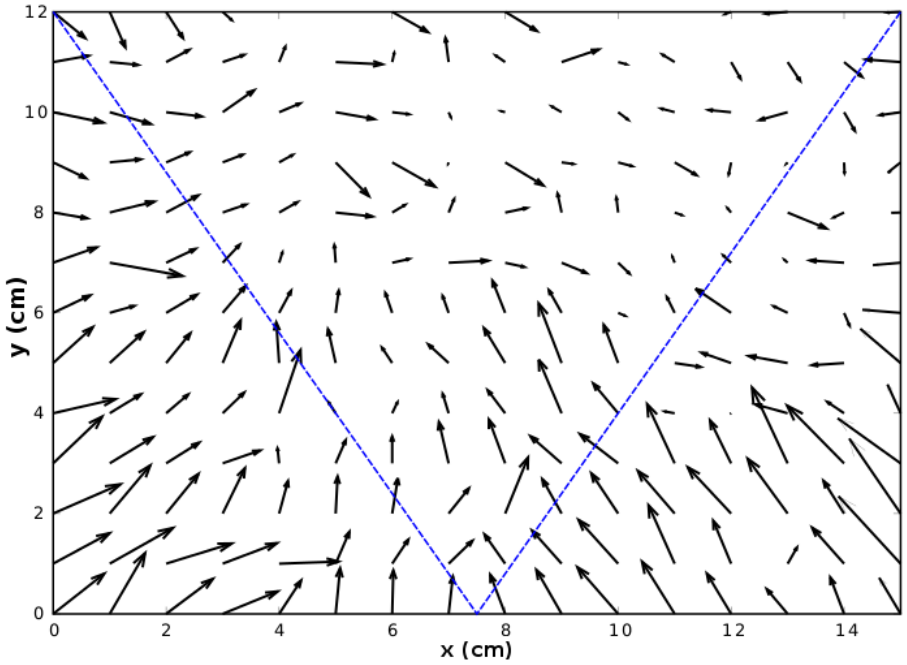


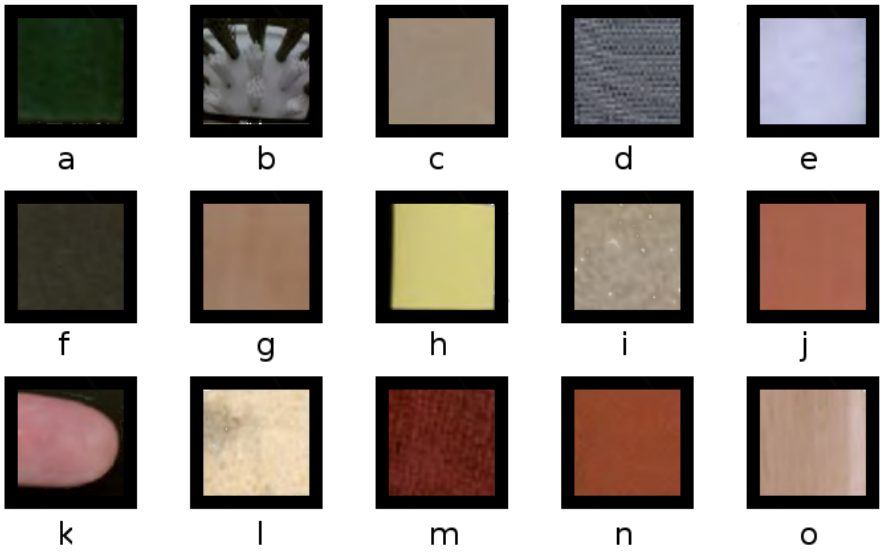
Figure 5.8: Residual error of calculated location of contact location.



A logistic regression model was trained using the collected dataset, and was assessed using 10-fold cross-validation. The logistic regression model was able to classify textures with an accuracy of 71.7%. A two-layer neural network was also trained; classification was only slightly better with a 73.1% accuracy. The confusion matrix for the logistic regression classifier is given in Table 5.2, demonstrating the classes have similar accuracy. Common confusion between cotton and dense

foam, as well as sandpaper and brush, is also observed.

Figure 5.9: Textures used for texture identification experiment.



5.5 Uncertainty Analysis

The dynamic model of the skin allows for uncertainty analysis to be performed, which provides a means of determining the effect that sensor noise has on localization predictions. Measurements from each sensor can be modeled with zero-mean Gaussian noise added; Equation 5.3 is then modified to include this noise

$$I_i(\omega, x, y) = \frac{I_0(F, \omega)}{\sqrt{(x - x_i)^2 + (y - y_i)^2}} + \mathcal{N}(0, \sigma_i) \tag{5.6}$$

where r is expressed explicitly in terms of the source and sensor node positions. Given a set

Table 5.1: List of textures used for classification experiment.

a)	Brillo Pad	b)	Brush	c)	Cardboard
d)	Coarse Wire Mesh	e)	Cotton	f)	Dense Foam
g)	Fine Wire Mesh	h)	Plastic	i)	Sandpaper
j)	Silicone Foam	k)	Skin	l)	Sponge
m)	Terry Cloth	n)	Textured Silicone	o)	Wood

Table 5.2: Confusion Matrix for the Logistic Regression classifier.

		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Actual Class	(A) Brillo Pad	**89.0**	4.0	0.0	1.0	0.0	0.0	0.0	1.0	2.0	0.0	0.0	0.0	1.0	1.0	1.0
	(B) Brush	2.0	**59.0**	4.0	6.0	0.0	3.0	0.0	1.0	9.0	2.0	1.0	3.0	6.0	4.0	0.0
	(C) Cardboard	1.0	1.0	**77.0**	0.0	2.0	2.0	5.0	2.0	1.0	1.0	4.0	3.0	0.0	0.0	1.0
	(D) Coarse Wire Mesh	3.0	7.0	3.0	**64.0**	2.0	2.0	1.0	4.0	2.0	1.0	5.0	2.0	3.0	1.0	0.0
	(E) Cotton	0.0	1.0	2.0	1.0	**69.0**	15.0	0.0	1.0	1.0	0.0	1.0	3.0	2.0	1.0	3.0
	(F) Dense Foam	2.0	3.0	3.0	2.0	14.0	**54.0**	3.0	0.0	4.0	3.0	1.0	3.0	3.0	1.0	4.0
	(G) Fine Wire Mesh	2.0	1.0	4.0	2.0	0.0	0.0	**80.0**	3.0	2.0	0.0	2.0	1.0	2.0	1.0	0.0
	(H) Plastic	1.0	3.0	1.0	0.0	3.0	0.0	2.0	**78.0**	6.0	1.0	4.0	0.0	1.0	0.0	0.0
	(I) Sandpaper	1.0	8.0	0.0	6.0	2.0	4.0	0.0	3.0	**58.0**	1.0	4.0	6.0	3.0	0.0	4.0
	(J) Silicone Foam	1.0	1.0	1.0	1.0	2.0	3.0	2.0	0.0	3.0	**77.0**	1.0	3.0	1.0	4.0	0.0
	(K) Skin	0.0	1.0	0.0	3.0	2.0	1.0	2.0	1.0	1.0	1.0	**81.0**	3.0	2.0	0.0	2.0
	(L) Sponge	0.0	3.0	5.0	5.0	4.0	3.0	1.0	1.0	0.0	2.0	3.0	**65.0**	3.0	2.0	3.0
	(M) Terry Cloth	2.0	2.0	6.0	6.0	2.0	4.0	1.0	1.0	2.0	0.0	0.0	4.0	**70.0**	2.0	3.0
	(N) Textured Silcone	2.0	2.0	4.0	4.0	0.0	5.0	1.0	1.0	0.0	5.0	0.0	3.0	3.0	**72.0**	1.0
	(O) Textured Silcone	1.0	3.0	0.0	0.0	1.0	0.0	0.0	2.0	4.0	1.0	0.0	5.0	2.0	5.0	**75.0**

of N sensor measurements, and the Jacobian matrix of the measured intensities can be calculated

$$J_I(x,y) = \begin{bmatrix} \frac{\partial I_1}{\partial x} & \frac{\partial I_1}{\partial y} \\ \vdots & \vdots \\ \frac{\partial I_N}{\partial x} & \frac{\partial I_N}{\partial y} \end{bmatrix} \tag{5.7}$$

The equations for measured intensities treats the source location as independent variables. The uncertainty in these measurements due to uncertainty of the source location is approximated using the Jacobian matrix [14]

$$\Sigma^I \approx J_I(x,y)\Sigma^{xy}J_I^T(x,y) \tag{5.8}$$

where Σ^I is the covariance matrix of the sensor measurements, and Σ^{xy} is the covariance

matrix of the source position. As sensor noise can be assumed to be independent, the covariance matrix of the sensor measurements will be diagonal. By left- and right-multiplying these equations by the pseudoinverse of the Jacobian, the covariance of the source position can be expressed as a function of sensor noise

$$\Sigma^{xy} \approx \left(J_I^T(x,y)J_I(x,y)\right)^{-1}\Sigma^I\left(J_I(x,y)J_I^T(x,y)\right)^{-1}. \tag{5.9}$$

5.5.1 Localization Uncertainty

Equation 5.9 allows for analyzing the effect of contact force, sensor spacing and contact position. Assuming a constant level of sensor noise (i.e., σ_i), the standard deviation of the position of the source (i.e., σ_x, σ_y) is calculated relative to the sensor noise. These calculates are provided in Figure 5.10 for the case where three sensor nodes are involved in computing source location.

The effect of source intensity (Figure 5.10, left) shows a power-law relationship between the source intensity and source position uncertainty. This is to be expected, given the intensity decreases inversely as a function of distance, r, and the presence of a $r^{3/2}$ term in the Jacobian.

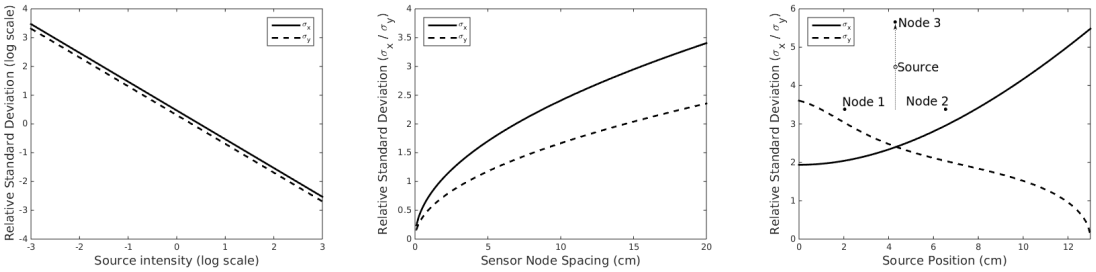
The effect of sensor spacing is given in Figure 5.10, center. This plot assumes the contact point is at the center of the triangle defined by the three sensor nodes. This relationship is useful for determining the node spacing necessary for a desired tactile acuity.

Finally, the effect of the position of the source with respect to the sensor nodes is given in Figure 5.10, right. The source is moved from the midpoint between two sensors to the third sensor, as shown in the inset of the figure.

5.6 Texture Identification

The robustness of the logistic regression classifier can also be estimated by determining the classifier accuracy with Gaussian noise introduced into the test data. Gaussian noise was introduced with a standard deviation ranging from 0% to 50% of the average energy in each bin in the test set. Ten noisy data sets were produced in this manner, and the classifier accuracy was evaluated

Figure 5.10: Left: Relative ncertainty of source position as a function of source intensity. Center: Relative uncertainty of source position as a function of sensor spacing. Right: Relative uncertainty of source position as a function of source location.



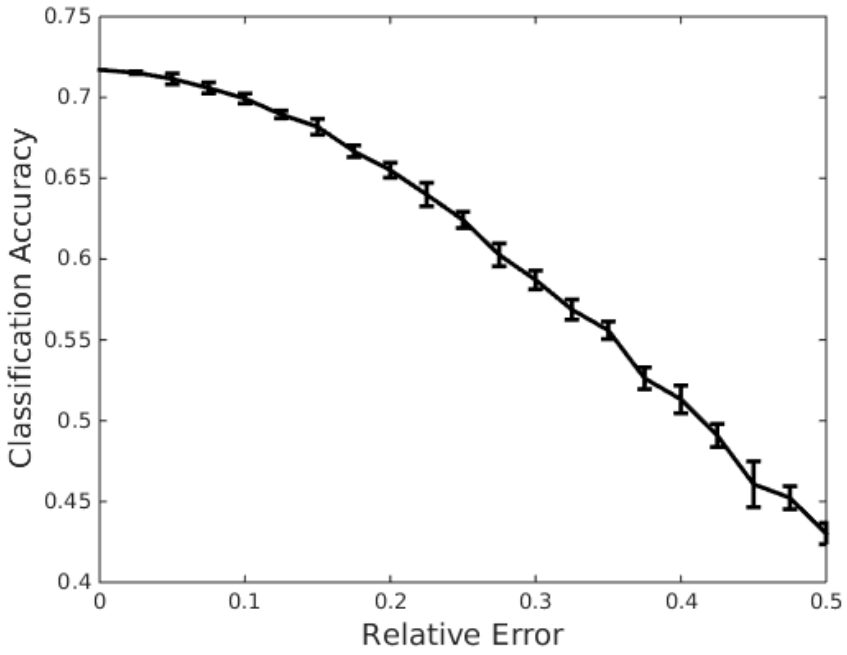
for each, shown in Figure 5.11. From Figure 5.3, the standard deviation of the measured signal reaches approximately 50% of the mean of the signal at 5–7 cm; the expected accuracy of a single node would be expected to be about 43% in the worse case for the prototype skin. However, at this point, multiple nodes would be capable of making estimates at this accuracy; it would be possible to combine predictions from several nodes to increase this accuracy.

5.7 Discussion

The skin presented in this section represents a robotic material with an amorphous algorithm designed as suggested by [91]: using a strong understanding of the dynamics of the underlying material, nodes share local state information to estimate the continuous state of the material. This approach provides several advantages, which are demonstrated in this chapter.

An accurate model of the dynamics of the material allows for a potentially simple approach to estimating the state of the material—the desired parameters of the material can be estimated by fitting the dynamics of the material to node observations. A second advantage is that uncertainty analysis can be applied to determine the robustness of the material to noise in measurements. However, there are several limitations which make this approach unsuitable to several applications. First, the dynamics of the material may not be well understood. More importantly, the physical stimuli experienced by the material may not be easily modeled. For example, modeling affective

Figure 5.11: Accuracy of logistic regression classifier with noisy data.



touch gestures is infeasible, given the complexity of gestures.

An additional limitation is signal processing requirements to model the material. In the example in this chapter, calculation of the spectrum of the signal consumes a significant portion of the available RAM (∼63%) on the microcontroller, as well as a significant portion of computing time (15 ms). This leaves few computing resources for additional processing (e.g., complex classification models).

Chapter 6

A Modular CNN-LSTM Architecture for Multi-Node Computing in Robotic Materials

This chapter describes a neural network based approach to computation in multinode robotic materials. A modular CNN-LSTM architecture is presented that extends on the CNN architectures used in Chapter 4 by adding an LSTM layer after the CNN layers, with an optional fully connected layer following the LSTM layer.

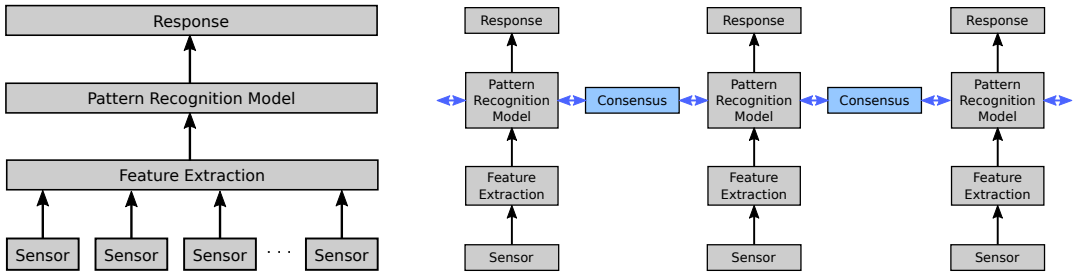
Communication between nodes is modeled using weighted connections between the output of the LSTM layer of a node and the input to the LSTM layer of a neighboring node. As with the single node CNN models, hardware limitations introduce potential constraints associated with communication—the number of values that can be communicated between nodes must remain small enough to ensure that communication bandwidth is not exceeded. At a minimum, it may be desired to limit communication between nodes to lower the amount of energy expended through communication. An approach to learning a communication protocol that limits the communication rate is also presented and validated with a simple example.

6.1 Centralized vs. Distributed Machine Learning

A typically machine learning approach assumes the complete input is available to the pattern recognition model. For an application typical of robotic materials, this would involve collecting information from *all* sensors in the material. Features would be extracted from the sensor data, which would be used as input to the pattern recognition model to generate a desired response, as

shown in Figure 6.1, left.

Figure 6.1: Comparison between centralized pattern recognition approaches and pattern recognition in sensor networks.



For robotic materials, this approach becomes infeasible for a variety of reasons. The primary hinderance is the inability for such an approach to scale. As the number of sensors increases, it becomes more infeasible to collect data from all sensors in a reasonable amount of time: a microcontroller will have a limited number of analog-digital converters to sample analog sensors, and digital sensors using two-wire communication would be limited by the bandwidth of the communication channel. Providing the desired response signals to actuators in the material is similarly hindered by communication limitations. Computing resources would also limit the number of sensors that can be reasonable processed. The number of features to adequetely summarize sensor values would need to increase to maintain a sufficient level of performance of the pattern recogntion model (unless the mutual information between sensors is exceedingly high, implying that *too many* sensors are present in the system), quickly exceeding the computing capacity of the microcontrollers used. Consequently, such an approach is suitable for robotic materials only when sensing / actuation is limited to a small set of local sensors / actuators.

An alternative approach, which has been proposed and explored by the sensor network community, is shown in Figure 6.1. In this approach, features are extracted from *local* sensor measurements, which is then used by a local pattern recognition model to produce the desired response. Performing machine learning in this manner was first proposed as a unified approach to sensor netowrk applicaitons in 2003 [122]. Implementations of this proposed approach has resulted in ap-

proaches using distributed Support Vector Machines, mixture models, and probabilistic graphical models. Support vector machines can leverage kernels which only have local spatial support to perform regression [43, 71, 72], and sharing kernel weights with neighboring nodes. Training of SVM models has also been performed in a decentralized manner by merging points describing local convex hulls of the training data [43, 73, 33]. Mixture models can be generated from sensor data in a distributed manner through distributed expectation maximization algorithms, which rely on gossip or consesus protocols to share sufficient statistics of local data [97], or to update local estimates of global model parameters [77, 42, 35]. Finally, inference in Bayesian models is performed using a belief propagation algorithm to perform global inference while requiring individual nodes to maintain only a few local random variables [18, 103, 102].

6.2 Neural Network Approach for Robotic Materials

The distributed machine learning approaches described above have been shown to be useful for several sensor network applications. However, these approach are lack properties which are desirable for robotic material applications:

- The sensor network applications cited focus on determining the distribution of some low-dimensional, slow-changing, spatially distributed environmental property: temperature, population density, distribution of a pollutant, etc. Robotic materials, on the other hand, will need to respond to stimuli that are high-bandwidth, spatially distributed, and likely to be rapidly changing: responding to human gait in wearables, affective gesture recognition, morphing shape or stiffness to adjust to dynamic loads, etc. Robotic material models will often require a temporal model to continually monitor sensor signals and generate responses.

- As sensor data in robotic materials are typically high-bandwidth, it is necessary to extract features prior to performing pattern recogntion. While commonly used classes of features exist (e.g., statistical moments, autoregressive coefficients, etc.), it would be ideal to gen-

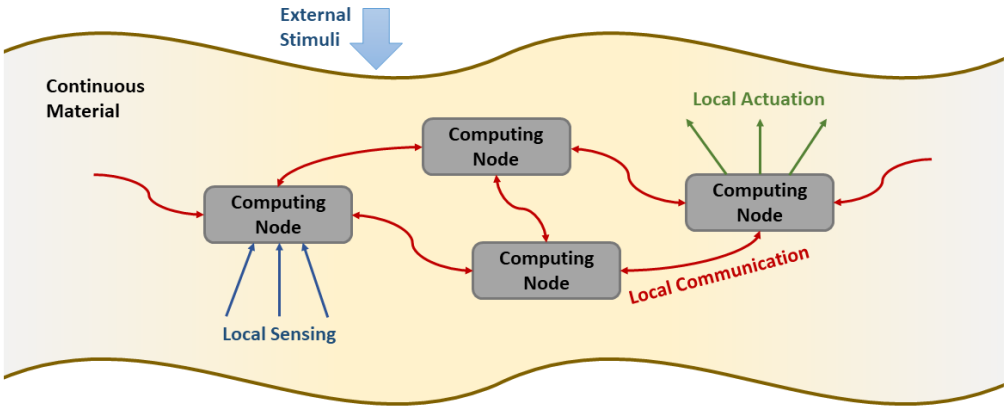
erate a minimal set of features specific for the expected sensor data, in order to minimize memory consumption. Additionally, the feature set generated should require a minimal amount of computation to perform.

- A pattern recognition model should be able to learn an arbitrary desired response. For instance, a logistic regression model, while simple, is unable to learn the XOR function [142].

6.2.1 Modular CNN-LSTM Model

Robotic materials tightly integrate a network discrete elements with local sensing, computing and acutaiton capablilities with an underlying engineered material [90]. Each computing node sensing local changes in the material due to external stimuli, and can locally modify some property of the material with local actuation. Communication can occur between neighboring nodes in order to allow for implementation of a distributed controller to generate desired global behavior. Each of these aspects is shown in Figure 6.2.

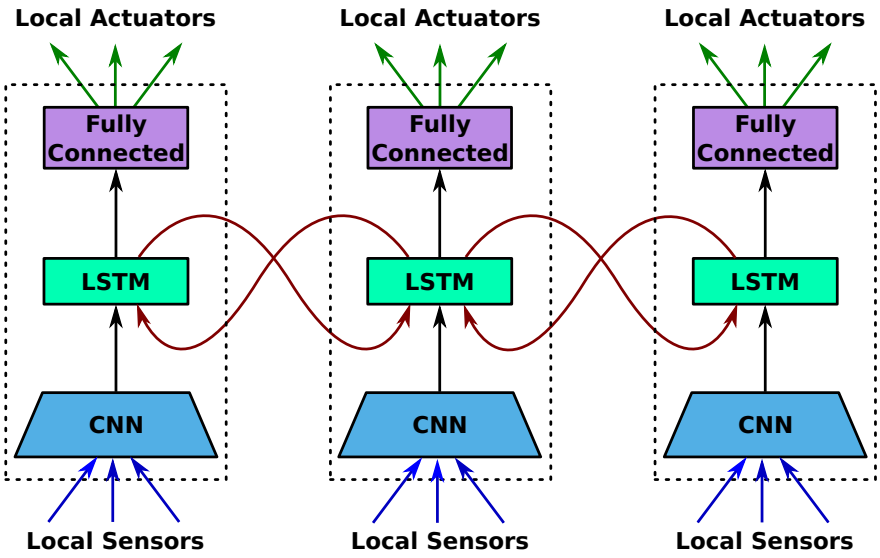
Figure 6.2: Robotic material, consisting of discrete elements with local sensing, computing, and actuation capabilities; a communication network between computing elements; and a continuous material. Reproduced from [90].



Implementing neural network models on the computing nodes needs to address the sensing, communication and actuation capabilities in a robotic material. The proposed modular CNN-

LSTM architecture is shown in Figure 6.3. The modules defined by the dashed lines are intended to be implemented on individual computing nodes in Figure 6.2. Each network module consists of a multi-layer CNN, an LSTM, and an optional fully connected layer. Local sensing is used as input to the CNN layer, and actuation is controlled by the output of the fully connected layer. The LSTM layer combines the output of the CNN and the previous output of neighboring LSTMs. The structure of a single module is similar to those used in recent approaches for speech recognition [117] and human activity recogntion [100].

Figure 6.3: General approach to implementing a modular CNN-LSTM architecture into local computing nodes in a robotic material. Local communication is performed between LSTM layers.



The three components in the architecture each serve distinct purposes: the CNN layer performs feature extraction and dimensionality reduction, converting raw sensor values to a high-level abstract representation; the LSTM layer maintains a representation of local state, and updates state based on local CNN output and neighboring state; and the fully connected layer allows for nonlinear mapping of local state to actuation values. As this thesis is concerned primarily with the perception in robotic materials, actuation will include classification and regression tasks—a host system can extract predictions from the material at arbitrary points, or a central node in

a hierarchical system can transmit results to a host system through the network as a separate operation.

This architecture can be viewed in two separate ways. From a *local* perspective, individual computing nodes implement independent neural network models, and information is communicated between nodes. From a *global* perspective, the neural network as a whole can be considered as a single model partitioned across several computing nodes. The former perspective is similar to that of distributed robotics or sensor networks, while the latter is similar to a distributed deep network, such as in [24].

6.3 Communication Control

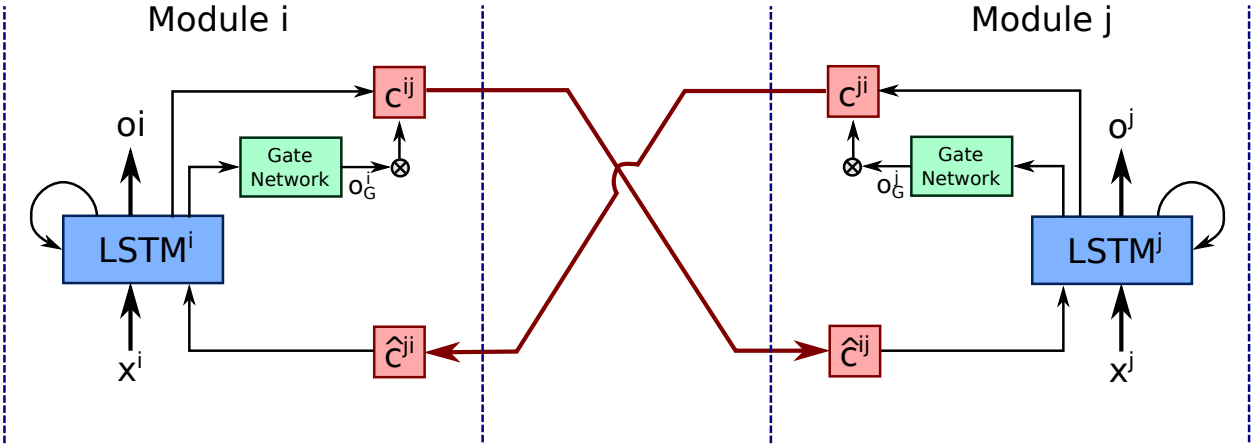
As with memory, communication between nodes in a robotic material is constrained by the bandwidth of the channel between the nodes, and limiting communication may be desirable in order to minimize the power requirements of the robotic material. As mentioned in Chapter 6, communication betweed nodes in a robotic material is modeled as connections between LSTM layers in each module in the neural network. From this perspective, parsimonious communication can be achieved in two ways: the dimensionality of the communication connection (i.e., packet size) can be minimized, or communication can occur sparsely by allowing non-zero activity in the connection a small percentage of the time (i.e., minimizing communication rate).

Several approaches to implementing communication using local neural network controllers have been explored in swarm robotics and multi-agent systems. In swarm robotics, communication has been utilized as weighted connections between nodes in a global neural network, allowing swarms to learn a global response to external stimuli [101]. Similarly, reinforcement learning has been used to train a robot swarm to perform desired tasks, given a simplified communication protocol [63]. Communication protocols can be learned directly in deep neural networks: CommNet incorporates a communication step between hidden layers in a deep neural network to simulate broadcasting of the mean state of agents at each step [127], which can be trained using backpropagation; deep reinforcement learning using recurrent neural networks has also been used to learn communication

protocols in multi-agent systems [34]. Both of these approaches assume communication can be performed at each time step.

The approach presented in this chapter assumes that information can be shared between LSTM layers in neighboring modules. Communication can be controlled using a gating network, which determines when a communication channel is active. The architecture for this approach is given in Figure 6.4.

Figure 6.4: Communication between LSTM layers controlled by a gating network.



Each module in Figure 6.4 consists of an LSTM, which receives input $x^i(t)$ from previous layers in the module's netowrk, and generates output $o^i(t)$. At each time step, a communication packet is generated, $c^{ij}(t)$, which is to be sent to module j. The communication packet can be considered a fully connected layer, whose value is given by

$$c^{ij}(t) = \phi\left(W^i_{LC} + b^i_C\right) \tag{6.1}$$

where W^i_{LC} and b^i_C are connection weights and biases, and ϕ is an arbitrary activation function. Additionally, each module contains a **Gate Network**, which generates an output $o^i_G(t)$ based on the output of the LSTM. Nominally, the output of the gate network is a scalar value of either zero or one, reflecting when the communication channel is **off** or **on**. The communication

packet is multiplied by the gate output, $\hat{c}^{ij}(t) = o^i_G(t)c^{ij}(t)$, which correlates to sending a packet when the channel is on, and sending zeros when the channel is off.

At each time step, module j incorporates the value on the communication channel at the previous time step into the input of the LSTM, that is

$$x^j_{LSTM}(t) = x^j(t) + \hat{c}^{ij}(t-1) \tag{6.2}$$

Assuming packets can be assigned continuous values, the network may largely be trained using backpropagation-through-time (BPTT). However, as the output of the gate network is nominally assigned a value of either zero or one, an error gradient doesn't exist for the output of the gate network, and so parameters from the gate network cannot be trained using BPTT. The approach to training the gate network relaxes the condition of strictly assigning a value of zero or one to the gate output by using a sigmoid activation to generate the gate output.

6.3.1 Stochastic Gating

Stochastic Gating involves training the communication network such that communication is allowed a certain percentage of the time, though specific timing of communication events is not defined. This involves using a sigmoid activation function to approximate the output of the gating unit; with sufficiently large positive or negative inputs, the value of the activation function approaches zero or one. This approach has the advantage of providing a error gradient for the gating network, allowing both the LSTM layer and the gating network to be trained concurrently using BPTT. Once trained, the sigmoid unit (i.e., "soft" gating unit) can be replaced with a threshold unit (i.e., "hard" gating unit), which takes a value of zero or one depending on the sign of the input.

Optimizing the LSTM network involves minimizing a cost function, $J(\theta)$, where θ represents the assignment of specific values to the paramters of the network. This cost function represent the error between the desired and true outputs of the LSTM network, given an input. To properly train the gating network, this cost function needs to be augmented with two cost functions associated

with the gating unit—one which encourages the gating unit to values near zero or one, and a second which controls the rate of activation (i.e., the percentage of time when the gate unit is one). The first cost function can be any function which approaches zero when the soft gating output approaches zero or one, and monotonically increases as the value approaches an intermediate value from either extreme. One such function is given as

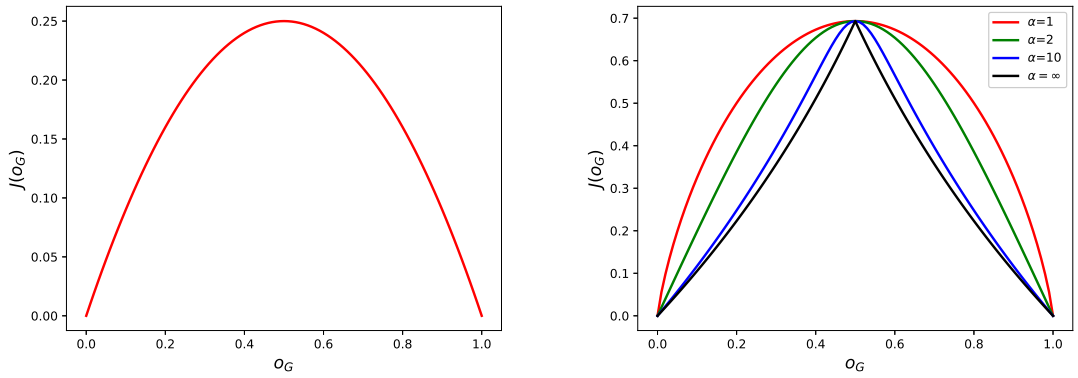
$$J_G(o_G^i|\theta_G) = o_G^i(1 - o_G^i). \tag{6.3}$$

Alternatively, considering the gate output a Bernoulli random variable, the Rényi entropy of the output [113] may also be used

$$J_G(o_G^i|\theta_G, \alpha) = H_\alpha(o_G^i) = \frac{1}{1-\alpha}\left(o_G^i\right)^\alpha \left(1 - o_G^i\right)^\alpha \tag{6.4}$$

where the order of the entropy, α, controls the slope of the cost. These two cost functions are shown in Figure 6.5.

Figure 6.5: Sparse gating cost function (left), and alternative cost function based on Rényi entropy (right) for a range of values for α.



A second cost function is required to limit the communication rate between nodes. The communication rate can be controlled by limiting the activation of each gate to be high a predefined percentage of time steps. The Kullback-Leibler (KL) divergence of the actual and desired activation

rates has been used as penalty function for training sparse autoencoders [80]; a similar penalty function is used to encourage a desired communication rate. Let ρ be the desired communication rate (i.e., percentage of time steps where a packet is sent). The true communication rate, $\hat{\rho}$, of the network is simply the percentage of time the gate is active

$$\hat{\rho} = \frac{1}{NT} \sum_{N} \sum_{t=1}^{T} o_G^i(t) \tag{6.5}$$

where N is the number of training sequences, and T is the duration of each training sequence. The activation rate cost for the gate is given by

$$J_{KL}(o_G^i|\theta_G) = KL(\rho||\hat{\rho}) = \rho log\left(\frac{\rho}{\hat{\rho}}\right) + (1-\rho) log\left(\frac{1-\rho}{1-\hat{\rho}}\right) \tag{6.6}$$

This cost function is shown in Figure 6.6. The two summands in the equation are also shown as dotted lines. While it is possible to use only the second summand as a cost function (which encourages low activation rates), this may not be ideal, as it could encourage a network to needlessly sacrifice performance for lower communication rates (i.e., allow $J(\theta)$ to be higher to allow for lower values of J_{KL}). The final cost function can then be added to the initial cost function of the network, $J(\theta)$,

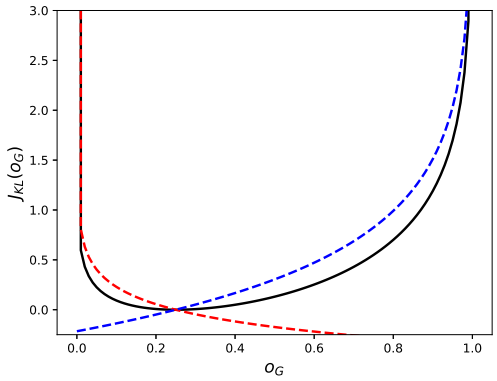
$$J_{comm}(\theta) = J(\theta) + \lambda_1 J_G(o_G^i|\theta_G) + \lambda_2 J_{KL}(o_G^i|\theta_G) \tag{6.7}$$

where λ_1 and λ_2 adjust the relative importance of encouraging gate activation to zero or one, and encouraging communication rate to match the desired rate, respectively. In practice, λ_1 and λ_2 can be annealed over several training steps, initially taking a values of zeros and increasing to the final desired values.

6.3.2 Communication Experiment

To evaluate the capability of the above approache, a network consisting of two communicating LSTM modules were trained to learn a simplified, multi-step variation of the Color-Digit MNIST

Figure 6.6: Sparse gating activation rate cost function, with a target activation rate of $\hat{\rho}$ of 0.25.



Game [34]. Each module receives a digit value, $d^a \in 0 \ldots 9$, and a colour label, $c^a \in 0, 1$. The modules must learn a two bit output: the parity of the sum of the two digits (i.e, even or odd), and the colour value assigned to the opposite module. At each time step, each module can send a 1-bit message to the other module.

The network is designed to allow each module to send a single value to the other module at each time step. The network is trained over ten time steps, and final evaluation is performed at the last time step. Communication is further constrained by forcing the communicated value to in the final network to take a value of zero or one; effectively, a single bit of information can be communicated at each time step. The two outputs of each module are dependent on information from the other module, and is therefore not expected to perform better than chance without correctly exchanging information. Each module must communicate at least two bits of information—its colour label, and the parity of its digit. Additionally, modules must learn to preprocess portions of its input (determining the parity of its input), and combine that with received information (the parity of the other module's digit), to generate the correct output.

In order to succeed, the network must learn a communication protocol that, where each module must decide which time steps to transmit information, and which order to send each bit; each module also must learn the communication protocol of the other module.

A dataset consisting of all combinations of digits and numbers was created, consisting of a total of 400 cases. 20% of the dataset was retained for testing purposes.

To validate the sparse gating approach, a network consisting of two LSTM modules was created. The LSTM layer in each module contained 25 cells. Hidden layers of 10 cells preceeded and followed the LSTM layer—this is necessary to ensure that parity can be calculated for the input digit, and combined with the parity received by the other module. A single gated communication channel was connected between the two modules, allowing each to send a single bit of information at any time step.

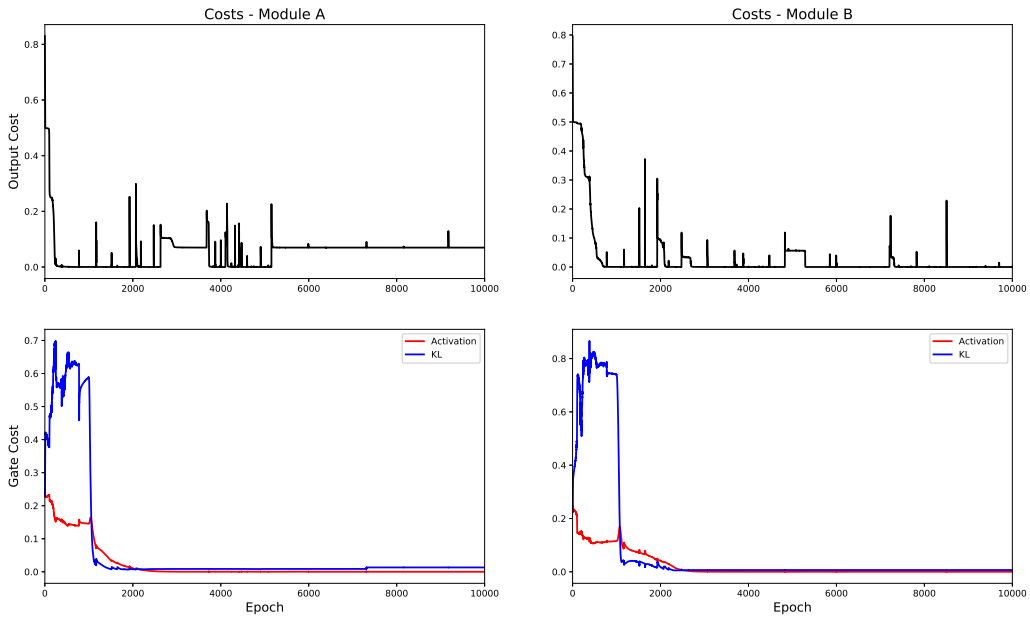
The network was trained over ten time steps, the inputs (digit and color) were available to each module at each time step, and the final output is used for evaluation. For training, the gate cost parameters were set to $\lambda_1 = 0.8$ and $\lambda_2 = 0.5$. Additionally, the communication packet value was trained to take values of one or zero in a similar manner to training the gating network. These values were set to zero for the first 1,000 training epochs, and were linearly annealled over 1,000 epochs for λ_1 and 2,000 epochs for λ_2. This ensures that network can learn the desired task in a relaxed case, after which stricter gating is learned.

Two cases were considered: one where each module was allowed to send a bit of information twice, and one where each module was only allowed to send a single bit of information. The former case is the minimal number of communication steps to succeed at the task, while the latter case will not allow both tasks to be performed successfully.

Figure 6.7 shows the training cost for each modules, as well as the activation cost and KL cost for the gate for each module, for the case where two bits are allowed to be sent by each module. The cost associated with the output decreases in only a few training epochs, demonstrating that the task can be easily learned when constraints are not placed on the communication gate. Once the gating cost is incorporated starting at 1,000 epochs, the output cost becomes unstable, jumping at several points during training as the network learns to communicate under harder gating constraints.

Figure 6.8 shows the accuracy of each module performing the parity and color tasks. Accuracy is given for both the case where the gate is allowed a soft activation (a continuum of values from

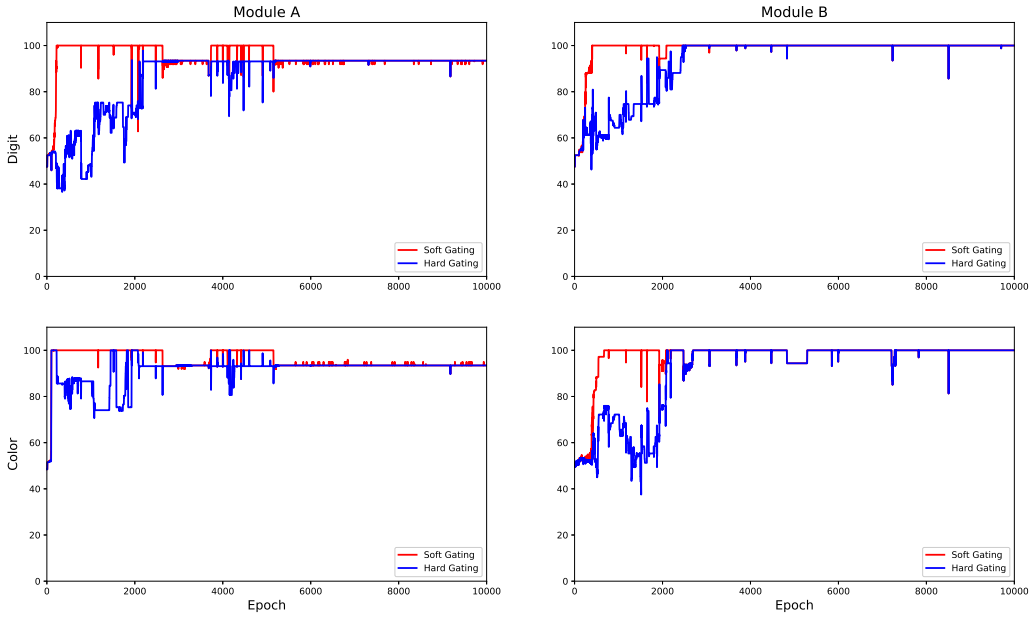
Figure 6.7: Cost function of the output of the LSTM networks (top) and cost of gating (bottom) of the two network modules as a function of training epochs; each module allowed to communicate two bits of information.



the output of the sigmoid unit), and a hard activation (assignment to either zero or one). As can be seen, tasks are readily learned for the case where soft activaiton is allowed, and the more difficult task of learning to communicate with hard activation requires several more training epochs once the easier task is learned.

Figure 6.9 shows the training cost for each module when only a single bit is allowed to be sent by each module. The training cost is slightly more unstable than for the two bit network, though it still converges to a low final cost. The accuracy of this network is given in Figure 6.10. Unlike the network capable of sending two bits of information, this network is only able to accurately perform the color task successfully with hard gating—the parity task achieves an accuracy of 50%, which is to be expected if no information regarding parity is received by a module.

Figure 6.8: Accuracy of the digit task (top) and color task (bottom) of the two network modules as a function of training epochs; each module allowed to communicate two bits of information.

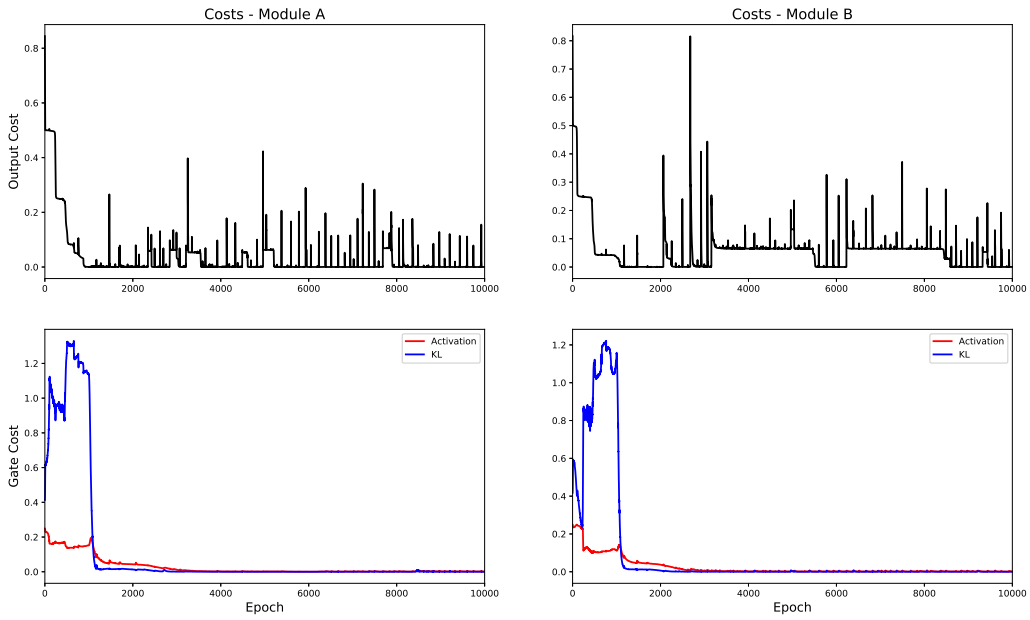


6.4 Discussion

A modular CNN-LSTM is presented as an approach to processing in robotic materials. The model builds on the CNN models presented in Chatper 3 for the single node case by incorporating an LSTM layer. This layer can be viewed as representing the internal state of the node, which can then be used to determine ancillary operations, as with the robotic skin in Chapter 4, or to determine a communication protocol with neighboring nodes.

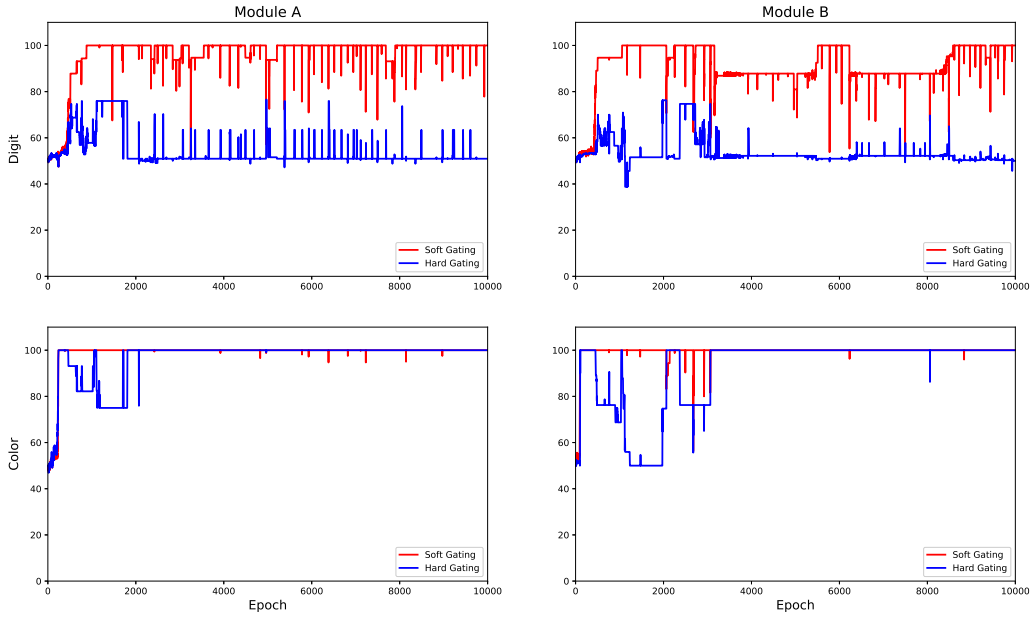
Experimental results demonstrate that an effective communication policy can be learned for the Color-Digit Game where the exact number of bits required is allowed to be shared between modules, and that reducing the number of communicated bits results in a failure of the task. For robotic materials, communication will generally consist of a vector of continuous values, rather than a single bit. The resulting training will likely be simpler, as the information communicated will not

Figure 6.9: Cost function of the output of the LSTM networks (top) and cost of gating (bottom) of the two network modules as a function of training epochs; each module allowed to communicate two bits of information.



be forced to take a binary value. As the communicated values will be continuous, there is no need to encode information in the soft assignment of the gating function, and thus the gating assignment can be pushed towards binary values without affecting the overall network performance.

Figure 6.10: Accuracy of the digit task (top) and color task (bottom) of the two network modules as a function of training epochs; each module allowed to communicate two bits of information.



Chapter 7

Multi-Node Robotic Material Experiments

This chapter presents an application of a modular CNN-LSTM in a robotic material context, namely human activity recogntion using a suite of wearable sensors located at critical points on the wearer's body. The investigation in this chapter uses the largest number of sensor channels among the applications shown so far, which highlights the utility of using modular CNN and CNN-LSTM architectures. Additionally, the communication model presented in Chapter 6 is leveraged to show the ability to limit communication bandwidth in the network without a significant drop in performance.

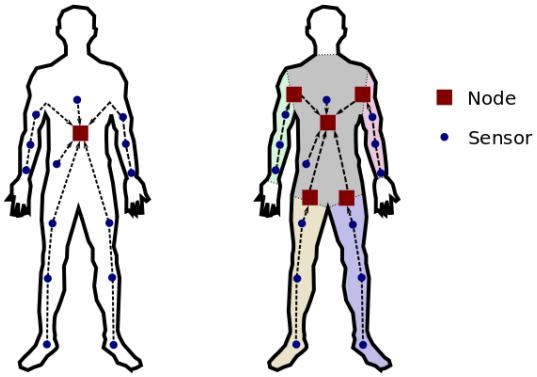
7.1 Human Activity Recognition

Human activity recognition is one of several applications which has resulted in the advancement of wearable computing and robotics [134, 106, 145, 141, 100, 55]. Human activity recognition attempts to identify various levels of activities of the wearer, such as mode of locomotion, tasks being performed, etc. As with Switchback (Chapter 4), wearable sensors and computing are unique in that no external measurement device is required, though there is significant constraints on power and weight to ensure the functionality of the system.

Figure 7.1 compared two approached to performing HAR in a wearable system. A centralize approach, such as what would be required in the references listed above, assumes all sensors values are collected and processed in a central location. As the number of sensors increases, however, communication and processing requirements places a burden on power requirements, ultimately

making such an approach unfeasible for extended use. An alternative approach is to process data in a hierarchical manner, processing sensors at each extremity, and aggregating the results at a central location.

Figure 7.1: Approaches to human activity recognition. Left: Data from a number of sensors are aggregated and processed at a central location. Right: Data are processed hierarchically, leading to more and more abstract representations. Note that network granularity is arbitrary [55].



7.2 Opportunity Dataset

The Opportunity Activity Recognition Dataset [115] is a publically available benchmark dataset for classifying human activites at several levels, from low-level locomotion (e.g., walking) to complex, high-level activities (e.g., making a sandwich). Using a set of body-work sensors as well as object and ambient sensors, data was collected from four users performing six runs of daily living activities. The body-worn sensors, which are of interest in this chapter, consists of a total of 133 total sensor channels from tri-axial accelerometers and IMUs, as listed in Table 7.1. Sensors were sampled at a rate of 30 Hz.

The task of interest for this chapter is to identify which of 18 mid-level activities, listed in Table 7.2, the wearer is performing.

Two datasets were created: one consisting solely of windows of 30 samples (1 second), and a second consisting of a sequence of 60 windows; the former was used with CNN architectures,

Table 7.1: Location of sensors used in the Opportunity Dataset.

Body (19*)	Left Arm (38)	Right Arm (38)	Left Leg (16)	Right Leg (22)	
Hip Acc	Upper Arm Acc	Upper Arm Acc	Shoe IMU	Upper Knee Acc	
Back Acc	Lower Arm Acc	Lower Arm Acc		Lower Knee Acc	
Back IMU	Upper Arm IMU	Upper Arm IMU		Shoe IMU	*
	Lower Arm IMU	Lower Arm IMU			
	Wrist Acc	Wrist Acc			
	Hand Acc	Hand Acc			

Numbers in parentheses indicate the total number of attributes (sensor values) per region.

Table 7.2: Mid-level activities to be classified in the Opportunity Dataset

Open Door 1	Open Door 2	Open Drawer 1
Open Drawer 2	Open Drawer 3	Close Door 1
Close Door 2	Close Drawer 1	Close Drawer 2
Close Drawer 3	Open Fridge	Close Fridge
Open Dishwasher	Close Dishwasher	Clean Table
Drink from Cup	Toggle Switch	*Null*

while CNN-LSTM architectures were used with the latter. The datasets were split into training, validation and test sets using an 80%-10%-10% split.

7.3 Network Architectures

7.3.1 CNN Architectures

The CNN architectures were based on those found in [141], which consists of three sets of convolutional and max-pooling layers, followed by a softmax classifcation. Decentralized versions of this architecture (without the final softmax layer) were also implemented to perform local processing of the sensors located on each leg, each arm, and on the body. The output of each CNN was transmitted to the body node, which performed final classification using a softmax classifier. Details on the derivation of these architectures is given in [55].

7.3.1.1 Centralized CNNs

Two centralized CNN models were used. The first (CNN-1) used the same architecture as that in [141], namely [(I,(30,133)), (C,5,50), (P,2,2), (C,5,40), (P,2,2), (C,3,20), (P,2,2)]. A second architecture (CNN-2) reduced the number of kernels used by the first model by 50%. This reduction in the number of kernels has minimal effect on the final accuracy of the model, though further reduction greatly reduces the accuracy. The kernel and pooling size and strides remain unchanged.

7.3.1.2 Distributed CNNs

Two distributed CNN models were developed to compare with the centralized versions. The first distributed CNN model (D-CNN-1) consists of CNNs in each body region with the same architecture as the CNN-2 model. A second distributed CNN model (D-CNN-2) estimated an optimal number of kernels for each layer specific to each region. Table 7.3 summarizes the number of kernels used in each region.

Table 7.3: Number of kernels in each layer for the D-CNN-2 model.

Layer	Body	Left Arm	Right Arm	Left Leg	Right Leg
First	40	40	40	25	25
Second	25	25	20	10	10
Third	10	12	10	4	4

7.3.2 CNN-LSTM Architectures

Thee CNN-LSTM architectures were also explored, based on the CNN-2 and the D-CNN-1 architectures.

7.3.2.1 Centralized CNN-LSTM

A centralized CNN-LSTM model (CNN-LSTM-1) was implemented based on the CNN-2 model. For this model, the convolutional and pooling layers are followed by an LSTM layer with 40 units, followed by a softmax classification layer.

7.3.2.2 Distributed CNN-LSTM

A distributed CNN-LSTM model was also implemented based on the D-CNN-1 model (D-CNN-LSTM-1). For each region of the body, the convolutional and pooling layers are followed by an LSTM layer with 20 units. The output of the five LSTM layers are then concatenated and used as input to the softmax classification layer.

7.3.2.3 Distributed CNN-LSTM with Sparse Gating

The final architecture extends the distributed CNN-LSTM model with sparse gating after each LSTM layer, which is used to control the frequency that each LSTM layer communicates its output to the final classificaiton layer. The LSTM layers communicate to a central LSTM layer with 50 units. The communication gate are controlled by the gating network described in Chapter 6.

7.4 Experimental Results

7.4.1 CNN Architectures

The classification accuracy and F1 score of the test set was calculated for each subject, and is summarized in Table 7.4. While the monolithic CNNs provide the highest level of accuracy, the performance of the D-CNNs is comparable to the monolithic CNNs.

The communication and memory requirements for each architecture was also determined, and is summarized in Table 7.5. Comparing the accuracy and memory requirements, it can be seen that there is a minor decrease in accuracy of the D-CNNs ($\sim$2% compared to the CNN-2 architecutre), while the average amount of memory required for each computing node is greatly

Table 7.4: Accuracy and F1 score for each of the CNN architectures

	CNN-1	CNN-2	D-CNN-1	D-CNN-2
Subject 1				
Accuracy	96.38%	94.15%	92.71%	92.77%
F1	0.965	0.945	0.930	0.931
Subject 2				
Accuracy	96.39%	93.62%	92.09%	91.69%
F1	0.965	0.938	0.924	0.920
Subject 3				
Accuracy	95.92%	93.60%	91.86%	90.94%
F1	0.960	0.938	0.922	0.914
Subject 4				
Accuracy	97.15%	92.79%	89.59%	90.74%
F1	0.972	0.931	0.902	0.912

reduced (∼2.25–3.0x reduciton). Additionally, the required number of sensor measurement for the distributed approaches is much less than for the centralized case, allowing for much quicker sampling and reduced storage requirements.

7.4.2 CNN-LSTM Architectures

The classification accuracy and F1 score of the three CNN-LSTM approaches were calculated for each subject, and is summarized in Table 7.6. This table demonstrates that the addition of the LSTM layer improves results over the equivalent CNN layers, increasing overall accuracy by ∼4%.

The gated version of the distributed CNN-LSTM model was trained with a target communication rate of 25%. While the reduction in communication does result in a degredation of accuracy, ranging from ∼1%–2.5%, the gated D-CNN-LSTM architecture still outperforms both the CNN-2 and D-CNN models.

7.4.3 Hardware Implementation

To analyze timing requirements and ensure the resulting distibuted models can perform correctly when implemented on a set of wearable sensor nodes, the D-CNN-1 model was implemented

Table 7.5: Memory and communication requirements for models considered

	Model	Number of Sensors	Number of Parameters	Communication Size
	CNN-1	133	46,138	0
	CNN-2	133	19,978	0
D-CNN-1	Body	19	6,448	40*
	Left Arm	38	8,103	10
	Right Arm	38	8,103	10
	Left Leg	16	5,353	10
	Right Leg	22	6,103	10
	Total	133	34,110	40
	Average	26.6	6,822	–
D-CNN-2	Body	19	10,363	30*
	Left Arm	38	13,811	12
	Right Arm	38	12,468	10
	Left Leg	16	3,499	4
	Right Leg	22	4,249	4
	Total	133	44,390	30
	Average	26.6	8,878	–

* Received from arm and leg nodes.

Table 7.6: Accuracy and F1 score for each of the CNN-LSTM architectures.

	Centralized CNN-LSTM	Distributed CNN-LSTM	Gated Distributed CNN-LSTM
Subject 1			
Accuracy	97.01%	96.05%	94.57%
F1	0.971	0.961	0.946
Subject 2			
Accuracy	96.34%	96.32%	95.25%
F1	0.965	0.965	0.954
Subject 3			
Accuracy	97.58%	97.06%	94.54%
F1	0.976	0.971	0.948
Subject 4			
Accuracy	97.78%	95.64%	94.77%
F1	0.978	0.959	0.949

on five Intel Edison modules. Neural network calcuations were performed using Python and Numpy. Sensor measurements were simulating by reading data from Run 1 of Subject 1 locally, with no

frame overlap. Communication was performed over on-board WiFi using TCP.

The calculation time for the three convolutional / max-pooling layers of each region was measured for each frame. The mean computing time ranged from 177.48–181.44 ms, with a standard deviation of 0.119 – 1.28 ms. Final classification with the softmax classifier required a mean computing time of 0.550 ms, with a standard deviation of 15.92 μs. Communication consisted of sending 80 byte (20 32-bit floats) to the body node. The mean time to communicate with the four arm and leg nodes was 62.17 ms, with a standard deviation of 36.15 ms. Given these values, a window of measurements could be processed in under 250 ms, resulting in an activity classification rate of approximately 4 Hz, which is sufficiently fast to perform real-time, on-line classification. Additionally, the communication time may be reduced by using UDP, or by using wired communication (e.g., USART or I^2C).

7.5 Discussion

The experiment discussed in this chapter demonstrate the suitability of using a modular CNN-LSTM approach for processing in robotic materials. CNN and CNN-LSTM models have produced state-of-the-art results in human activity recogntion, however, the number of sensing elements distributed on the wearer, as well as the need for a powerful processor (e.g., GPU on a smartphone), limits the applicability of using these models in a centralized approach. The human activity recognition experiment in this chapter demonstrates that a modular approach performed nearly as well as a centralized approach, but can be implemented on less powerful computing modules that are specifically designed for wearable applications.

Chapter 8

Conclusion

Robotic materials envisions a suite of novel functional materials which tightly integrates sensing, computing and actuation within the material to allow the behavior of the material to be defined algorithmically. Approaches to the computing aspect of robotic materials has drawn inspiration from a variety of fields, including swarm robotics, amorphous computing, and sensor networks. This thesis considers the recent advances in neural networks (i.e., deep learning) as a promising approach to performing in-material processing of high-bandwidth sensing signals present in robotic materials, and demonstrates the convolutional neural networks (CNN) and modular CNN-LSTM architectures can learn to perform a desired computation in a robotic material, based on sensor input. Chapter 6 presents a modular neural network architecture which allows for significant local processing of sensor measurements using a convolutional network, uses a recurrent layer to maintain local state information as well as communicate with neighboring nodes.

The primary limitations on the computing capabilities of robotic materials stems from the necessary use of small and inexpensive microcontrollers to control individual nodes. The memory capacity of the microcontrollers used in the projects in this thesis are generally on the order of several dozen to hundreds of kilobytes of program memory, and several to several dozen kilobytes of RAM. This places a significant limit on the size of neural network which can be implemented on the microcontroller. This limitation is adressed in Chapter 3, where an approach to evolving CNN architectures is presented. This approach uses a multiobjective evolutionary algorithm to jointly minimize the loss function of the networks as well as minimize the number of network parameters

in the network. I demonstrate the utility of this approach by evolving networks to recognize handwritten digits and perform terrain identification from smart tire measurements. The former experiment produced architectures which either were more accurate or required fewer parameters than similar simple CNN models. Similarly, high-quality CNN models were evolved to perform classification on terrain data, resulting in a set of efficient architectures a designer can select from based on hardware constraints or desired performance. This latter model was used to perform classification in the terrain-sensitive tire in Chapter 4.

A secondary consideration is the desire to limit communication between nodes. This can either be to ensure the communication bandwidth in the network is not exceeded, or to simply limit power consumption due to communication between nodes. In Chapter 6, the modular CNN-LSTM architecture is modified by incorporating gating networks to control when communication occurs, using the current state of the LSTM layer as input. This architecture is capable of learning a communication protocol with a limited communication rate. This is shown by an example task requiring sending two bits of information from a node to a neighboring node. In practice, communication in a robotic material would likely involve continuous values, rather than discrete bits, which are more easily trained using backpropagation. The communication model was implemented in Chapter 7, which demonstrated that reducing the communication rate to 25% of the maximum rate results in a minimal reduction of overall activity recognition accuracy of ∼1%–2.5%.

While performing classification or regression tasks in-material is an important aspect of this thesis, such tasks will often need to be considered from a system-level perspective. As noted by the amorphous algorithm in Chapter 5, one state that a node is in involves determining when a perceived signal is simply due to ambient conditions, or if further processing is warrented. An analog approach to this using machine learning techniques is given in Chapter 4 for the robotic skin experiment: the **Approach** state uses the features extracted from individual frames to determine if an obstacle should be simply avoided, or if further processing in the form of gesture recogntion is warrented.

Finally, this thesis demonstrate the suitability of CNN and modular CNN-LSTM architectures

for processing of sensor measurements in robotic materials for a variety of domains. Two examples are given for wearable computing applications: processing of an input touch gesture is performed in Chapter 4, and human activity recognition is performed using multiple sensing / computing nodes in Chapter 7. The benefit to a robotic material approach in this domain as opposed to centralized processing is primarily due to the reduction in power and weight. While it is not infeasible to sample and process the touch input measured in Chapter 4 using a more powerful device (e.g., smartphone), using a small microcontroller to perform gesture classification requires little power (e.g., up to $\sim$ 5–60mW for an ATtiny85) when compare to smartphones (e.g., several hundred mW in idle state [10]). Similarly, the approach used in Chapter 7 nominally utilizes five Intel Edison modules, whereas the network in [100] requires the GPU capabilities of a smartphone at minimum. Experiments with robotic skin, specifically for affective touch recogntion, represents the second domain where this approach has been applied, and smart tires for autonomous vehicles represents a third domain. In these two domains, the ability for robotic materials to perform in-material processing removes a significant amount of sensor query and processing for the host robot or vehicle, and allows the specific material to be treated as a loosely coupled modular component, rather than a tightly-integrated component of the robot or vehicle.

8.1 Future Work

Several avenues are available for future work, both for the modular CNN and CNN-LSTM network architectures developed and the various applications demonstrating this approach. A primary limitation to the work presented here is that the networks were trained off-line using data collected by the material. Ideally, training should be performed in-material. A na ive approach involving backpropagating error signals between nodes in the material would be possible, but could suffer from slow training times due to the high communication costs and need to synchronize training throughout the material. A more suitable approach would minimize communication of error signals between nodes, opting for a majority of training to be performed by a single node. Such an approach would have the added benefit of allowing a node to learn when not actively

processing sensor measurements, such as during the **Idle** states in the skins in Chapters 4 or 5.

In this thesis, communication delay between nodes is assumed be much smaller than the duration of one time step in the LSTM layers. Thus, it is assumed that the appropriate communication packets are available at each update. For faster update rates, communication packets may not be received when needed, resulting in LSTM layers possibly incorporating stale data. The effects of such delays on the performance of the material should be explored, as well as potential approaches to mitigate any negative effects. For example, potential errors could be reduced by incorporating random delays in packet communication during training.

The examples provided in this thesis involved learning classification or regression tasks for the material to perform. A more interesting set of tasks, which would incorporate the actuation elements of a robotic material, would be to learn distributed control policies for the material. One simple approach would be to learn a distributed policy given some global algorithm as an oracle, and to simply use the global policy to generate training examples. A more complex approach would be to perform reinforcement learning in the material, though the memory limitations of individual nodes would necessitate algorithms which do not require storing experiences (e.g., Deep Q Networks and variants).

A final point of interest is the application of the modular CNN-LSTM model to swarm robotics. Recent experiments with simple distributed feed-forward networks have shown that an emergent group mind can emerge from a collection of swarm robots [101]. The model in this paper treats connections in the neural network as communication events between roobts in the swarm, which limits the number of communication steps allowed to the depth of the individual neural networks. The modular CNN-LSTM model developed in this thesis, however, explicitly defines the number of communication events allowed, resulting in a shallower (and thus more easily trained) local neural network architecture.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Harold Abelson, Don Allen, Daniel Coore, Chris Hanson, George Homsy, Thomas F Knight, Jr., Radhika Nagpal, Erik Rauch, Gerald Jay Sussman, and Ron Weiss. Amorphous computing. Communications of the ACM, 43(5):74–82, May 2000.

[3] Tugce Balli Altuglu and Kerem Altun. Recognizing touch gestures for social human-robot interaction. In Proceedings of the 2015 ACM on International Conference on Multimodal Interaction, pages 407–413. ACM, 2015.

[4] Sliman Bensmaïa and Mark Hollins. Pacinian representations of fine surface texture. Perception & Psychophysics, 67(5):842–854, 2005.

[5] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13(Feb):281–305, 2012.

[6] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In Advances in Neural Information Processing Systems, pages 2546–2554, 2011.

[7] Andrew A. Berlin and Kaigham J. Gabriel. Distributed mems: New challenges for computation. IEEE Computational Science and Engineering, 4(1):12–16, 1997.

[8] Xi Laura Cang, Paul Bucci, Andrew Strang, Jeff Allen, Karon MacLean, and HY Liu. Different strokes and different folks: Economical dynamic surface sensing and affect-related touch recognition. In International Conference on Multimodal Interaction, pages 147–154. ACM, 2015.

[9] Giorgio Cannata, Marco Maggiali, Giorgio Metta, and Giulio Sandini. An embedded artificial skin for humanoid robots. In IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, pages 434–438. IEEE, 2008.

[10] Aaron Carroll, Gernot Heiser, et al. An analysis of power consumption in a smartphone. In USENIX Annual Technical Conference, volume 14, pages 21–21. Boston, MA, 2010.

[11] Jia-Ren Chang and Yong-Sheng Chen. Batch-normalized maxout network in network. arXiv preprint arXiv:1511.02583, 2015.

[12] Dan Ciresan, Alessandro Giusti, Luca M. Gambardella, and Jürgen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In Advances in Neural Information Processing Systems, pages 2843–2851, 2012.

[13] Andrea Cirillo, Pasquale Cirillo, Giuseppe De Maria, Ciro Natale, and Salvatore Pirozzi. A distributed tactile sensor for intuitive human-robot interfacing. Journal of Sensors, 2017, 2017.

[14] Anthony A. Clifford. Multivariate Error Analysis: a Handbook of Error Propagation and Calculation in Many-Parameter Systems. Wiley, 1973.

[15] Nikolaus Correll, Prabal Dutta, Richard Han, and Kristofer Pister. New directions: Wireless robotic materials. arXiv preprint arXiv:1708.04677, 2017.

[16] Nikolaus Correll, Nicholas Farrow, and Shang Ma. Honeycomb: a platform for computational robotic materials. In Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction, pages 419–422. ACM, 2013.

[17] Nikolaus Correll and Christoffer Heckman. Materials that make robots smart. arXiv preprint arXiv:1711.00537, 2017.

[18] Christopher Crick and Avi Pfeffer. Loopy belief propagation as a basis for communication in sensor networks. In Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence, pages 159–166. Morgan Kaufmann Publishers Inc., 2002.

[19] R. S. Dahiya, P. Mittendorfer, M. Valle, G. Cheng, and V. J. Lumelsky. Directions toward effective utilization of tactile skin: A review. IEEE Sensors Journal, 13(11):4121–4138, 2013.

[20] Christoffer Heckman Dana Hughes and Nikolaus Correll. Terrain sensitive tires for autonomous driving. In Material Robotics Workshop at Robotics: Science and Systems, 2017.

[21] Dipankar Dasgupta and Douglas R. McGregor. Designing application-specific neural networks using the structured genetic algorithm. In International Workshop on Combinations of Genetic Algorithms and Neural Networks, pages 87–96. IEEE, 1992.

[22] Omid E. David and Iddo Greental. Genetic algorithms for evolving deep neural networks. In Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation, pages 1451–1452. ACM, 2014.

[23] Richard J. De Souza and Kensall D. Wise. A very high density bulk micromachined capacitive tactile imager. In 1997 International Conference on Solid State Sensors and Actuators, volume 2, pages 1473–1476. IEEE, 1997.

[24] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In Advances in Neural Information Processing Systems, pages 1223–1231, 2012.

[25] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. A. M. T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Transactions on Evolutionary Computation, 6(2):182–197, 2002.

[26] Travis Desell. Large scale evolution of convolutional neural networks using volunteer computing. arXiv preprint arXiv:1703.05422, 2017.

[27] Emmanuel Dufourq and Bruce A. Bassett. Eden: Evolutionary deep networks for efficient machine learning. arXiv preprint arXiv:1709.09161, 2017.

[28] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285, 2016.

[29] Edmond M. DuPont, Rodney G. Roberts, Majura F. Selekwa, Carl A. Moore, and Emmanual G. Collins. Online terrain classification for mobile robots. In ASME 2005 International Mechanical Engineering Congress and Exposition, pages 1643–1648. American Society of Mechanical Engineers, 2005.

[30] Clemens Eppner, Sebastian Höfer, Rico Jonschkowski, Roberto Martin Martin, Arne Sieverling, Vincent Wall, and Oliver Brock. Lessons from the amazon picking challenge: Four aspects of building robotic systems. In Robotics: Science and Systems, 2016.

[31] Anna Flagg and Karon MacLean. Affective touch gesture recognition for a furry zoomorphic machine. In Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction, pages 25–32. ACM, 2013.

[32] Dario Floreano, Peter Dürr, and Claudio Mattiussi. Neuroevolution: from architectures to learning. Evolutionary Intelligence, 1(1):47–62, 2008.

[33] Kallirroi Flouri, Baltasar Beferull-Lozano, and Panagiotis Tsakalides. Distributed consensus algorithms for svm training in wireless sensor networks. In Signal Processing Conference, pages 1–5. IEEE, 2008.

[34] Jakob Foerster, Yannis Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In Advances in Neural Information Processing Systems, pages 2137–2145, 2016.

[35] Pedro A. Forero, Alfonso Cano, and Georgios B. Giannakis. Consensus-based distributed expectation-maximization algorithm for density estimation and classification using wireless sensor networks. In IEEE International Conference on Acoustics, Speech and Signal Processing, pages 1989–1992. IEEE, 2008.

[36] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In Competition and Cooperation in Neural Nets, pages 267–285. Springer, 1982.

[37] Yona Falinie A Gaus, Temitayo Olugbade, Asim Jan, Rui Qin, Jingxin Liu, Fan Zhang, Hongying Meng, and Nadia Bianchi-Berthouze. Social touch gesture recognition using random forest and boosting on distinct feature sets. In Proceedings of the 2015 ACM on International Conference on Multimodal Interaction, pages 399–406. ACM, 2015.

[38] Scott Gilliland, Nicholas Komor, Thad Starner, and Clint Zeagler. The textile interface swatchbook: Creating graphical user interface-like widgets with conductive embroidery. In International Symposium on Wearable Computers, pages 1–8. IEEE, 2010.

[39] Seth Copen Goldstein, Jason D. Campbell, and Todd C. Mowry. Programmable matter. Computer, 38(6):99–101, 2005.

[40] Tim Gollisch and Markus Meister. Rapid neural coding in the retina with relative spike latencies. Science, 319(5866):1108–1111, 2008.

[41] J. P. Grant, R. N. Clarke, G. T. Symm, and N. M. Spyrou. In vivo dielectric properties of human skin from 50 mhz to 2.0 ghz. Physics in Medicine & Biology, 33(5):607, 1988.

[42] Dongbing Gu. Distributed em algorithm for gaussian mixtures in sensor networks. IEEE Transactions on Neural Networks, 19(7):1154–1166, 2008.

[43] Dongbing Gu and Zongyao Wang. Distributed regression over sensor networks: An support vector machine approach. In IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 3286–3291. IEEE, 2008.

[44] Carlos Guestrin, Peter Bodik, Romain Thibaux, Mark Paskin, and Samuel Madden. Distributed regression: An efficient framework for modeling sensor network data. In International Symposium on Information Processing in Sensor Networks, pages 1–10. ACM, 2004.

[45] Jin-Seok Heo, Jong-Ha Chung, and Jung-Ju Lee. Tactile sensor arrays using fiber bragg grating sensors. Sensors and Actuators A: Physical, 126(2):312–327, 2006.

[46] Binyamin Hochner. How nervous systems evolve in relation to their embodiment: What we can learn from octopuses and other molluscs. Brain, Behavior and Evolution, 82(1):19–30, 2013.

[47] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[48] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural Computation, 9(8):1735–1780, 1997.

[49] Sarah B. Holt. Genetics of dermal ridges: The relation between total ridge-count and the variability of counts from finger to finger. Annals of Human Genetics, 22(4):323–339, 1958.

[50] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. Neural Networks, 4(2):251–257, 1991.

[51] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. Neural Networks, 2(5):359–366, 1989.

[52] David H. Hubel and Torsten N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. The Journal of Physiology, 160(1):106–154, 1962.

[53] Dana Hughes and Nikolaus Correll. A soft, amorphous skin that can sense and localize textures. In 2014 IEEE International Conference on Robotics and Automation (ICRA), pages 1844–1851. IEEE, 2014.

[54] Dana Hughes and Nikolaus Correll. Texture recognition and localization in amorphous robotic skin. Bioinspiration & Biomimetics, 10(5):055002, 2015.

[55] Dana Hughes and Nikolaus Correll. Distributed convolutional neural networks for human activity recognition in wearable robotics. In Distributed Autonomous Robotic Systems, 2016.

[56] Dana Hughes and Nikolaus Correll. Distributed machine learning in materials that couple sensing, actuation, computation and communication. arXiv preprint arXiv:1606.03508, 2016.

[57] Dana Hughes and Nikolaus Correll. In-material computation of high-bandwidth sensor signals in robotic skin. In The Robotic Sense of Touch: From Sensing to Understanding Workshop at IEEE International Conference on Robotics and Automation, 2017.

[58] Dana Hughes, Nicholas Farrow, Halley Profita, and Nikolaus Correll. Detecting and identifying tactile gestures using deep autoencoders, geometric moments and gesture level features. In Proceedings of the 2015 on International Conference on Multimodal Interaction, pages 415–422. ACM, 2015.

[59] Dana Hughes, Alon Krauthammer, and Nikolaus Correll. Recognizing social touch gestures using recurrent and convolutional neural networks. In IEEE International Conference on Robotics and Automation, pages 2315–2321. IEEE, 2017.

[60] Dana Hughes, John Lammie, and Nikolaus Correll. A robotic skin for collision avoidance and affective touch recognition. IEEE Robotics and Automation Letters, 2018.

[61] Dana Hughes, Halley Profita, and Nikolaus Correll. Switchback: an on-body rf-based gesture input device. In International Symposium on Wearable Computers, pages 63–66. ACM, 2014.

[62] Dana Hughes, Halley Profita, Sarah Radzihovsky, and Nikolaus Correll. Intelligent rf-based gesture input devices implemented using e-textiles. Sensors, 17(2):219, 2017.

[63] Maximilian Hüttenrauch, Adrian Šošić, and Gerhard Neumann. Learning complex swarm behaviors by exploiting local communication protocols with deep reinforcement learning. arXiv preprint arXiv:1709.07224, 2017.

[64] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. LION, 5:507–523, 2011.

[65] Roland S. Johansson and J. Randall Flanagan. Coding and use of tactile signals from the fingertips in object manipulation tasks. Nature Reviews Neuroscience, 10(5):345, 2009.

[66] Roland S. Johansson and A. B. Vallbo. Tactile sensibility in the human hand: Relative and absolute densities of four types of mechanoreceptive units in glabrous skin. The Journal of Physiology, 286(1):283–300, 1979.

[67] Merel M. Jung, Xi Laura Cang, Mannes Poel, and Karon E. MacLean. Touch challenge'15: Recognizing social touch gestures. In Proceedings of the 2015 ACM on International Conference on Multimodal Interaction, pages 387–390. ACM, 2015.

[68] Merel M. Jung, Ronald Poppe, Mannes Poel, and Dirk K. J. Heylen. Touching the void–introducing cost: Corpus of social touch. In Proceedings of the 16th International Conference on Multimodal Interaction, pages 120–127. ACM, 2014.

[69] Daniel Kappler, Franziska Meier, Jan Issac, Jim Mainprice, Cristina Garcia Cifuentes, Manuel Wüthrich, Vincent Berenz, Stefan Schaal, Nathan Ratliff, and Jeannette Bohg. Real-time perception meets reactive motion generation. arXiv preprint arXiv:1703.03512, 2017.

[70] Yasir Niaz Khan, Philippe Komma, and Andreas Zell. High resolution visual terrain classification for outdoor robots. In IEEE International Conference on Computer Vision Workshops, pages 1014–1021. IEEE, 2011.

[71] Woojin Kim, Jaemann Park, H. Jin Kim, and Chan Gook Park. A multi-class classification approach for target localization in wireless sensor networks. Journal of Mechanical Science and Technology, 28(1):323–329, 2014.

[72] Woojin Kim, Jaemann Park, Jaehyun Yoo, H. Jin Kim, and Chan Gook Park. Target localization using ensemble support vector regression in wireless sensor networks. IEEE Transactions on Cybernetics, 43(4):1189–1198, 2013.

[73] Woojin Kim, Miloš S. Stanković, Karl H. Johansson, and H. Jin Kim. A distributed support vector machine learning over wireless sensor networks. IEEE Transactions on Cybernetics, 45(11):2599–2611, 2015.

[74] Hiroaki Kitano. Designing neural networks using genetic algorithms with graph generation system. Complex systems, 4(4):461–476, 1990.

[75] E. S. Kolesar, C. S. Dyson, R. R. Reston, R. C. Fitch, D. G. Ford, and S. D. Nelms. Tactile integrated circuit sensor realized with a piezoelectric polymer. In IEEE International Conference on Innovative Systems in Silicon, pages 372–381. IEEE, 1996.

[76] David Kortenkamp, Reid Simmons, and Davide Brugali. Robotic systems architectures and programming. In Springer Handbook of Robotics, pages 283–306. Springer, 2016.

[77] Wojtek Kowalczyk and Nikos Vlassis. Newscast em. In Advances in Neural Information Processing Systems, pages 713–720, 2005.

[78] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems, pages 1097–1105, 2012.

[79] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In Proceedings of the International Conference on Machine Learning, pages 473–480. ACM, 2007.

[80] Quoc V. Le, Jiquan Ngiam, Adam Coates, Abhik Lahiri, Bobby Prochnow, and Andrew Y. Ng. On optimization methods for deep learning. In International Conference on Machine Learning, pages 265–272. Omnipress, 2011.

[81] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. The handbook of Brain Theory and Neural Networks, 3361(10):1995, 1995.

[82] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. Nature, 521(7553):436, 2015.

[83] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[84] Yuxi Li. Deep reinforcement learning: An overview. arXiv preprint arXiv:1701.07274, 2017.

[85] Zachary C. Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. arXiv preprint arXiv:1506.00019, 2015.

[86] Vladimir J. Lumelsky, Michael S. Shur, and Sigurd Wagner. Sensitive skin. IEEE Sensors Journal, 1(1):41–51, 2001.

[87] Markos Markou and Sameer Singh. Novelty detection: a review–part 1: Statistical approaches. Signal Processing, 83(12):2481–2497, 2003.

[88] Rainer Martin. An efficient algorithm to estimate the instantaneous snr of speech signals. In Third European Conference on Speech Communication and Technology, 1993.

[89] M. Andy McEvoy and Nikolaus Correll. Thermoplastic variable stiffness composites with embedded, networked sensing, actuation, and control. Journal of Composite Materials, 49(15):1799–1808, 2015.

[90] Michael A. McEvoy. Shape-Changing Robotic Materials Using Variable Stiffness Elements and Distributed Control. PhD thesis, University of Colorado at Boulder, 2017.

[91] Michael Andrew McEvoy and Nikolaus Correll. Materials that couple sensing, actuation, computation, and communication. Science, 347(6228):1261689, 2015.

[92] Yiğit Mengüç, Nikolaus Correll, Rebecca Kramer, and Jamie Paik. Will robots be bodies with brains or brains with bodies? Science Robotics, 2(12):eaar4527, 2017.

[93] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Dan Fink, Olivier Francon, Bala Raju, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. Evolving deep neural networks. arXiv preprint arXiv:1703.00548, 2017.

[94] Siva Natarajan and R. Russell Rhinehart. Automated stopping criteria for neural network training. In Proceedings of the American Control Conference, volume 4, pages 2409–2413. IEEE, 1997.

[95] R. B. Nelson, T. J. van Dover, S. Jin, S. Hackwood, and G. Beni. Shear-sensitive magnetoresistive robotic tactile sensor. IEEE Transactions on Magnetics, 22(5):394–396, 1986.

[96] Fernando Nobre, Michael Kasper, and Christoffer Heckman. Drift-correcting self-calibration for visual-inertial slam. In IEEE International Conference on Robotics and Automation, pages 6525–6532. IEEE, 2017.

[97] Robert D. Nowak. Distributed em algorithms for density estimation and clustering in sensor networks. IEEE Transactions on Signal Processing, 51(8):2245–2253, 2003.

[98] Masahiro Ohka, Hiroaki Kobayashi, Jumpei Takata, and Yasunaga Mitsuya. Sensing precision of an optical three-axis tactile sensor for a robotic finger. In IEEE International Symposium on Robot and Human Interactive Communication, pages 214–219. IEEE, 2006.

[99] Christian W. Omlin and C. Lee Giles. Constructing deterministic finite-state automata in recurrent neural networks. Journal of the ACM, 43(6):937–972, 1996.

[100] Francisco Javier Ordóñez and Daniel Roggen. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. Sensors, 16(1):115, 2016.

[101] Michael Otte. Collective cognition and sensing in robotic swarms via an emergent group-mind. In International Symposium on Experimental Robotics, pages 829–840. Springer, 2016.

[102] Mark Paskin, Carlos Guestrin, and Jim McFadden. A robust architecture for distributed inference in sensor networks. In International Symposium on Information Processing in Sensor Networks, page 8. IEEE Press, 2005.

[103] Mark A. Paskin and Carlos E. Guestrin. Robust probabilistic inference in distributed systems. In Proceedings of the 20th conference on Uncertainty in Artificial Intelligence, pages 436–445. AUAI Press, 2004.

[104] Radhen Patel and Nikolaus Correll. Integrated force and distance sensing for robotic manipulation using elastomer-embedded commodity proximity sensors. In Proceedings of Robotics: Science and Systems, 2016.

[105] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011.

[106] Thomas Plötz, Nils Y. Hammerla, and Patrick Olivier. Feature learning for activity recognition in ubiquitous computing. In International Joint Conference on Artificial Intelligence, volume 22, page 1729, 2011.

[107] E. Rehmi Post and Maggie Orth. Smart fabric, or "wearable clothing". In First International Symposium on Wearable Computers, pages 167–168. IEEE, 1997.

[108] Halley Profita, Nicholas Farrow, and Nikolaus Correll. Flutter: An exploration of an assistive garment using distributed sensing, computation and actuation. In Proceedings of the 9th International Conference on Tangible, Embedded, and Embodied Interaction, pages 359–362. ACM, 2015.

[109] J. Andrew Pruszynski and Roland S. Johansson. Edge-orientation processing in first-order tactile neurons. Nature Neuroscience, 17(10):1404, 2014.

[110] Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann LeCun. Efficient learning of sparse representations with an energy-based model. In Advances in Neural Information Processing Systems, pages 1137–1144, 2007.

[111] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: An astounding baseline for recognition. In IEEE Conference on Computer Vision and Pattern Recognition Workshops, pages 512–519. IEEE, 2014.

[112] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Quoc Le, and Alex Kurakin. Large-scale evolution of image classifiers. arXiv preprint arXiv:1703.01041, 2017.

[113] Alfréd Rényi. On measures of entropy and information. Technical report, Hungarian Academy of Sciences Budapest Hungary, 1961.

[114] Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In European Conference on Machine Learning, pages 317–328. Springer, 2005.

[115] Daniel Roggen, Alberto Calatroni, Mirco Rossi, Thomas Holleczek, Kilian Förster, Gerhard Tröster, Paul Lukowicz, David Bannach, Gerald Pirkl, Alois Ferscha, Jakob Doppler, Clemens Holzmann, Marc Kurz, Gerald Holl, Ricardo Chavarriaga, Hesam Sagha, Hamidreza Bayati, Marco Creatura, and José del R Millàn. Collecting complex activity datasets in highly rich networked sensor environments. In International Conference on Networked Sensing Systems, pages 233–240. IEEE, 2010.

[116] Joseph M. Romano, Kaijen Hsiao, Günter Niemeyer, Sachin Chitta, and Katherine J. Kuchenbecker. Human-inspired robotic grasp control with tactile sensing. IEEE Transactions on Robotics, 27(6):1067–1079, 2011.

[117] Tara N Sainath, Oriol Vinyals, Andrew Senior, and Haşim Sak. Convolutional, long short-term memory, fully connected deep neural networks. In IEEE International Conference on Acoustics, Speech and Signal Processing, pages 4580–4584. IEEE, 2015.

[118] S. Sankaralingam and Bhaskar Gupta. Determination of dielectric constant of fabric materials and their use as substrates for design and development of antennas for wearable applications. IEEE Transactions on Instrumentation and Measurement, 59(12):3122–3130, 2010.

[119] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt. Support vector method for novelty detection. In Advances in Neural Information Processing Systems, pages 582–588. The MIT Press, 1999.

[120] Hava T. Siegelmann and Eduardo D. Sontag. On the computational power of neural nets. Journal of Computer and System Sciences, 50(1):132–150, 1995.

[121] Patrice Y. Simard, David Steinkraus, and John C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In International Conference on Document Analysis and Recognition, volume 3, pages 958–962, 2003.

[122] Slobodan N. Simic. A learning-theory approach to sensor networks. IEEE Pervasive Computing, 2(4):44–49, 2003.

[123] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In Advances in Neural Information Processing Systems, pages 2951–2959, 2012.

[124] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. Evolutionary Computation, 10(2):99–127, 2002.

[125] W. D. Stiehl, J. Liberman, C. Breazeal, L. Basel, L. Lalla, and M. Wolf. Design of a therapeutic robotic companion for relational, affective touch. In Proceedings of the International Workshop on Robot and Human Interactive Communication, pages 408–415, 2005.

[126] Walter Dan Stiehl and Cynthia Breazeal. A sensitive skin for robotic companions featuring temperature, force, and electric field sensors. In IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 1952–1959. IEEE, 2006.

[127] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. In Advances in Neural Information Processing Systems, pages 2244–2252, 2016.

[128] Jiri Svacina. Analysis of multilayer microstrip lines by a conformal mapping method. IEEE Transactions on Microwave Theory and Techniques, 40(4):769–772, 1992.

[129] Viet-Cuong Ta, Wafa Johal, Maxime Portaz, Eric Castelli, and Dominique Vaufreydaz. The grenoble system for the social touch challenge at icmi 2015. In Proceedings of the 2015 ACM on International Conference on Multimodal Interaction, pages 391–398, 2015.

[130] Graham W. Taylor, Rob Fergus, Yann LeCun, and Christoph Bregler. Convolutional learning of spatio-temporal features. In European Conference on Computer Vision, pages 140–153. Springer, 2010.

[131] Cho-Huak Teh and Roland T. Chin. On image analysis by the methods of moments. IEEE Transactions on Pattern Analysis and Machine Intelligence, 10(4):496–513, 1988.

[132] Sreenivas Sremath Tirumala, Shahid Ali, and C Phani Ramesh. Evolving deep neural networks: A new prospect. In International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery, pages 69–74. IEEE, 2016.

[133] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In IEEE International Conference on Computer Vision, pages 4489–4497. IEEE, 2015.

[134] Pavan Turaga, Rama Chellappa, Venkatramana S Subrahmanian, and Octavian Udrea. Machine recognition of human activities: A survey. IEEE Transactions on Circuits and Systems for Video Technology, 18(11):1473–1488, 2008.

[135] Abhinav Valada, Luciano Spinello, and Wolfram Burgard. Deep feature learning for acoustics-based terrain classification. In Robotics Research, pages 21–37. Springer, 2018.

[136] Brett Warneke, Matt Last, Brian Liebowitz, and Kristofer S. J. Pister. Smart dust: Communicating with a cubic-millimeter computer. Computer, 34(1):44–51, 2001.

[137] Christian Weiss, Holger Frohlich, and Andreas Zell. Vibration-based terrain classification using support vector machines. In IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 4429–4434. IEEE, 2006.

[138] Karsten Weiss and Heinz Woern. Tactile sensor system for an anthropomorphic robotic hand. In Proceedings of IEEE International Conference on Manipulation and Grasping, pages 12–17, 2004.

[139] Darrell Whitley. A genetic algorithm tutorial. Statistics and Computing, 4(2):65–85, 1994.

[140] Keith Worden, Graeme Manson, and Nick R. J. Fieller. Damage detection using outlier analysis. Journal of Sound and Vibration, 229(3):647–667, 2000.

[141] Jianbo Yang, Minh Nhut Nguyen, Phyo Phyo San, Xiaoli Li, and Shonali Krishnaswamy. Deep convolutional neural networks on multichannel time series for human activity recognition. In International Joint Conference on Artificial Intelligence, pages 3995–4001, 2015.

[142] Zhao Yanling, Deng Bimin, and Wang Zhanrong. Analysis and study of perceptron to solve xor problem. In International Workshop on Autonomous Decentralized System, pages 168–173. IEEE, 2002.

[143] S. Yohanan and K. E. MacLean. The role of affective touch in human-robot interaction: Human intent and expectations in touching the haptic creature. International Journal of Social Robotics, 4(2):163–180, 2012.

[144] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In European Conference on Computer Vision, pages 818–833. Springer, 2014.

[145] Ming Zeng, Le T. Nguyen, Bo Yu, Ole J. Mengshoel, Jiang Zhu, Pang Wu, and Joy Zhang. Convolutional neural networks for human activity recognition using mobile sensors. In International Conference on Mobile Computing, Applications and Services, pages 197–205. IEEE, 2014.

[146] Hong Zhang and Eric So. Hybrid resistive tactile sensing. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 32(1):57–65, 2002.