5-2020

Stay-At-Home Motor Rehabilitation: Optimizing Spatiotemporal Learning on Low-Cost Capacitive Sensor Arrays

Reid Sutherland
University of Arkansas, Fayetteville

Stay-At-Home Motor Rehabilitation: Optimizing Spatiotemporal Learning on Low-Cost Capacitive Sensor Arrays

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science

by

Reid Sutherland
University of Arkansas
Bachelor of Computer Science, 2018

May 2020
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

Alexander Nelson, Ph.D.
Thesis Director

Susan Gauch, Ph.D.
Committee Member

Pat Parkerson, Ph.D.
Committee Member

ABSTRACT

Repeated, consistent, and precise gesture performance is a key part of recovery for stroke and other motor-impaired patients. Close professional supervision to these exercises is also essential to ensure proper neuromotor repair, which consumes a large amount of medical resources. Gesture recognition systems are emerging as stay-at-home solutions to this problem, but the best solutions are expensive, and the inexpensive solutions are not universal enough to tackle patient-to-patient variability. While many methods have been studied and implemented, the gesture recognition system designer does not have a strategy to effectively predict the right method to fit the needs of a patient. This thesis establishes such a strategy by outlining the strengths and weaknesses of several spatiotemporal learning architectures combined with deep learning, specifically when low-cost, low-resolution capacitive sensor arrays are used. This is done by testing the immunity and robustness of those architectures to the type of variability that is common among stroke patients, investigating select hyperparameters and their impact on the architectures' training progressions, and comparing test performance in different applications and scenarios. The models analyzed here are trained on a mixture of high-quality, healthy gestures and personalized, imperfectly performed gestures using a low-cost recognition system.

ACKNOWLEDGEMENTS

I would like to thank the following people their role in the development of this thesis and my graduate career:

First, I would like to extend my great appreciation to my research and thesis advisor, Dr. Alex Nelson, for the following: his mentorship, direction, and encouragement; the many hours spent in discussion with me and the lessons learned from them, including late-night and weekend phone calls; the knowledge that I have gained from his teaching; the many opportunities he has given me to do research, teach, and take an active role in his work at the University of Arkansas; and most importantly, the advice he has given me on the development of this thesis and the importance his work has had regarding the subjects discussed here.

Dr. Pat Parkerson and Dr. Susan Gauch, for taking time out of their schedules to participate in the committee for this thesis, read my work, and hear my defense. I would like to thank Dr. Parkerson for his punctuality, communication, and his understanding regarding the stresses of a graduate student while I was a teaching assistant for his course. I would also like to thank Dr. Gauch for giving me a chance to creatively design assignments, allowing me to take an active role in academic integrity, and for simply being a very engaging and enjoyable personality during meetings.

Haoyan Liu, for his role in the conception of this thesis and my research as a whole, the immeasurable amount of advice he has given me, the time he has spent teaching me new concepts when it was not required of him, his close collaboration on several projects inside and outside of the lab, and his patience with me throughout my time as his lab associate. I also must

acknowledge his cooperation around sharing crucial lab equipment with me while we were both forced to work from home.

Enrique Sanchez, for his role in previous work that is relative to the topics presented here, his teamwork and advice in the lab, and for sharing his work with me.

George Holmes and Jason Crawley, for providing a huge amount of IT support and general advice throughout the development of this thesis.

Lillian Glaeser, for her constant support and advice throughout the writing, editing, and overall development of this thesis, as well as my graduate career as a whole, for the time she spent checking my work for errors and providing valuable encouragement along the way, and especially for her help making graphs and other figures for this document.

Ian Spatz, for his in-depth discussions on important electrical concepts, for sharing his knowledge, and for his time spent answering late-night questions.

Jared Lambert and Jacob Fahr, for making room for the large amount of lab equipment suddenly placed in our home without complaint.

Finally, I would like to thank the subjects who participated in the experiments presented here for their time spent performing a high volume of repetitive gestures.

DEDICATION

I would like to dedicate this thesis to my parents, Pebble and David Sutherland, for their constant source of love and support they provided throughout the development of this document, as well as my college career as a whole. Without their continued financial support and their belief in me to succeed even at my lowest points, I would certainly not be where I am today. I would also like to include my siblings, Clay for being my role model to succeed from a young age and guiding me through the early years of college, and Susanna for inspiring me with her independence and drive to achieve the highest level of education.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1: Introduction

Recent research has indicated that touch-interaction is intuitive to humans; consequently, it has become one of the primary interfaces for human computer interaction (HCI). Capacitive sensing is currently emerging as one of the leading technologies used by touch-interaction systems due to its robustness, application flexibility, and ability to sense multi-touch gestures. This has resulted in the development of a wide variety of devices that use capacitive sensing to allow humans to better communicate with technology and improve their lives. Not every application can be universally applied, however, as some tasks require tailoring to specific individuals or needs. In order to create a useful HCI system, there are many factors that the designer must consider, but it can be difficult to navigate the vast complexities of modern deep learning without getting lost. This thesis examines the relationship between HCI, capacitive sensing, and deep learning through the lens of a smart rehabilitation system.

1.1 Motor Rehabilitation

Stroke is the fifth leading cause of death and the leading cause of long-term disability in the United States [1] [2]. One of the most common disabilities as a result of stroke is reduced motor skills or motor impairment. The literature is clear that motor skill repetition is an important part of the process towards recovering motor capabilities and that exercising those muscles is evidenced to largely decrease recovery time after a stroke [3], [4]. In these rehabilitation exercises, it is crucial that the gestures are monitored by a physical therapist to ensure that the gesture is performed correctly. Repeated incorrect gestures can hinder, and even somewhat reverse the rehabilitation benefits, which slows the patient's recovery process [5]. However, they must also be performed daily in order to be effective. This creates a huge demand

for therapists' valuable time and results in a large amount of travel time to and from the hospital for patients every day, which can be difficult and even harmful for a recovering stroke patient. In other situations, during a pandemic for instance, it is simply dangerous and irresponsible to continue this practice.

Gesture recognition solves this issue by eliminating the need for professional monitoring. There are many suggested methods to approach this such as monitoring EMG (electromyography) patterns [6], virtual reality-based gesture therapy [7], and other forms of motion-tracking [4] that have seen success. However, there are several issues that arise with some of these solutions. 1.) These approaches are very expensive, and some of them cannot be easily transformed into an application that a patient could use in isolation without expertise. 2.) Camera-based solutions are intrusive and are subject to issues of privacy when security is not ensured. 3.) The relevant applications serve a target demographic that is highly associated with elderly patients. Elderly people are generally willing to use new technology if there is a clear benefit; however, their biggest concern is often lack of instructions for complicated systems [8].

Therefore, a crucial goal of this thesis is to advance the efficiency and accuracy that low-cost, unobtrusive forms of gesture recognition can achieve independently and reduce the current dependency on accelerometers, depth imaging, camera tracking, and other mediums of sensory input. This will reduce application complexity and make it easier for elderly patients to adapt.

Figure 1. The Portable Transparent Activities Table (Activity board).

The ideal gesture recognition platform for rehabilitation should be easy to use, cheap, and non-intrusive. For example, an activity board (Figure 1) equipped with gesture recognition can evaluate and record a patient's exercises without the need for supervision or the presence of cameras. Such an application would be capable of recording accuracy on gestures performed daily. This tool could then provide feedback directly to the patient or create a report for a physical therapist to read and provide feedback to the patient weekly. The primary application studied in this thesis employs the use of low-cost capacitive sensor arrays to accomplish this.

1.2 Smart Touch Systems

Touch-based technology is an important part of our society and an important tool for HCI. Many modern phones and tablets utilize capacitive touch as a paradigm for this interaction which has created a universal familiarity with touchscreen systems, leading to a wave of research studying the effectiveness of touchscreen technology as an assistive or rehabilitation-oriented tool for elderly or disabled people. Many forms of home automation for the disabled that use touchscreen technology have since been implemented and tested [9], [10].

Interestingly, it has been shown that elderly persons are both better at adopting, and have a significant preference for larger touchscreen devices, such as tablets or laptop-sized screens [8]. Recent studies have capitalized on this phenomenon and applied it to the field of gesture recognition for motor rehabilitation, such as a handheld tablet that uses modern capacitive touch technology for hand gestures [11], and a multi-touch tabletop game platform equipped with frustrated total internal reflection technology [12]. This work continues the investigation into touchscreen systems by studying recognition methods for large, inexpensive, and simple capacitive sensing applications.

1.3 Deep Learning vs. Machine Learning

Many of the studies regarding motor rehabilitation or home automation discussed in this thesis use traditional machine learning algorithms such as k-nearest neighbor (k-NN), support vector machine (SVM), and different forms of clustering. While these techniques are extremely useful, they are somewhat limited in their capabilities. An application designer must have some expertise on the subject being learned in order to extract the most useful features to make the algorithms work. These algorithms can also be slow to make predictions about new data that is taken from the environment or supplied after training, particularly k-NN where inference time scales with the amount of data. The speed of inference time is important for gesture recognition so that patients are quickly notified when they are performing a gesture inadequately.

Deep learning has been studied heavily over the last several decades as an alternative to traditional machine learning algorithms with several advantages. A well-trained deep neural network can respond incredibly quickly to new inputs with accurate predictions [13]. Thanks to the development of APIs and libraries that use deep learning, it is now possible for someone to create a task-specific network capable of solving complex problems, such as gesture recognition,

without expert knowledge of the environmental parameters used for learning. However, an extremely common characteristic of deep learning is that it operates like a black box: it is given an input, then produces an output, but the observer cannot interpret much of what is happening in between. Explainable artificial intelligence (XAI) attempts to provide insight to a network's performance; however, approaches in the field are relatively new and are far from perfect [14]. This black-box behavior can be desirable because a designer does not need to manually break down the problem, since an end-to-end network can take the most basic forms of input and translate them directly to a prediction or output. The downside emerges when designing the network itself, since it is difficult to know which architecture will best suit the specific problem.

1.4 Problem

Based on its low cost, non-intrusiveness, simplicity, and design flexibility, capacitive sensing is the ideal solution to motor rehabilitation and touchscreen systems for stroke patients. A shift towards deep learning may also improve gesture recognition results for these systems, where it is applicable. However, both capacitive sensing and deep learning are relatively young fields, with much left to be perfected and studied. Under these assumptions, there is still a problem: it is unclear to a task-specific application designer which learning architectures are the most capable and the most appropriate for the desired task.

1.5 Objective

Through meta-analysis of different hyper parameters and application-specific needs, this thesis establishes a decision-tree-like method to determine the best approach to a given learning problem in the field of gesture recognition with capacitive sensing. More specifically, this thesis analyzes the performance of different learning models on several different kinds of capacitive

sensor arrays using multiple gesture recognition problems. Based on several experiments, a consistent method is identified for choosing the best model depending on the problem, the technology available, and the primary goals of the problem-solvers.

Chapter 2: Background

In order to investigate the complex and specific topics presented here, it is necessary to provide an introduction to the foundational concepts of those fields, as well as a very brief history on their development. Specifically, this thesis focuses on the understanding and careful comparison of the hyperparameters and intricate architectural differences of spatiotemporal gesture recognition. Consequently, relevant forms of neural networks and their functions will be covered in the most detail. Additionally, the experiments presented here are all performed using capacitive sensor arrays, and thus it is crucial to review concepts related to that technology.

2.1 Capacitive Sensing

Capacitive sensing has a vast list of applications in modern society and is crucial to many forms of human computer interaction. The field emerged as a solution to many problems which required non-traditional forms of input for automated systems [15], as well as extending the use of technology to people with hindering disabilities and illnesses [16]. Examples of such applications include touchscreen kiosks, buttonless keyboards [17], and perhaps most significant, the touchscreen mobile phone [18]. Some of these employ the use of surface capacitance, which is useful for determining the rough location of a touch event on a screen but has many shortcomings. The rest of what is discussed in this section is presented with projected capacitance in mind, a more complex and accurate technology.

Projected capacitive sensing fundamentally revolves around manipulating the electric field produced by a capacitor with a conductor, such as a human hand or finger, and measuring the change in capacitance experienced by the capacitor [15]. A capacitor is most commonly made up of two parallel conductive plates separated by some non-conductive medium known as

the dielectric or insulator. When voltage is supplied to the capacitor, a positive charge builds up on the positive side, and an equal and opposite charge builds up on the negative side, creating a potential difference across the plates as well as an electrostatic field (E-field) that propagates between plates. This stored charge can be discharged later, and is described in Equation 1,

$$Q = CV \tag{1}$$

where Q represents stored charge, C is the capacitance between the plates, and V is the voltage difference across them [19].

A capacitor generally acts as a rechargeable battery-like construct within a circuit that typically holds small amounts of charge but can be recharged extremely fast. However, it is useful here because the amount of charge stored in the capacitor can be measured as a way of interpreting the state of the environment it resides in. More importantly to this discussion, capacitance is also described in Equation 2,

$$C = \varepsilon_0 \varepsilon_r \frac{A}{d} \tag{2}$$

where ε_0 represents the permittivity of free space, ε_r represents the relative permittivity of the insulating material between the plates, A is the plate area, and d is the distance between plates [19]. Changing the distance between plates or introducing a conductive body that interferes with the propagation of the E-field will proportionally and measurably affect the capacitance. Combining these phenomena yields the ability to accurately sense touch, pressure, proximity, and other experimental measurements. There are two main types of projected capacitance that are discussed here: self-capacitance and mutual capacitance.

2.1.1 Self-capacitance

Self-capacitance generally refers to the measurement of an individual electrode's capacitance with respect to earth ground [20]. Figure 2 shows an example schematic that incorporates several relevant capacitances within a circuit between a microcontroller unit (MCU) and a single sensing electrode. Most of the components on the MCU are uniquely coupled with the local ground, which is in turn coupled with earth ground [19]. The most important factor is the capacitance formed between a nearby earth-grounded human finger [21] and the electrode (C_{touch}). As the foreign conductor moves closer, the distance to the electrode decreases, which increases the overall capacitance between the two plates (Equation 2), providing a measure of proximity. When a touch event occurs, the capacitance increases suddenly, due to the absence of air as a dielectric, providing a measurable indication of touch.

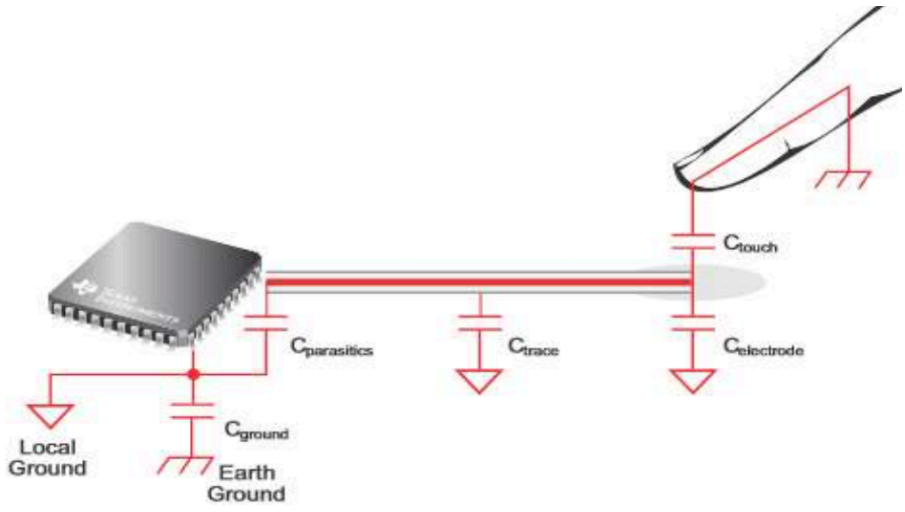

Figure 2. Visualization of the different capacitances throughout a self-capacitance circuit [19].

This form of capacitance yields stronger signals than mutual capacitance and does not inherently require multiple electrode layers. This paradigm can also be used with an array of electrodes to approximate touch within a two-dimensional space. However, with this approach,

self-capacitance is incapable of determining the exact locations of two simultaneous touches, an effect known as “ghosting” [22], and therefore cannot incorporate multi-touch gestures [19].

2.1.2 Mutual Capacitance

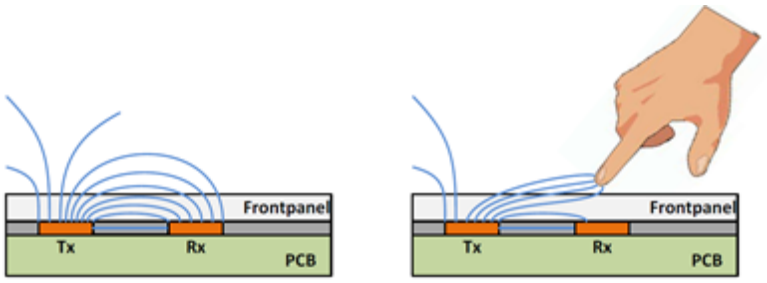

Figure 3. Disturbance of the E-field propagation between two electrodes [20].

While self-capacitance has a large potential for many applications, most of the work presented here involves mutual capacitance applications. Mutual capacitance is very similar to self-capacitance, except that a single capacitor is formed from a pair of nearby electrodes rather than an electrode with the earth ground. The positively charged electrode that emits the E-field is commonly known as the transmit electrode (Tx), while the negative electrode that receives the E-field is known as the receive electrode (Rx). As a conductor approaches an electrode pair, it disrupts the E-field formed between electrodes (Figure 3), which causes a decrease in capacitance that can be measured by the MCU. This is especially effective because humans are coupled with earth ground, which means that a touch is analogous to introducing a ground shield [20].

This approach is also very good at sensing pressure. If the Tx and Rx electrodes are stacked on top of one another, then applying pressure to the electrode area will decrease the distance between the electrodes and therefore increase the capacitance between them (Equation

2). Note that this requires either the substrate between them to be compressible, the surface of the sensor to be flexible, or some other component to be non-rigid.

2.1.3 Capacitive Sensor Arrays

One downside to mutual capacitance is that a single sensing element requires two pins on an MCU. However, like self-capacitance, if the Tx and Rx electrodes are run perpendicularly, then a two-dimensional matrix can be formed that contains many more sensing elements than the number of pins required to implement it. This configuration is known as a capacitive sensing array, or CSA. In a CSA, when a Tx electrode and an Rx electrode cross paths, they form an electrode pair, which represents a single sensing element within the array, analogous to a single pixel within a digital image. Figure 4 shows a 4x4 mutual capacitance array that has 16 sensing elements, but only requires 8 MCU pins. There are several patterns that can be used to create two-dimensional capacitive sensor arrays with mutual capacitance. When the diamond pattern is used as in Figure 4, the thin, straight bridge connecting two diamonds is where two electrodes meet and form a pair [23]. Other well-studied patterns include multi-square patterns [23], fork-shaped patterns [24], and snowflake-shaped patterns [25]. However, the applications evaluated here use only straight lines and diamond-shaped patterns.

In this thesis, mutual capacitance is the primary paradigm that is used, for three reasons. 1.) Unlike self-capacitance, mutual capacitance does not suffer from the issue of ghosting, which means that multi-touch interpretation is possible. With self-capacitance, each electrode forms an individual capacitor, such that activity is reported on an individual row or column basis. With mutual capacitance, activity is reported on an exact coordinate basis, since each sensing element is a unique capacitor that can be measured independently [22]. 2.) The E-fields between electrode pairs are restricted to a small space, which means that mutual capacitance allows for a

higher resolution and less noise between nearby elements. 3.) Because mutual capacitance does not rely on electrodes coupling directly with ground, parasitic capacitances to ground throughout the circuit are much less detrimental to sensor accuracy [20].

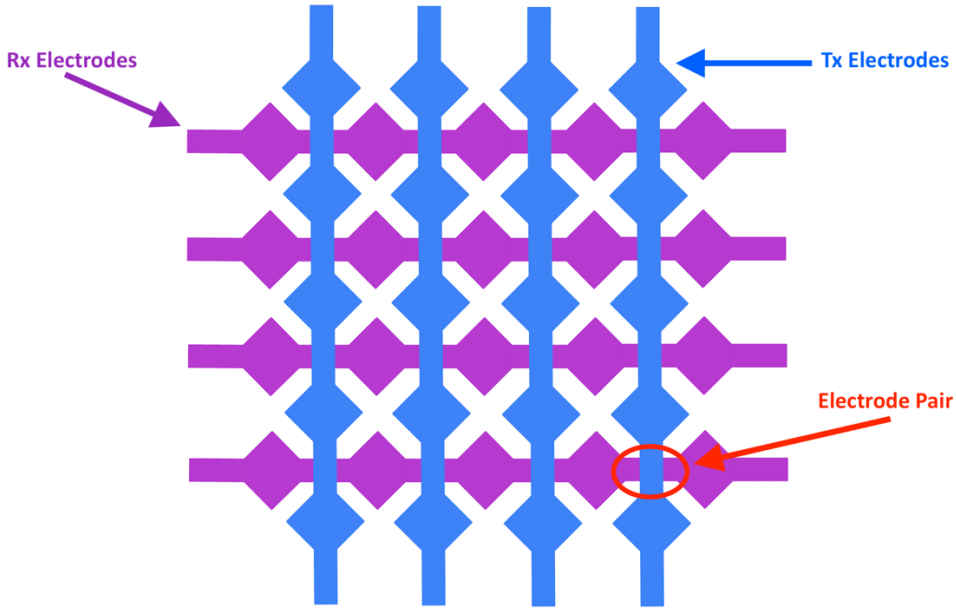

Figure 4. A 4x4 diamond-shaped pattern of electrodes with 16 electrode pairs. The bottom right-most electrode pair is denoted by the red circle.

2.2 Neural Networks and Deep Learning

In a broad sense, machine learning is heavily tied to linear regressions and Gaussian mathematics, and thus has been around for centuries [26]. The modern design of the artificial neural network (ANN), often simply referred to as a neural network (NN), was originally inspired by the anatomy of the neurons located in the mammalian brain (Figure 5) [27], and more recent works continue to compare neural networks to biological structures [28]. Often, neural networks are made up of many interconnected perceptrons that carry information across the network. The perceptron, sometimes referred to as a node within a network, can be thought of as a combinatorial processor between its inputs and its output, where the inputs are a set of signal-transmitting neurons that carry learnable weights, and the output is typically a single real value

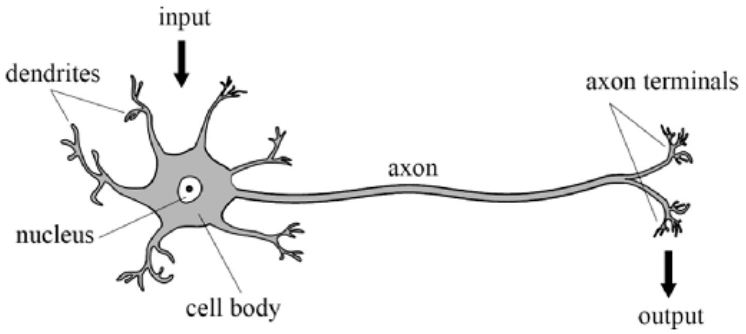

Figure 5. A neuron of the mammalian brain [29].

[30], [31]. If the linear combination of the input neurons and their respective weights is above some arbitrary threshold, then the perceptron "fires", otherwise it does not [30], [32]. The threshold can also be learned by introducing a single-value bias. Together, the weights, biases, and any other learnable values of a network are known as the network's parameters. An example perceptron is illustrated in Figure 6, and the output value of a perceptron is formalized in Equation 3, where w is a weight, x is the value of an input node, b is the bias, and n is the number of input nodes.

$$h(x) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i + b > 0 \\ 0 & \text{else} \end{cases} \tag{3}$$

$$h(x) = f\left(\sum_{i=1}^{n} w_i x_i + b \right) \tag{4}$$

In a mathematical sense, a basic perceptron with a "binary activation" (formally known as the unit step function) will output either a 0 or a 1, which mimics the behavior of biological neurons [33]. This introduces a level of nonlinearity which allows the network to make decisions based on each perceptrons' inputs and is generally referred to as the activation function. In more recent neural networks, the behavior of a perceptron is modified to better reflect Equation 4, where the function f represents the activation function for that perceptron. There are many other

activation functions besides unit step, such as Rectified Linear Unit (ReLU) [34], Sigmoid [35], and Tanh [36].

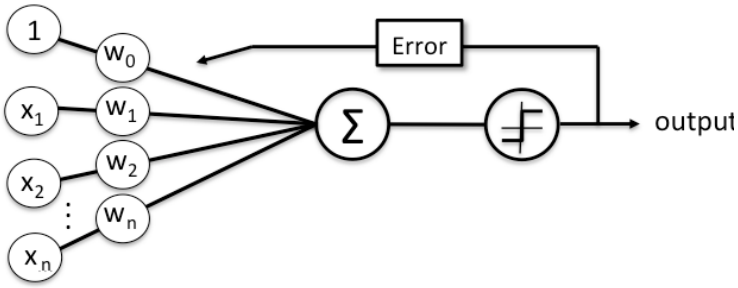

Figure 6. Schematic of a basic perceptron with a unit step activation [37]. The bias is represented by w_0.

A typical neural network consists of many sensory input nodes, collectively known as the input layer, that are all interconnected to many perceptrons that form a hidden layer. Such a structure of nodes that are fully connected to all nodes in the next layer is often called a linear layer or a fully connected layer. Additionally, the outputs of those perceptrons may act as inputs to another layer of perceptrons, where each perceptron of the previous layer is connected to every perceptron in the next layer (Figure 7). This process can be repeated an arbitrary number of times, resulting in a structure that is generally known as a multilayer perceptron (MLP). Deep learning is the study of deep networks with many layers, while networks with very few layers are described as shallow [38]. By nature, information flows in one direction from the input layer to the output layer, and thus this structure is known as a feed-forward neural network, illustrated in Figure 7 [39]. The final layer is known as the output layer; the output layer demonstrates the behavior of the network in response to the input stimuli and is typically used to come to some conclusion or prediction about the nature of the environment.

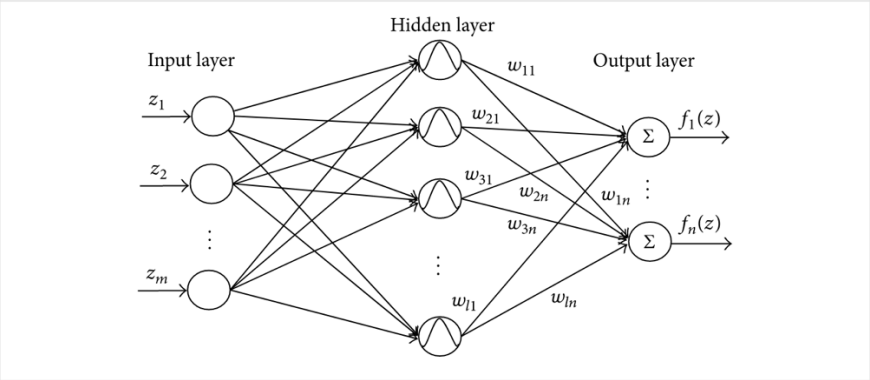

Figure 7. A feed-forward Neural Network with three layers, and activations at the hidden and output layers [29].

In order for a network to learn, it must assign credit to different weights to decide how to update them so that the network exhibits the desired behavior. Supervised learning (SL) accomplishes this by comparing the network's output to a user-defined label given to the input data [40]. In order to quantify this comparison, an error function (or loss function) is chosen and used to analyze the correctness of the network's output relative to the provided label. From there, the process of backpropagation (BP) provides a link between the final error and the weights throughout the network that contributed the most to that error, using differentiable calculus to find the gradient of the error function with respect to each individual parameter (Figure 7) [41], [42]. Gradient descent (GD) was later developed and is now widely used as an algorithm to manage the process of learning the parameters [43]. Equation 5 demonstrates GD to update the weights of a network,

$$W_k = W_{k-1} - \eta \frac{\partial E(W)}{\partial W} \tag{5}$$

where k refers to the current iteration of the algorithm, E is the error or loss function, and η is the learning rate (LR), which the designer can use to control how heavily the parameters are adjusted. Gradient descent combined with backpropagation offers a solution to the credit

assignment problem by retroactively deciding which weights to update and by how much in order to minimize the loss function and improve performance.

There are many other methods of learning (unsupervised, reinforcement, etc.) and gradient descent (Adam, RMS, etc.) that are not covered here. In this thesis, the focus is on supervised learning models, which is typical for gesture recognition. Specific algorithms used in the experiments will be established in greater detail in Chapter 5.

2.2.1 Convolutional Neural Networks

While neural networks are very powerful, they do not excel at learning spatial qualities of their inputs. The order of the inputs that a linear layer receives does not typically matter, which implies that a node's neighbors do not have any effect on that node's meaning. When the input is something with spatial quality, such as an image, this is obviously a desirable feature. The convolutional neural network (CNN) was developed in order to train neural networks to learn spatial features from their inputs. LeNet was one of the earliest CNNs that implemented gradient-based backpropagation in order to learn and was a very influential work in the field [43]. LeNet-5, shown in Figure 8, features two essential components of CNNs: convolutional layers and subsampling layers.

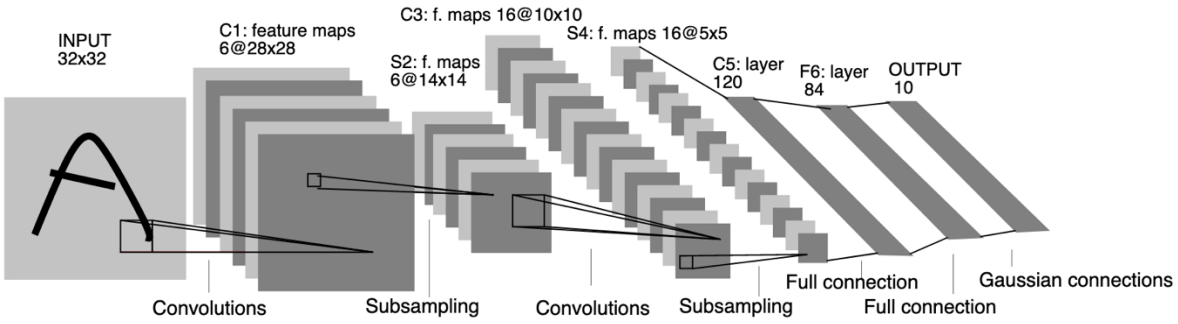

Figure 8. The architecture of LeNet-5. The architecture features multiple convolution layers, each followed by a subsampling layer, with several fully connected layers at the end [43].

Convolution can be thought of as sliding a small set of weights, collectively known as a kernel or filter, over an array of information. At each stride, the dot product between the kernel and the window of information it is currently sliding over is computed, and the resulting output value is written to the corresponding spatial location in the output image. This dot product essentially produces a larger value if the window of input information is similar to the kernel, and a smaller value if there is little correspondence between the two. Each unique kernel learns its weights independently and produces a different unique output channel, or feature map, that is dependent on the input it passes over. The input can also include a third dimension, usually referred to as the channel or depth dimension. For example, color images are typically represented with three channels, one for red, green, and blue, where each pixel value represents the intensity of one color in that specific pixel of the image.

Regardless of the number of input channels, the output dimensionality of a convolutional layer is equal to the number of kernels that it learns. When there are multiple input channels, all input channels are convolved with an individual kernel, and the resulting feature map produced by the kernel is equal to the depth-wise sum of all convolutions. This behavior is what makes a convolutional layer linear, similar to a simple linear layer, as every input channel is connected to every output channel. Finally, a convolutional operation generally reduces the spatial size of the input slightly, by about half of the kernel size. This is usually helpful for reducing the number of features learned but is not always desired. Zero-padding is a method of overcoming this spatial reduction; by increase the size of the input image to include extra zeros around the border, it is possible to eliminate any change in spatial dimensions.

When feature reduction is desired, the small reduction offered by a convolutional layer is not the most efficient. In many CNNs, a subsampling layer follows each convolutional layer.

This is also called pooling, and simply reduces the size of a feature map by evenly compressing the image along each dimension. A common example is known as max pooling, where only the maximum value from each block of pixels is retained, and the rest are discarded. After a number of convolution and subsample pairs, a CNN consists of many small feature maps that each represent high-level spatial characteristics about the original input. At this point, these features might be "unrolled" to a series of linear layers, as is done in LeNet-5, where the spatial representation no longer matters because the remaining features are essentially linked to the original spatial qualities [43]. More recent architectures such as U-Net [44] and DenseNet [45] demonstrate the impressive feats that convolutional architectures are capable of. Convolution is arguably one of the foundations of modern deep learning due to its many useful functions and is also applied to several of the architectures evaluated here.

2.2.2 Recurrent Neural Networks

While CNNs excel at learning spatial features, they generally do not have a strong ability to learn features that span a time-ordered sequence. Convolution offers a possible solution when consecutive timesteps are convolved together as input channels, but this is not the best approach. The recurrent neural network (RNN) was later introduced as a method of learning time relationships. Initially, learning this way was not very efficient, until backpropagation through time (BPTT) was introduced, a method that is still commonly used today [46]. Later, the RNN was popularized when it began making large improvements to the field of language modeling and text prediction [47].

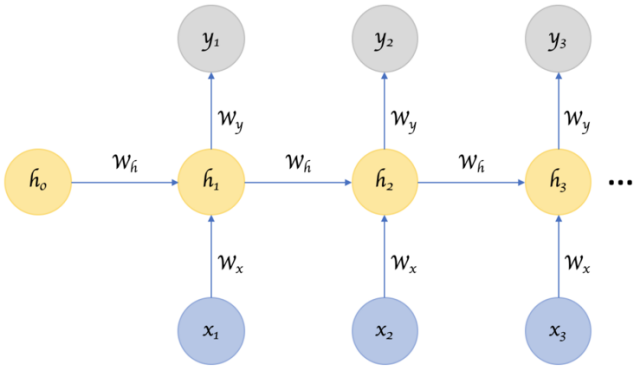

Figure 9. A simple recurrent neural network structure, where *x* is the input, *h* is a hidden node, *y* is the output, and *w* is a weight. The numbered subscripts represent the location in time [48].

A simple RNN structure is shown in Figure 9, where each input node represents the state of the input at that time and is connected to a hidden node, also called a recurrent cell. Each recurrent cell passes information forward in the time direction to the next recurrent cell and produces its own output as well, providing an output at every timestep. Similar to a linear layer, additional hidden recurrent layers can be stacked such that the output of one layer acts as the input to the next layer. Unlike a hidden node within a linear layer, a recurrent cell in an RNN often contains an array of hidden values that form the hidden dimension, which introduces a complexity that allows the network to learn complex temporal relationships. The foundational equation for computing the hidden feature vector within a typical modern RNN cell is given in Equation 6,

$$h_t = \tanh\left(W_x x_t + W_h h_{t-1} + b\right) \tag{6}$$

where x_t is the input feature vector at every timestep *t*, h_t is the current hidden state, *b* is the bias, and W_x and W_h are the weight matrices associated with the states and inputs.

When backpropagating through time, gradients often begin to "vanish" rather quickly due the large number of cells they must pass through to get back to the beginning of the time series. The long short-term memory (LSTM) cell was invented as a solution to this problem, where a

recurrent cell consists of many learnable "gates" that help control the flow of information, allowing recurrent networks to "remember" much more information when a sequence spans a large time period [49]. When LSTMs initially remembered information too well, the forget gate was added to help the LSTM learn when sequential information is not important [50]. Peephole connections were incorporated later to provide the ability to accurately learn the timing of features as well as the order [51]. An LSTM cell is visualized in Figure 10, and the general equations that an LSTM cell learns are listed in Equation 7,

$$\begin{aligned} i_t &= \sigma(W_{xi}X_t + W_{hi}h_{t-1} + b_i) \\ f_t &= \sigma(W_{xf}X_t + W_{hf}h_{t-1} + b_f) \\ g_t &= \tanh(W_{xg}X_t + W_{hg}h_{t-1} + b_g) \\ c_t &= f_t \circ c_{t-1} + i_t \circ g_t \\ o_t &= \sigma(W_{xo}X_t + W_{ho}h_{t-1} + b_o) \\ h_t &= o_t \circ \tanh(c_t) \end{aligned} \tag{7}$$

where *i*, *f*, *g*, and *o* represent the *input*, *forget*, *cell*, and *output* gates, respectively, *t* is the current time step, *W* is the weight matrix for the entire layer, *b* is the bias at each gate, *X* is the input feature vector, *h* is the hidden state vector, *c* is the cell state vector, σ is the sigmoid activation function, and $\circ$ is the Hadamard product (element-wise multiplication) [42].

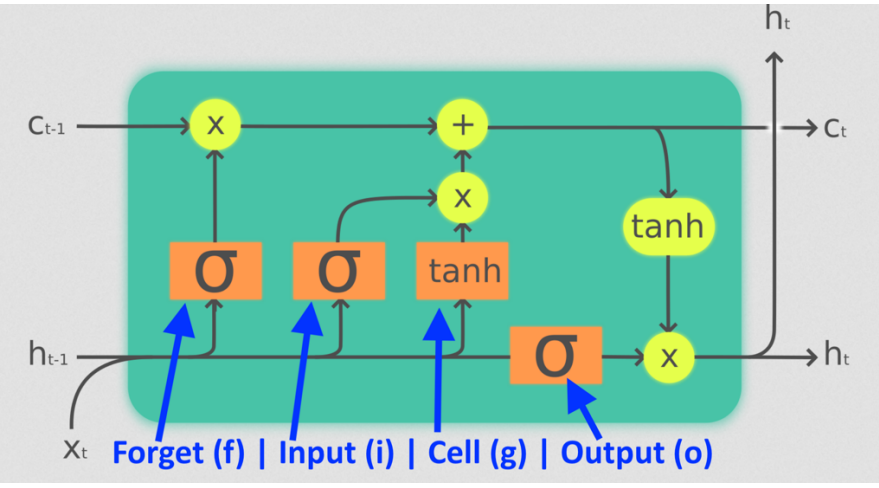

Figure 10. A standard LSTM cell (without peephole connections). The black arrows are copies where inputs are matrix multiplied by weights. Yellow circles are element-wise operations, and orange boxes are activation layers. The gates are labeled at the bottom in blue.

LSTMs have proven to be a critical tool in the field of temporal learning and sequence modeling and are used at the forefront of several architectures evaluated in this thesis. However, an LSTM network requires a rather large number of learnable parameters to work effectively. More recently the Gated Recurrent Unit (GRU) was proposed as an alternative solution to the vanishing gradient problem in RNNs with a much lower parameter space [52]. Equation 8 outlines the equations that define a GRU,

$$\begin{aligned} r_t &= \sigma(W_{xr}X_t + W_{hr}h_{t-1} + b_r) \\ z_t &= \sigma(W_{xz}X_t + W_{hz}h_{t-1} + b_z) \\ n_t &= \tanh(W_{xn}X_t + r_t \circ (W_{hn}h_{t-1} + b_n) \\ h_t &= (1 - z_t) \circ n_t + z_t \circ h_{t-1} \end{aligned} \tag{8}$$

where *r*, *z*, *n*, represent the *reset*, *update*, and *new* gates, respectively.

2.2.3 Deep Networks

The U-Net mentioned previously is an example of an extremely deep CNN [44] that creates a heavy abstraction from the original input to what is learned much farther down. In general, deep learning also includes a wide variety of networks and different forms of learning to accomplish complex and specific tasks. Some examples are networks that combine architectures such as convolutional and recurrent layers [53], [54], and networks that operate on multiple forms of input at once [55]. One major focus of this thesis is experimenting with several different deep architectures, reducing them to fit the complexity of simpler applications, and analyzing how they might be applied to achieve the same level of results.

Chapter 3: Motivation

This work is a continuation of recent research in the fields of gesture recognition and capacitive sensing, primarily for rehabilitation purposes. Prior to the work presented in this document, a series of investigations were made into other capacitive sensing applications, such as soft robotic actuators, wearable data collection sensors, and different forms of machine learning, which helped motivate the experiments presented here. Together, these studies, along with the questions they raised, helped motivate the thesis statement made here and the experiments performed to reinforce that statement.

3.1 Related Work

This section outlines the recent works mentioned in the previous paragraph. In addition, many of the techniques experimented with in this thesis are inspired by works that are outside of the field of smart motor rehabilitation, but still hold strong relevance. This thesis attempts to combine these methods and tune them to the specific task of optimizing low-resolution gesture recognition architectures that are compatible with microcontrollers. Most importantly, this work builds heavily off of the use of gesture recognition using low-cost capacitive sensing for rehabilitation purposes [56], [57].

3.1.1 Gesture Recognition

Gesture recognition is a popular topic in recent literature with many approaches to the problem including deep spatiotemporal learning. Space-time augmentation has been used to standardized multiple dimensions and avoid overfitting with 3D convolution [58]. Some works focus on using multiple components such as depth images to learn two mediums of features

simultaneously [59], [60]. 3D convolution has also been combined with recurrent structures to model large time frames as shorter clips [61], recognize actions frame by frame [55], and label entire video sequences [54].

3.1.2 Capacitive Sensor Arrays

Capacitive sensor arrays hold relevance in several fields including wearable technology, motor rehabilitation, and smart-home systems. Specifically, wearable CSA technology is a major focus of recent efforts by our lab and the individuals within it. Several works have been published by our PI detailing the use of machine learning for gesture recognition together with wearable CSAs, specifically for the benefit of patients recovering from a loss of motor functions. These works are tied very closely to the original work presented in this thesis, and thus are important to the discussion. Wearable CSAs, accelerometers, and flex-sensing gloves have been combined with machine learning to provide home automation for disabled patients [62], [63]. Additionally, several methods have been tested to tackle the issue of rotational, translational, and person-to-person variance in gesture performance [64]. More recently, it has been evidenced that the specific system, gesture set, and learning algorithm must all be personalized to the user's specific type of injury [56].

Rigid CSAs, as opposed to flexible CSAs, have also been the subject of recent gesture recognition models. Recent work by our team has shown that CSAs with convolutional LSTMs are a low-cost and highly effective solution for analyzing gestures as a part of motor rehabilitation with high-range motions [57]. Small hand-sized CSAs are also being evaluated for their use in gaming paradigms that are adaptive to those with motor or other physical disabilities [65].

3.1.3 Soft Robotics

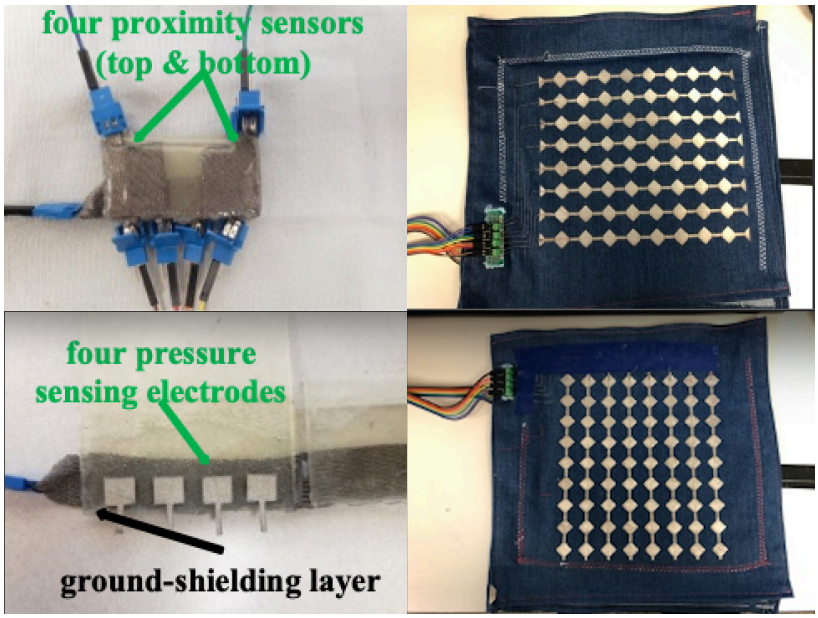

Figure 11. Flexible capacitive sensor made embedded in silicon rubber (Left). Wearable capacitive sensor array embedded in denim fabric (Right).

In recent years, the field of soft robotics has grown significantly, and the focus of many applications has been to mimic biological systems, in ways such as flexibility and environmental proprioception. One such method of achieving this is by equipping soft robotic actuators with multiple kinds of flexible capacitive sensors that work together to gain a useful proprioception [66]. Our team has performed experiments of our own as well, studying the performance of several multi-modal flexible capacitive sensor designs that contain multiple sources of feedback (Figure 11). The aim of those studies was to establish a proof of concept showing that flexible proprioception can be achieved at a very low cost with a reasonable degree of accuracy. A technical report was written outlining the proof of concept and its results, which were successful in achieving touch, proximity, and pressure, and is awaiting submission. Additionally, we collaborated with a Mechanical Engineering lab to create several prototypes of flexible, motorized soft robotics that are capable of moving, rotating, and bending in multiple dimensions.

These robotic actuators were equipped with wearable fabric sensors as well as flexible sensors integrated into the silicon material of the actuators, with the aim of establishing proprioception within a soft robotic application.

3.2 Thesis Motivation

After observing the experiments, results, and conclusions of the works in the previous section, it is clear that there are many different approaches to the problem of gesture recognition that are capable of success. However, there is a reason for this. Most of these works are very applied, meaning they are very specifically tuned to a certain task, and that fact is reflected in the machine learning algorithms used. For instance, soft robotics require different types of feedback than hard robotics, and thus require different algorithms to manage. Even within the field of capacitive sensing, there are many methods that have been proven to work on a range of different CSA applications. Regarding non-touch capacitive sensing applications, there are quite a few factors which affect the capacitance substantially, as it is a complex signal processing science with many forms and physical implementations. That is to say that there is not one single model which could work optimally for every problem.

Table 1: Breakdown of relevant CSA-based gesture recognition papers. Table 4 in the Appendix provides a list of algorithm acronyms and their full names.

First Author (Year)	Sensor Used	Recognition Algorithm / Model	Best Accuracy
Nelson (2015) [64]	Wearable CSA	HMM, DTW	97%, 99%
Singh (2016) [63]	Wearable CSA	NNC, DTC, NBC	98%
Nelson (2018) [67]	Wearable CSA and accelerometer	DTW	100%
Liu (2019) [57]	Activity board CSA	C-LSTM	100%

Table 1 shows a breakdown of several recent works in gesture recognition that use capacitive sensing, and what method they used to learn gestures. The results indicate that there is

a wide variety of approaches that can be used for solving a wide variety of capacitive gesture-based problems. While there are many different studies that have been successful, there has not been a thorough analysis that demonstrates which approaches are the most efficient for which problems, and why. From that, we came up with an important fundamental question: can it be known, or at least confidently guessed which model is the right one without implementing and testing all models considered? It has not been established whether there exists a reliable method for identifying the best approach to use for a given application before the testing phase. Such a method would be very useful, as it would provide inspiration to those attempting to solve new problems, save time during the development phase, and generally improve experimental results early on.

3.3 Goals

A series of experiments are presented here that are performed on capacitive sensing applications in order to establish method of determining which spatiotemporal learning methods should be considered in an application-specific manner. This is expanded to include the constraint of real-time recognition for motor rehabilitation. This means that partial or incomplete gestures and a network's ability to classify those are considered as well. Such a constraint serves as one motivation for the exploration into deep learning for this task; simple RNNs often fall short when there are issues of time dilation or less than complete samples. LSTMs excel at learning time-variant patterns but combining LSTM with convolution has been shown to perform better when spatial relationships are important. Additionally, low-cost sensors such as those used here are often subject to high degrees of noise and random variance, as is demonstrated in Figure 12. Variances between different subjects and how they perform gestures are even larger. Deep learning is deployed here to maximally eliminate all of these variances when the highest

classification accuracy is desired, and alternatively learn the variances when personalization or the identification of imperfections is an interest. The experiments performed here cover a broad range of hyperparameters and design structures that are used to analyze deep learning as a tool for low-cost gesture recognition platforms on low-resolution CSAs.

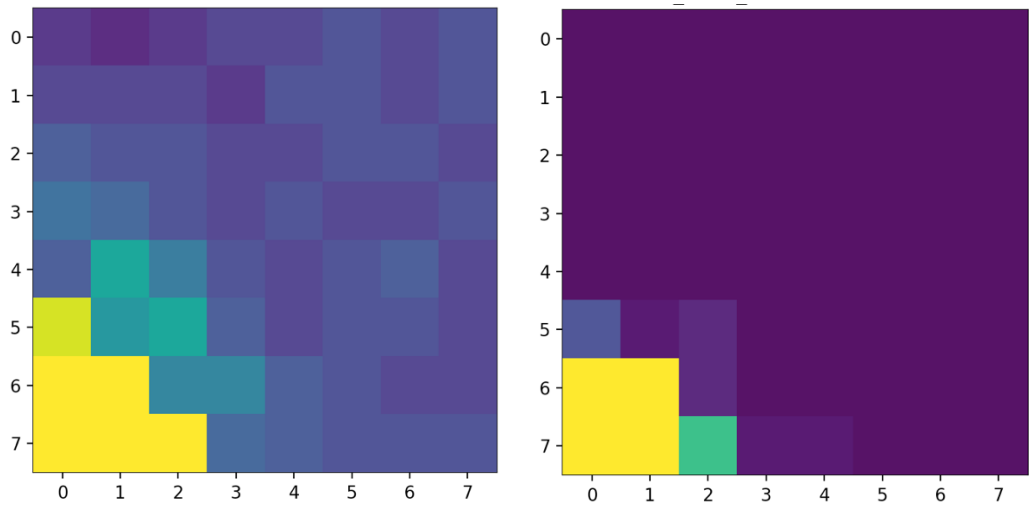

Figure 12. Two snapshots from the same CSA, taken at an identical point in time within the same gesture performed by the same subject. The second sample was recorded immediately after the first. Purple represents 0 intensity and yellow represents maximum intensity.

Chapter 4: Methodology

The requirements of this thesis include the ability to experiment on real capacitive sensor array data from several different designs, the ability to quickly create datasets from that data that simulate a variety of different applications, and the ability to easily compare the performance of different spatiotemporal learning architectures that are universally compatible with the collected data. This chapter outlines the steps that were taken in order to build working capacitive sensor arrays that interface with a microcontroller, collect gesture data, process and augment the data, and develop the algorithms and models to learn with the data.

4.1 Physical Approach

The experiments described in the next chapter are performed using a microcontroller, a programmer board, an activity board, several CSA designs made out of a variety of conductive and structural materials, and many custom-built connectors to facilitate the flow of information across the different components. This section provides specific detail on the construction of components and sensors, as well as the capabilities of the hardware used.

4.1.1 Hardware Specifications

The microcontroller used for collecting CSA data is the MSP430FR2676 MCU (microcontroller unit), made by Texas Instruments (Figure 5). For capacitive sensing, the MSP430FR2676 has 16 sensor inputs that are capable of sensing 16 electrodes in self-capacitance mode or 64 electrodes in mutual capacitance mode, and it supports the parallel scanning of up to 4 electrodes at a time at a sampling rate of roughly 50 Hz [68]. By default, the MCU has an operating voltage of 3.0 V but is tolerant to supply voltages from 3.6 V down to 1.8

V. Additionally, the MCU has five low-power modes including wake-on-touch support, five 16-bit timers for application use, over 40 I/O pins, and a 12-bit, 12-channel on-chip ADC. The FR2676 chip supports clock speeds of up to 16 MHz, as well as UART, IrDA, I2C, and SPI communication [68]. These features make it a highly capable board for embedded learning applications.

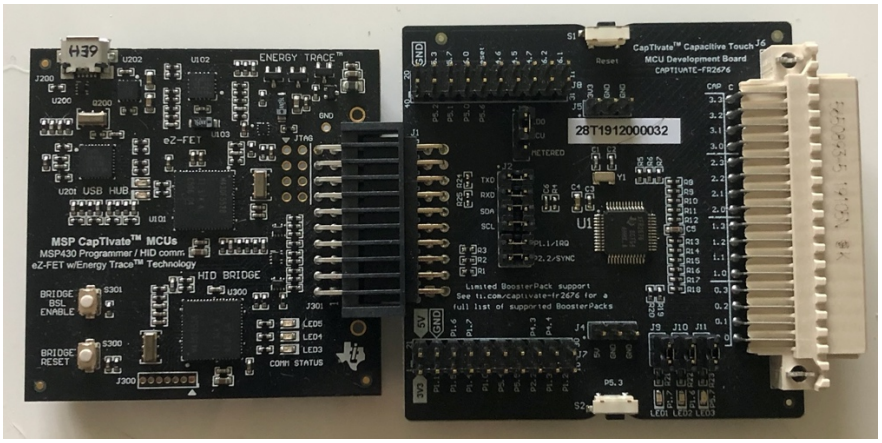

Figure 13. The MCU Programmer board (Left), connected to the MSP430FR2676 MCU (Right).

For testing designs, the MSP430 MCU Programmer board is used to program and debug the FR2676 chip while connected to a PC. The Programmer board is also capable of recording energy traces and other useful metrics while operating in the field but is not required for the MCU to perform inference [69].

4.1.2 Activity Board

The portable transparent activities table (referred to as the activity board) is a unique piece of equipment that has recently seen use in gesture classification and has been adopted to the field of motor rehabilitation [56], [57]. The activity board is made up of a 61x57cm wooden frame, that is hinged to a wooden base, such that the frame's angle to the ground is adjustable, which is beneficial to patients with different posture requirements. The wooden frame is topped

by a glass panel surface, with several slots underneath the surface to hold additional glass or plexiglass frames. These slots allow large CSAs to be inserted into the table underneath the surface, so that a subject can perform gestures on the glass surface without interfering with the CSA signals.

4.1.3 Sensor Designs

Two rigid CSA designs are applied here to collect experimental data. Sensor electrodes are made out of adhesive copper tape, conductive fabrics, and conductive threads. There are many methods for fabricating these electrodes; one method that is used frequently here utilizes the ScanNCut2 CM350e electronic cutting machine to precisely fabricate intricate electrode patterns. Both straight lines and diamond-shaped patterns are used. Plexiglass is the primary substrate between Rx and Tx layers.

Figure 14. 8x8 CSA used in activity board (Left). Activity board with CSA and hatched ground frames inserted (Right)

The first design is intended to be used for upper extremity motor rehabilitation and gestures involving the motion of the arms and hands [57]. The CSA is built on a 52x52cm

plexiglass frame that fits directly into the activity board frame slots. The frame is 2mm thick which represents the distance between Rx and Tx electrodes. The electrodes are 6mm-wide strips of copper adhesive cut into straight lines that cover the length of the frame. The Rx and Tx lines run perpendicular to each other as is typical for two-dimensional mutual capacitance CSAs. The Tx electrodes are placed on the surface side and run vertically, while the Rx electrodes are positioned on the opposite side and run horizontally. A group of jumper wires carries signals from each electrode to the MCU. To ground the CSA, an additional plexiglass frame is inserted into the activity board below the CSA at a distance of 20cm. The ground plane is a hatched-ground design made from 6mm strips of copper adhesive and connected directly to the MCU ground. Other details about the design's construction can be found in [57].

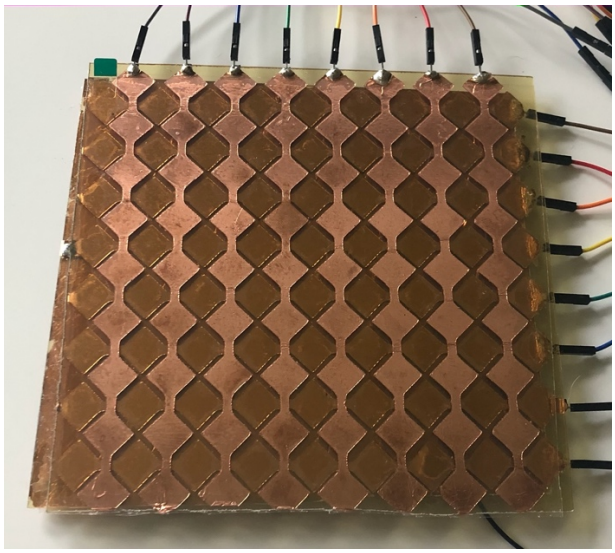

Figure 15. Small 8x8 CSA with diamond-shaped electrode pattern.

The second CSA design is much smaller and is meant for hand and finger gestures such as swipes or drawn symbols. It is very similar to the previous design with the main differences being its size and the shape of the electrodes. The CSA itself is built using two 16x16cm plexiglass frames, stacked on top of one another and held together with double-sided adhesive. The Rx and Tx electrodes are placed on opposite faces of the top frame, and the ground plane is

located on the underside of the bottom frame. The electrodes are laminated for protection and are cut in a diamond-shaped pattern connected by thin bridges. As before, the vertical Tx electrodes are on the top surface, and the horizontal Rx electrodes are underneath. The ground plane is a single uniform sheet of copper adhesive covering the entire frame surface and is connected to MCU ground. Both frames are 2mm thick which means all three electrode layers are spaced equal distances apart. The whole apparatus is attached to and sits on adhesive silicon mounts, so that the ground plane is raised about 4mm above whatever surface it is placed on.

Finally, both designs suffer from issues with cross-sensor noise. The hardware on the MCU is designed so that it should not matter what formation the signal bus is in when connected to the MCU. In this case, because the maximum number of MCU capacitive input pins are being used, the jumper wires carrying the signals are very close together, which can cause some parasitic capacitance between different wires. To handle this, each CSA inputs signals to the MCU in a unique formation to minimize the noise between electrodes.

4.2 Signal Processing

In order to learn from CSA data, the data must be accessed and organized in a way that can be understood by modern neural network architectures. First the MCU must record raw sensor data captured from the CSA using charge-transfer technology [70]. Specifically, the MSP430 repeatedly charges an on-chip capacitor with the charge from an individual sensing element (electrode pair) and counts the number of charge-transfer cycles that occur during a sampling window. The element “counts” are used as a metric for quantifying the capacitance change on each electrode. Higher count values represent a lower capacitance on that electrode. However, the count can be affected by a number of environmental variables such as temperature, moisture and the board’s operating voltage [71]. Due to its low-cost design, variations in the

physical sensor itself and cross-sensor noise can also affect the count slightly. To counter this, the long-term average (LTA) of each sensor is subtracted from its "count" to yield the "delta" value D at every timestep t (Equation 9). This delta value is used to numerically represent the amount of activity that a single electrode is reporting relative to the electrode's no-activity state. Finally, standard normalization is used on the delta values by subtracting the minimum value and dividing by the range of values across the entire sample, which results in all data existing within the range $0 \leq D \leq 1$ (Equation 10).

$$D_t = count_t - LTA_t \tag{9}$$

$$D_{norm} = \frac{D - D_{min}}{D_{max} - D_{min}} \tag{10}$$

This signal processing approach provides a direct digital representation of the proximity of a user's hand, arm, or other body part that can sufficiently serve as the input to deep learning architectures evaluated in this thesis.

4.3 Software Approach

A large number of libraries and tools are utilized to implement CSA design, signal pre-processing, data collection, data visualization, architecture development, and network training. Anaconda, a data science platform for managing machine learning libraries and virtual environments, is used to organize these tools and keep them working together cohesively. This section establishes the specific tools used, their purpose, and how they are used to accomplish the necessary tasks.

4.3.1 Design Tools

Several high-level design tools are used to interface with the microcontrollers. CapTIvate Design Center (CDC) is used to design capacitive sensor arrays with a GUI, which allows for several different sensor specifications to be tuned and controlled. From there, this software generates the C language code required to interface with the external sensors. Code Composer Studio is used to edit code and flash the programs onto the MCU. Once flashed, CDC can be used as a live GUI tool to visualize the sensor feedback and test the designs. This information is passed over UART from the MCU to the PC.

4.3.2 Data Logging

The CDC features a built-in data logging module that is used to record sensor data to csv files. Due to limitations with the MCU Programmer, the data logger sampling rate is about 30Hz for each electrode. The delta value is compared with a threshold to determine if there is sensor activity. For time-based gestures, many gestures can be recorded at once, where the threshold is used to tokenize the entire recording into individual gesture samples. When static images are desired, a simple pushbutton circuit is used to interface with the MCU and data logger, to specify when the snapshot should occur. From this point, several python scripts are used to convert the raw sensor data into a collection of samples that are each associated with a label. The NumPy and pandas libraries are utilized heavily here. To form a dataset, the samples are first shuffled, and then a predefined number of samples are chosen and split into train and test groups. These groups are stored separately as csv files. MATLAB is used to visualize the data and check for bad samples or issues in the signal processing pipeline.

4.3.3 PyTorch Implementations

PyTorch, similar to Tensorflow and Keras, is a large machine learning library that is implemented in Python and is the primary software tool that is used to develop and test the different models evaluated in this thesis. PyTorch is centered around organizing multi-dimensional arrays of data into structures called tensors, which are capable of storing gradients during the forward pass of a network so that the backpropagation process is efficient and quick.

First, in order to use the collected data with PyTorch, the PyTorch Dataset class is extended to a class written specifically to manage CSA data. The PyTorch DataLoader class is then used to handle batching the data for the training process and convert data stored in .csv files to learnable PyTorch tensors [72]. One obstacle for training time-based gestures is batching together sequences of different length. One of the methods evaluated circumvents this by normalizing all samples to a fixed number of frames. In general, a collate function is used dynamically with the DataLoader to zero-pad all samples of a batch to the length of the longest sample [72]. Sorting the data by sequence length beforehand also helps to minimize the amount of padding required for each batch. This strategy makes temporal learning in the network much simpler.

To implement the different network designs, a class extending the PyTorch Module class is created for each architecture [73]. The architecture is created by identifying and initializing the different modules, or layers, used within the architecture. The aforementioned modules are a combination of modules that are predefined within the PyTorch library and represent conventional neural network structures, and custom-built modules that utilize the predefined modules as well as other tools to implement experimental network structures in a bottom-up fashion. This format allows for a high level of control over the way each architecture operates.

Lastly, a forward pass function must be written to outline the specific sequence of modules used, and how the data is transformed in between modules. The PyTorch library contains many other useful modules that are used for training, such as a mini-batch gradient descent optimizer that is capable of performing the backpropagation of an architecture automatically.

Chapter 5: Experiments

A large assortment of experiments was conducted to investigate the challenges of spatiotemporal learning on capacitive sensor arrays. Because the goal of this thesis is to outline a design strategy for low-cost CSAs, many variables must be considered and tested as decision-making points that cover different applications, environments, and target demographics. Therefore, these experiments operate on a variety of human subjects, datasets, sensor devices, and spatiotemporal learning architectures that are utilized to simulate a wide range of applications.

5.1 Sensor Applications

Two devices were primarily used as platforms for data collection: 1.) the large 8x8 activity board CSA, and 2.) the smaller 8x8 diamond-patterned CSA (Section 4.1.3). For conciseness, these two devices will be referred to here as the "activity CSA" and the "small CSA," respectively. The activity CSA serves as the platform for bilateral arm gestures, the small CSA is the platform for multi-touch touchscreen hand gestures, and both CSAs are used for the EdgeWrite dataset to serve as a comparison point between the two sensor devices. These datasets are detailed in the next section.

5.2 Datasets

Four datasets are utilized as blueprints for data collection; two are derived from the EdgeWrite dataset for single-stroke gestures and are identical except for the sensor used, and the other two are custom-designed to incorporate different types of motions discussed here. All gesture samples used for training were recorded specifically for this work by multiple subjects

that performed gestures on the previously described devices. This section details the following datasets, which will be referred to by their underlined name for the remainder of this document:

1.) Bilateral: Exercises that involve moving two arms simultaneously have been shown to be a necessary part of the stroke recovery process [5]. It has even been suggested that when utilized correctly, bilateral arm gestures can be much more effective than unilateral gestures [5], [74]. This dataset establishes several bilateral gestures that involve moving both hands across the activity CSA in a mirrored fashion with the intention of mimicking the arm movements incorporated in bilateral rehabilitation exercises. These gestures focus on reaching the arm to different positions with the hand fixed, which is one of several key characteristics of useful bilateral motions studied in the past [5]. As such, subjects were instructed to perform the gestures with their wrists on the surface of the activity board and both hands in a static fist shape, sliding the fists across the board at a consistent pace (Figure 16). The gestures were designed to share similar movements in order to challenge the architectures. Two subjects performed the four different gestures a fixed number of times each for a total of 800 samples.

2.) EWActivity & EWSmall: The EdgeWrite dataset was originally created as a baseline for text entry across multiple platforms using single-stroke motions and can be used in a variety of applications [75]. Several works have adapted this dataset for analysis of gesture recognition on wearable capacitive sensor arrays [64], [63], [67]. Five gestures were adapted for network analysis from the EdgeWrite dataset, specifically the letters H, M, N, W, and Y (Figure 17). which were selected based on their high similarity in order to challenge the different architectures tested here. These gestures were recorded on both the activity CSA and the small CSA, forming two distinct but very similar datasets that differ only in the sensor application used to record them. Together, they serve as a baseline between the two primary sensor applications

evaluated here. A total of 1000 samples were collected as training data for each dataset. Additionally, several variants of EWActivity were recorded on the activity board with additional subjects and additional instructions; these variants are covered in Section 5.7.

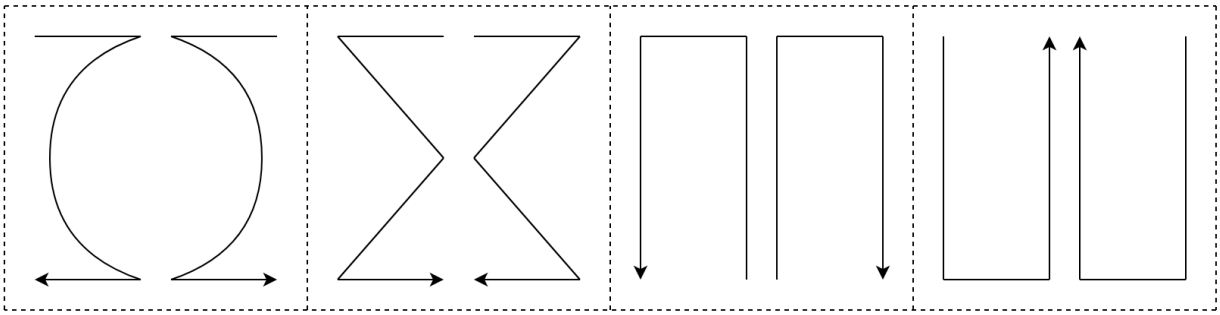

Figure 16. The four gesture types recorded in the Bilateral dataset. Each gesture is performed with both hands moving simultaneously in a mirrored fashion across the activity board.

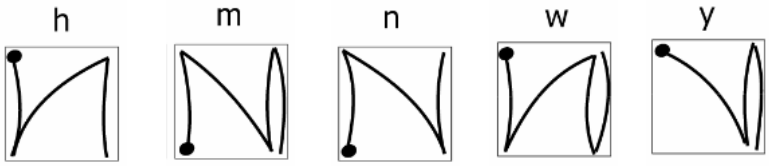

Figure 17. Five letters from the EdgeWrite Dataset used as gesture templates.

3.) MultiTouch: In addition to proximity-based motor rehabilitation, touchscreen applications for home automation and other complex HCI systems are evaluated here. The MultiTouch gesture set is specifically collected with variance in mind, as would be appropriate for a context-aware system that aims for the highest classification accuracy above all else. Each sample in the MultiTouch gesture set is performed with a level of variance in rotation, intensity, and spatial location. The MultiTouch dataset consists of several two-digit and five-digit hand gestures, including a pinch, a spread (opposite of pinch), a spiral motion, and a swipe. A list of gestures performed in this dataset can be found in Table 5. Each gesture includes a variant that is labeled as a separate class; the spiral is performed both clockwise and counterclockwise; the swipe, the pinch, and the spread are all performed with two and five fingers at a time. These gestures were recorded on the small CSA and vary largely from sample to sample, whereas the

other gesture sets are fairly consistent across all samples. Two subjects recorded 50 repetitions each for every gesture.

5.3 Gesture Data Collection

Five human subjects were asked to perform the gestures that make up the datasets mentioned in the previous section. Four subjects recorded gestures for the EWActivity and EWSmall datasets, and two of those subjects recorded gestures for the MultiTouch and Bilateral datasets. The fifth subject later recorded a number of gestures for EWActivity test variants (Section 5.7). None of the subjects are afflicted by any known motor impairments or suffer from any of the medical diseases discussed in this thesis. For consistency, and to reduce friction on the sensors, the subjects wore conductive gloves known for their compatibility with mobile phone screens. Note that the gloves do not affect the measured capacitance changes although regular gloves may reduce conductivity. This section details the instructions that subjects were given for each set of gestures and provides insight to what variations these instructions led to and their implications.

1.) The EWActivity dataset was recorded first. Each participant was shown an image depicting the EdgeWrite letter and its starting point (Figure 17), then asked to perform the gesture a specified number of times without further instruction. Each subject showed slightly different mannerisms in their arm movement, speed, and hand position. Two subjects made a fist, one subject held their hand in a position similar to if they were preparing to type on a keyboard, and one subject actually "drew" the gestures, making quick, straight movements with short pauses in between as if they were actually drawing with a pencil on paper. However, each subject maintained personally consistent gesture performance across all of their gestures. No instruction was given on which hand to use; all subjects used their right hand.

2.) Recording the EWSmall dataset was conducted in a similar manner to the EWActivity set. Each subject was once again showing an image of each gesture, without being given any other instructions. Again, variance between subjects was visible, but individual subjects performed most of their gestures with a high degree of consistency. One subject used their thumb to "draw" the gestures on the board, one subject used two fingers, and the other two used one finger. All but one of the subjects held their hand low and close to the board while performing the small gestures, leading to more than just their finger being visible on the signal readings, due to the close proximity.

3.) For the Bilateral dataset, each subject was asked to make a flat fist for performing gestures, which corresponds to the kind of position that is beneficial for motor rehabilitation. For each gesture, they were shown a diagram of what the motion should look like (Figure 16), a short verbal description, and one demonstration. This led to very consistent movements and paces across all subjects which is similar to what a patient should be doing for rehabilitation.

4.) Similar to the Bilateral dataset, the MultiTouch dataset was recorded after giving the subject a verbal description and a demonstration of each gesture. However, they were also instructed to use a large amount of variance in the spatial location, size, and orientation of the gestures, as well as some variance in the pace. This is because the MultiTouch dataset intends to test architectures on how well they can classify a gesture where maximum prediction accuracy is desired, such as in a home automation system.

5.4 Architectures

Three different deep spatiotemporal learning architectures are evaluated on low-cost capacitive sensing applications. ConvLSTM, CLDNN, and 3D Convolution are all implemented and tested for each of the applications described above. Each architecture is trained with varying

hidden dimensionalities and other hyperparameters within the network. For the architectures involving recurrent structures (CLDNN and ConvLSTM), this work also presents evaluations where LSTM cells are replaced with RNN and GRU cells. Note that since both CSAs are 8x8, the inputs to all networks are three-dimensional tensors of shape TxNx64, where T is the number of frames for all sequences in the batch and N is the batch size.

5.4.1 CLDNN

C-LSTM typically refers to the combination of convolutional layers, alongside or followed by LSTM layers, in some large, deep network. One of the first works that introduced this presented an architecture they called CLDNN, short for Convolutional, Long Short-Term Memory, Deep Neural Network [76]. This architecture has been previously shown to have high success when learning bilateral gestures on a low-cost CSA [57] and has been re-adapted here to fit the experiments performed. CLDNN is essentially the combination of convolutional layers, recurrent layers, and fully connected linear layers in one deep architecture. An architectural summary of the design used in experiments is included here:

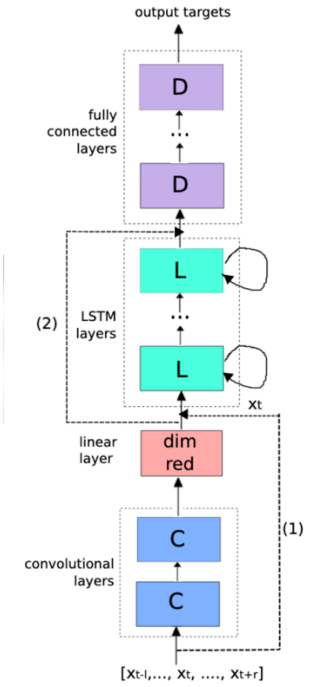

Figure 18. The CLDNN architecture [76]

Key frames are selected from the input sequence such that they are equally spaced apart by a constant step size. Each key frame X_t is convolved with l previous frames (X_{t-l}, …, X_t) as input channels in two back-to-back 2D convolutional layers with a kernel size of 3x3, and sufficient zero-padding so that there is no spatial reduction. The two convolutional layers each produce $\frac{H}{2}$, and H feature maps, respectively. The output channels are then downsampled through max pooling to a 4x4 spatial resolution and concatenated temporally such that the time dimensionality of the whole batch is reduced to a size of T. Each output at timestep t is then processed by a linear layer to reduce its feature dimensionality by a factor of 8.

The output of the linear layer at each timestep t is then concatenated with the features from the original key frame X_t to create a stronger short-term connection (1) at each timestep [76]. All T timesteps are then unrolled and input to an LSTM with two layers. The recurrent output is concatenated with the output of the linear layer from before along the time dimension

(2). The max value from every timestep *t* is then input to a final group of two linear layers such that the number of features is reduced first by half, and then down to the number of classes. This network is very complex, yet it maintains a very low parameter space due to the connections that skip layers.

5.4.2 ConvLSTM

"ConvLSTM" was originally introduced as a method of predicting rainfall intensities over a short period of time [77]. Conceptually, it is identical to a typical LSTM framework, with one main deviation: instead of using matrix multiplication to connect the input and hidden states with the network's weights, the states are convolved with weights using 2D convolution. Equation 7 in Section 2.2.2 lists the six equations that are considered standard to LSTM. The modified convolutional LSTM equations are shown in Equation 11:

$$\begin{aligned} i_t &= \sigma(W_{xi} * X_t + W_{hi} * h_{t-1} + b_i) \\ f_t &= \sigma(W_{xf} * X_t + W_{hf} * h_{t-1} + b_f) \\ g_t &= \tanh(W_{xg} * X_t + W_{hg} * h_{t-1} + b_g) \\ c_t &= f_t \circ c_{t-1} + i_t \circ g_t \\ o_t &= \sigma(W_{xo} * X_t + W_{ho} * h_{t-1} + b_o) \\ h_t &= o_t \circ \tanh(c_t) \end{aligned} \tag{11}$$

where * represents the convolutional operator [77]. This approach essentially transforms the hidden state vector into a three-dimensional matrix where every spatial "cell" contains its own hidden state vector.

The ConvLSTM implementation used in experiments here is adapted from a public PyTorch implementation [78] and extended to include the option to use GRU or RNN cells instead of LSTM cells. The implementation first inputs the features to the ConvLSTM module with two recurrent layers. After recurrent convolution takes place, the output features are in the shape *T*x*H*x8x8, where *T* is the number of timesteps, and *H* is the number of hidden nodes. The

largest feature value along the time dimension is kept and the rest discarded. The remaining *H*x8x8 features are input to two more linear layers, reducing the number of features by half first, and then down to the number of classes for prediction.

5.4.3 3D CNN

In addition to combining convolution with recurrent structures, three-dimensional convolutional neural networks are useful for learning spatiotemporal features. The 3D CNN model evaluated here is inspired by a relatively early work in the field of video-based action recognition [79]. C3D is also another important architecture that was considered, implemented, and tested here, but is not included in the official evaluations [80]. To clarify, when shapes are described here, the time dimension is first, followed by height and then width.

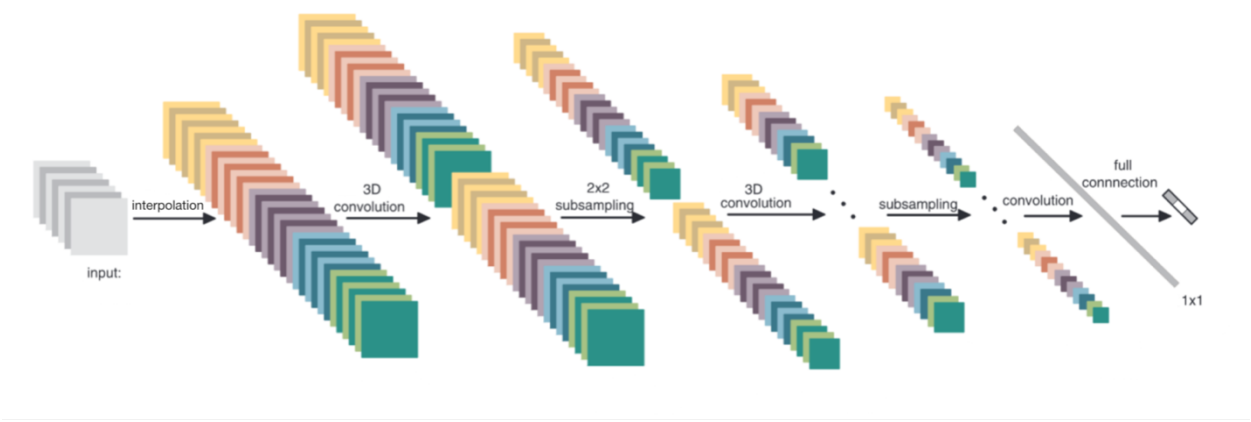

Figure 19. The 3D CNN architecture [79].

The 3D CNN evaluated here has three convolutional layers that convolve in three dimensions, and two max pooling layers that subsample in three dimensions (Figure 19). Instead of hidden nodes, the constant *H* that is used for hidden dimensionality in the other three architectures is used here to define the number of channels between convolutions. The number of output feature maps that each convolutional layer produces are, from first to last, H, $1.5 * H$, and

$3 * H$. The first convolutional layer uses the standard kernel size of 3x3x3, no padding in the time dimension, and 1 pixel of zero-padding in the spatial dimensions, such that there is a small reduction in depth but no reduction in height or width. This is followed by a max pooling layer that only pools in the time dimension with a stride and kernel size of 2, to keep the spatial resolution at 8x8 but reduce the depth by half. The next convolutional layer is the same as the first, except for the number of input and output channels as previously mentioned. Another max pooling layer follows, but this time there is equal pooling in all three dimensions that reduces all by half. This layer outputs a 6x4x4 3D image. The final convolution layer utilizes a 6x4x4 kernel that is the same size as the input, meaning that the output feature maps are all 1x1x1. This makes it somewhat similar to a linear layer with grouped connections. The final fully connected linear layer reduces the number of features to the number of classes in the dataset.

5.4.4 Standard LSTM

The final architecture evaluated here is a very simple LSTM network that serves as a control group. This network requires far fewer parameters than the other networks, so its performance is generally worse than the other architectures on most datasets shown here. The network consists of a single recurrent layer that uses LSTM cells but is switched to GRU or RNN for the recurrent cell structure comparison. Recurrent layers do not inherently learn spatial relationships as their inputs are typically two-dimensional, where one dimension is time. This is true for this architecture as well, meaning that the network simply receives 64 features at every timestep as input without an understanding of their order. The output hidden state is maximized along the time dimension similar to the previous architectures and then finally input to two linear layers such that the number of features is reduced by half and then down to the number of classes in the dataset.

5.5 Hyperparameters

There are several hyperparameters associated with the previously described architectures that are variable and depend on the designer's specific implementation. The modification of these parameters can drastically affect a network's performance depending on the classification task and the type of parameter. The primary hyperparameters included in experiments are activation functions, recurrent cell type, and the number of hidden nodes used in the recurrent layers. When one of these parameters is not being directly evaluated, a default value is used. There are other important hyperparameters, such as kernel size for convolution, that are not being investigated and held constant simply for consistency. These default parameters are listed in **Error! Reference source not found.**.

Table 2. Default Hyperparameters and Architectural Constants.

Parameter/Function/Structure	**Default**
Activation function	ReLU
Recurrent cell structure	LSTM
Number of hidden nodes	20, 40, 60, 80, 128
Kernel size	3x3
Stride for convolution	1
Dropout (input)	0.2
Dropout (convolution)	0.3
Dropout (recurrent)	0.35
Dropout (linear)	0.5

ReLU activation is used for all linear and convolutional layers. Recurrent layers are activated by the output activation that is standard to each type of recurrent cell, where LSTM is the default cell structure. In general, if the number of hidden nodes is not specified directly, then the best performance between 40 and 60 nodes is used for comparison. All convolutional kernels are 3x3x3 or 3x3 in size, except for the final convolution layer in Conv3D, which is more closely

tied to a linear layer than a convolutional layer. Finally, dropout is used in all networks. Input features are directly dropped randomly

5.6 Training Strategy

There are many hyperparameters associated with convolution, recurrence, and other deep learning frameworks as well as training a network overall. In an effort to minimize the experimental variance, all hyperparameters not being investigated directly are kept constant across different architectures. This is a necessary step because every network can be optimized independently by an expert, but the goal of this study is to identify the best strategy for a designer that is a non-expert in deep learning. Optimization techniques, data handling, and other constant parameters are summarized in Table 3, and outlined here:

Mini batch gradient descent is used to train the models, using a batch size of 5, for 100 epochs. A small weight decay value of 0.001 is used to offer light regularization to the training process. Due to the relatively small sample size of the datasets used here, extremely high accuracies can be achieved very quickly through gradient descent optimization alone. Momentum-based updates can accelerate training and improve accuracy in fewer epochs when used properly [81]. However, training a network on a small amount of data in this way can often lead to overfitting which results in much lower success in real-world inference, as unseen variations are encountered more. This effect was very apparent from early testing; therefore, momentum updates are not used, and a relatively small learning rate is used with a simple schedule. Specifically, the schedule applied is a cosine annealing learning rate schedule beginning at 0.01 with no restarts [82]. Certain architectures still learn much faster than others however, so early stopping is utilized as a way to generalize their final state and prevent them

from heavily overfitting the training data. In these experiments, if the training loss does not improve by 0.05 after 5 epochs, then training is halted early for that architecture.

The output of each model is a tensor with C class predictions for each class, where C is the number of classes in the dataset used for training. The LogSoftmax function is then applied to the output where the Negative Log-Likelihood Loss function is used to obtain the loss value for the mini-batch. The process of using these two functions consecutively is also known as Cross-Entropy Loss. From this point, gradients are computed using standard backpropagation through time (BPTT) for recurrent structures. Mini batch gradient descent is then used to update the parameters throughout the model. This is achieved with a Stochastic Gradient Descent optimizer class in PyTorch, which makes the process very simple and does not require mathematical understanding from the designer.

Table 3: Training Constants. These training parameters are used across all experiments unless otherwise specified.

Hyperparameter	**Value**
Batch Size	5
Number of Epochs	100
Early Stopping	Train Loss (0.05)
Loss Function	Negative Log-Likelihood Loss
Learning Algorithm	Stochastic Gradient Descent
Learning Rate	0.01
Learning Rate Schedule	Cosine Annealing LR [82]
Momentum Updates	0 (Not used)
Weight Decay	0.001

Prior to training, datasets are shuffled randomly, and then split into training and testing groups at a 20% test ratio. Training data is shuffled after every epoch to introduce some additional regularization. In most cases, each sample in a batch is a sequence with a different length, so the samples must be aligned temporally for temporal convolution to work. Therefore, a

collate function is used to dynamically zero-pad the sequences of a batch each time a batch is generated. In this case, zero-padding means that if one clip is 10 frames shorter than the longest clip in the batch, then 10 frames of zeros are appended to the end of the shorter clip.

5.7 Other Considerations

The overall purpose of the datasets summarized previously is to compare the performance of different architectures when deployed to different applications. In general, the results outlined in the next chapter are achieved by training and validating the network with a dataset that is randomly split into a train group and a test group. The test group is used as a benchmark for a network, and the network does not learn from its attempts to classify the test data; this is a very common practice in the realm of deep learning. When training networks with a large volume of varied data, such as the Moving MNIST dataset, this approach is beneficial because samples in the test group might be very unique when compared to a sample in the train group. However, a major fallback of this study is that when a subject is asked to perform a large quantity of gestures, many of the samples are extremely similar. There is enough noise and variance between individual gestures of the same class for the network to learn, but there is still a large degree of congruency between the test and train groups, which leads to a level of overfitting that is hard to precisely identify without a unique test set.

In the case of motor rehabilitation, this is not entirely a problem, as it is encouraged for a patient to be as consistent as possible. For example, a patient may create a training set under supervision from a rehabilitation therapist, and therefore a network's score would be correlated with the patient's consistency. In that scenario, a lower score indicates the need for improvement, and a higher score indicates a well-performed gesture that is beneficial to

recovery. Still, the sample similarity is not always ideal, and should generally be considered when observing evaluations, excluding networks trained on the MultiTouch dataset.

In an effort to counter the similarity between train and test groups, additional samples were collected on the EWActivity dataset to serve as unique test benchmarks. Note that these are merely test groups, meaning that networks are still trained with the original EWActivity dataset. A fifth subject performed the five gestures normally for 50 samples, then again with a variety of speeds for 50 samples, and again with "bad" or incomplete gestures for 50 samples. The first set is referred to as EWActivity-5 and is completely normal but enables the testing of a network on a new subject with unique mannerisms not found in the training set. The second set, referred to as EWActivity-slow, includes samples that range from extremely slow to slightly faster than normal speed, as well as samples that vary in speed from start to finish. This set serves to test how a network handles extreme temporal variance. The third set, EWActivity-poor, is made up of gestures that are generally poorly performed compared to the main pool of EWActivity samples. Gestures that start or end early (incomplete), gestures that only take up a small subspace of the board rather than the full area, and gestures that have rotational variance are all examples. This set is useful for evaluating how well a network can classify poor gestures after being trained with "perfect" gestures.

Chapter 6: Results

This chapter outlines the evaluations of the ConvLSTM, CLDNN, and Conv3D architectures on the different sensor applications and datasets defined previously. Additionally, LSTM is evaluated as a control architecture for comparison. All evaluations listed here are obtained from networks that are trained on "equal ground", meaning that they are generally not optimized fully with the best training parameters and long, thorough training processes. Rather, the results are obtained using consistent training cycles for each experiment. This serves to provide a basic, low-level understanding of what the architectures are capable of with minimal understanding from a non-masterful designer.

6.1 Evaluation Methodology

The results shown here include classification accuracy and loss from both the train set and the test set during the training process. Additionally, architectures trained on the EWActivity dataset are also tested against the dataset's aforementioned test variants. To reiterate, when the number of hidden recurrent nodes is not being directly investigated, each experiment was repeated multiple times with different numbers of hidden nodes, specifically 20, 40, 60, 80, and 128 unless otherwise specified, and the best result among those repetitions was chosen for comparison. Accuracy and loss are both recorded directly from the PyTorch framework during the training process. Post-training tests are performed by loading a pre-trained network, computing the forward pass, and recording the predictions made at the output layer. Graphs are generated with Excel and matplotlib.

6.2 Architectural Hyperparameters

Three common activation functions, three recurrent cell structures, and several different hidden dimensionalities are evaluated across the four architectures. Each of these hyperparameter groups are applied to all four architectures that are trained on the EWActivity dataset,

6.2.1 Activation Function

The three activation functions analyzed here are the Rectified Linear Unit function (ReLU), the Tanh function, and the Sigmoid function. Figure 20 shows a comparison of activation functions for all four architectures on the EWActivity dataset. Only architectures with 40 and 60 hidden nodes were trained and evaluated for this experiment. ReLU activation led to the best results across all networks on average when all test sets are considered. For ConvLSTM and LSTM however, Tanh improved classification accuracy on most EWActivity variants. This indicates that ReLU and Tanh are both good activations for this application and should be chosen dependent on the specific task. Additionally, it is clear that sigmoid is not fit for this application. Interestingly, all networks trained with the sigmoid function on the EWActivity variants achieved a best accuracy of exactly 20%. Comparisons of activation functions on the other EWActivity test variants can be found in the Appendix.

6.2.2 Recurrent Cell Structure

For the architectures that include recurrent layers, substituting the default LSTM structure with GRU and RNN structures is analyzed here. Note that 3D CNN is excluded here as it does not contain any recurrent layers. Figure 21 shows a comparison of the best test accuracies achieved by the four architectures on the EWActivity-slow test set with different recurrent

structures. Generally, it is clear that LSTM performs much better than GRU and RNN on the EWActivity dataset. This is especially true with the EWActivity-slow test variant, which does not come as a surprise because LSTM excels at recognizing patterns even when they are stretched over time. Also, while GRU is often claimed as a sufficient alternative to LSTM, GRU did not perform better than RNN or LSTM in most experiments here, indicating that it should not be considered for these applications. Comparisons of recurrent cell structures on the other EWActivity test variants can be found in the Appendix.

6.2.3 Number of Hidden Nodes

Several different recurrent hidden dimensionalities were tested on the EWActivity dataset to determine if there is an optimal inner dimensionality of each network. Note that the 3D CNN architecture does not have recurrent "hidden" nodes, but the number of inner convolution channels between layers is determined based on the constant represented by H, which is shared by the other architectures. Figure 22 presents a comparison between different numbers of hidden nodes when tested on the EWActivity-slow test set. This experiment unsurprisingly failed to identify a consistently optimal number of hidden nodes that is applicable to every scenario. In many cases, more hidden nodes led to better performance with one architecture, but too many would lead to much worse performance. In some cases, the fewest nodes led to the best performance for a specific architecture. These observations show that the optimal hidden dimensionality is highly dependent on the specific application and network architecture. Comparisons of hidden dimensionality on the other EWActivity test variants can be found in the Appendix.

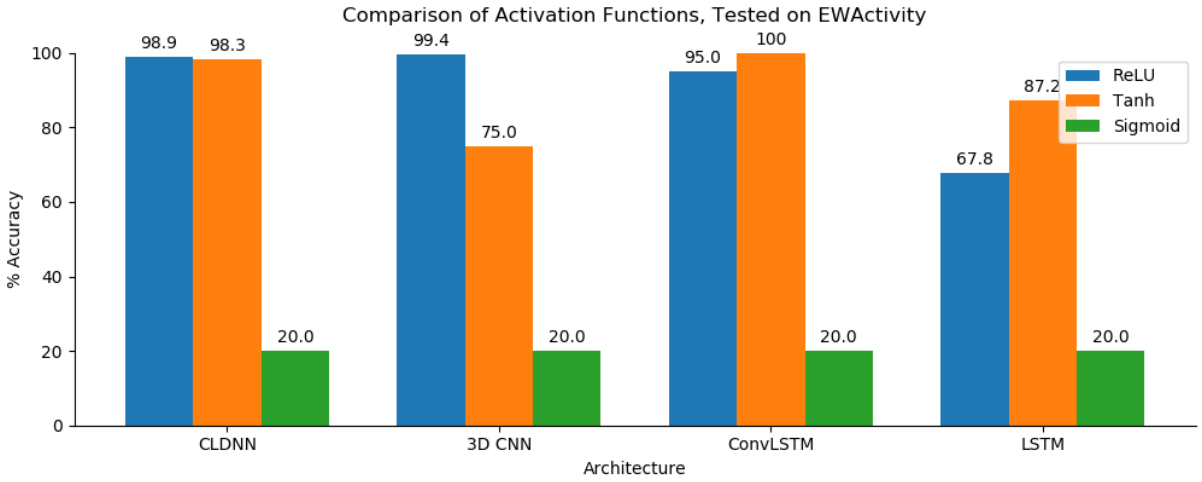

Figure 20. Comparison of activation functions tested on the EWActivity test set. Each color represents one of the three activation functions evaluated.

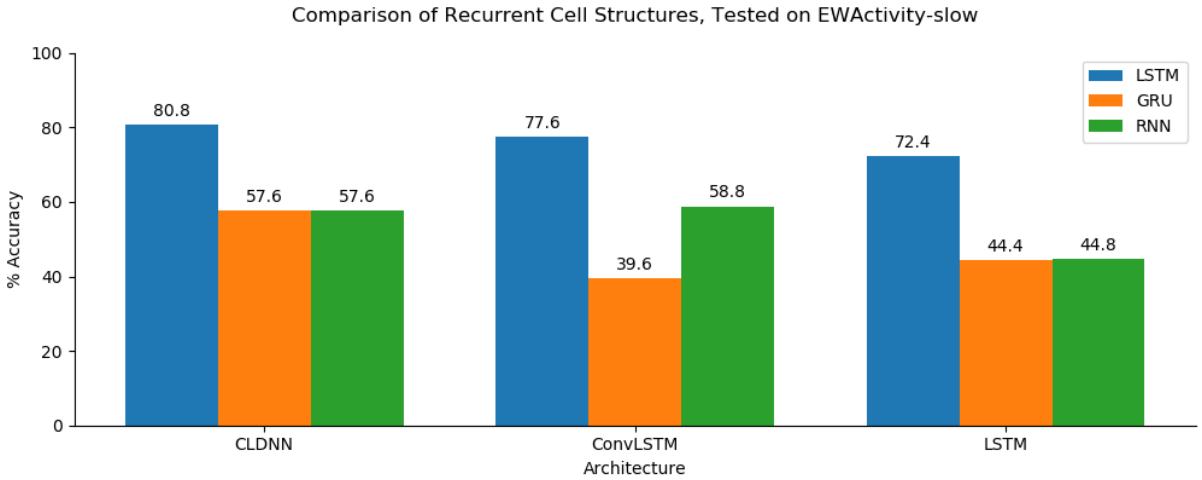

Figure 21. Comparison of recurrent cell structures tested on the EWActivity-slow test variant. Each color represents one of the three recurrent cell structures evaluated.

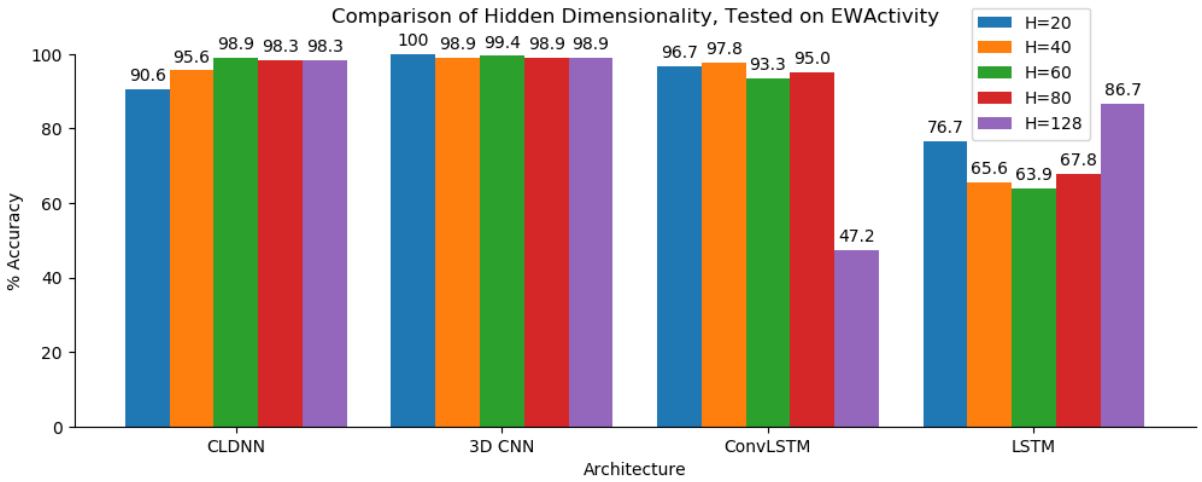

Figure 22. Comparison of hidden dimensionality tested on the EWActivity test set. Each color represents the number of hidden nodes in the recurrent layers.

6.3 Other Applications

The tests performed on the other sensor applications are simpler and are primarily included to act as a comparison to different kinds of CSA platforms. All tests evaluated on these models are repeated experiments from the EWActivity set to provide insight on how results may change across applications with different sizes and structures. Overall, the ConvLSTM and LSTM architectures performed very poorly while the CLDNN and 3D CNN achieved high accuracies.

6.3.1 Bilateral

This dataset was designed to include consistently paced movements of multiple conductive bodies moving simultaneously. Unsurprisingly, performance is very good as the dataset does not contain a large amount of variance. Figure 23 shows the classification accuracy of all four architectures on the Bilateral test set.

6.3.2 EWSmall

The results shown in Figure 24 are primarily a cross-device comparison to the EWActivity results obtained in Figure 22. Contrary to the EWActivity dataset, the gestures performed on the small CSA required more hidden nodes to achieve the same level of success with the CLDNN network, and the ConvLSTM simply did not perform well compared to the ConvLSTM trained on activity board data. This is likely due to the close proximity of the user's wrist and forearm on the small CSA. When performing gestures on the small CSA, several subjects used a technique that includes a low-hanging wrist, which adds noise to the electrodes near where the gesture is being performed. This observed noise was quite large for some subjects, covering a significant portion of the board.

6.3.3 MultiTouch

While the dataset is much different, the results are largely similar to the previous small CSA application. On average, ConvLSTM and LSTM both performed even worse on MultiTouch than on EWSmall, while CLDNN classification was better overall (Figure 25). This provides some indication about the success of these architectures on the smaller CSA board in general, due to issues of noise from extra-gestural parts of the body. The resulting CLDNN performance is more significant here because the MultiTouch dataset contains much more variance than EWSmall, which means the test data is less similar to the train data, giving the networks a harder problem to solve.

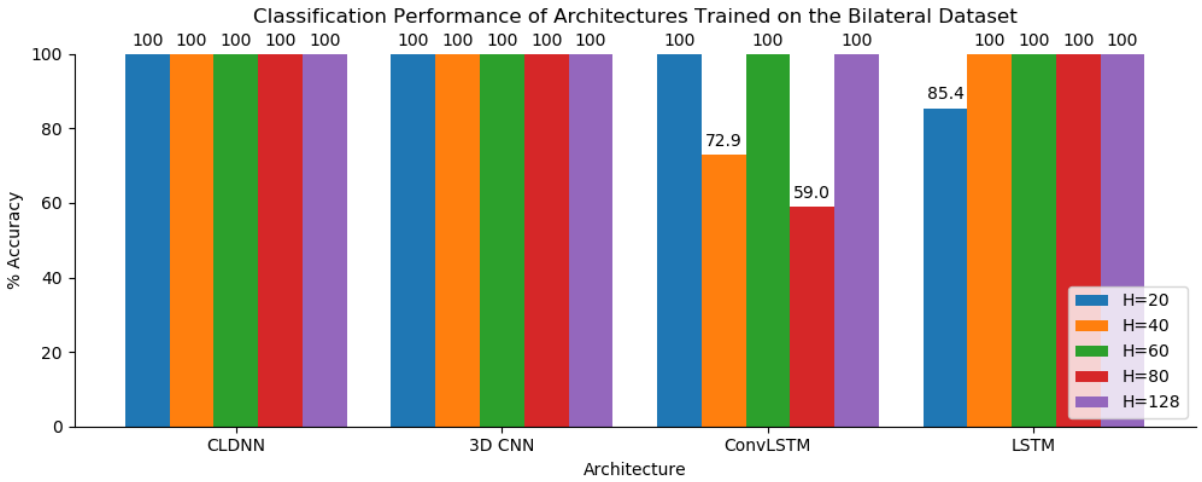

Figure 23. Classification performance of architectures trained on the Bilateral dataset. Each color represents the hidden dimensionality used.

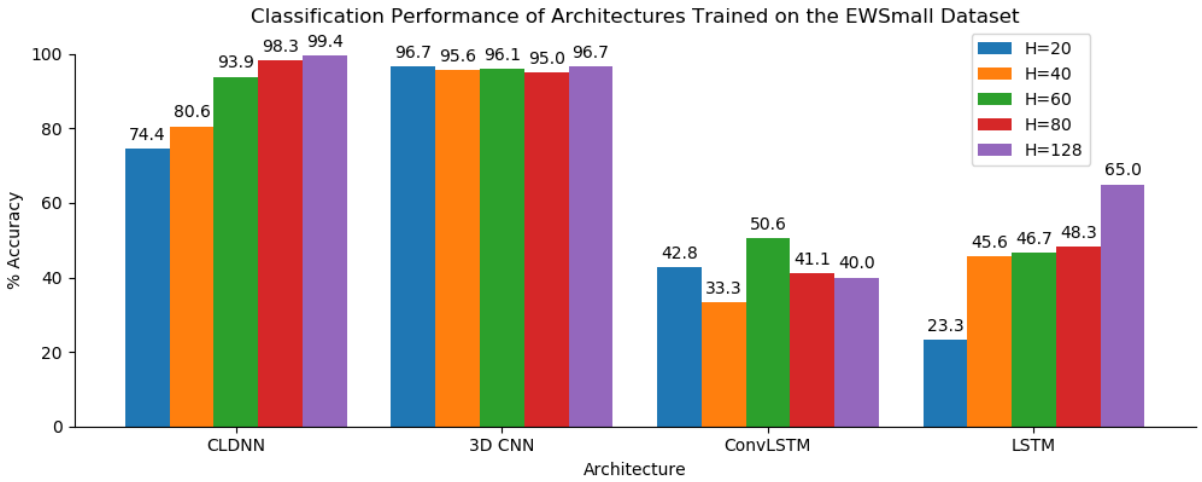

Figure 24. Classification performance of architectures trained on the EWSmall dataset. Each color represents the hidden dimensionality used.

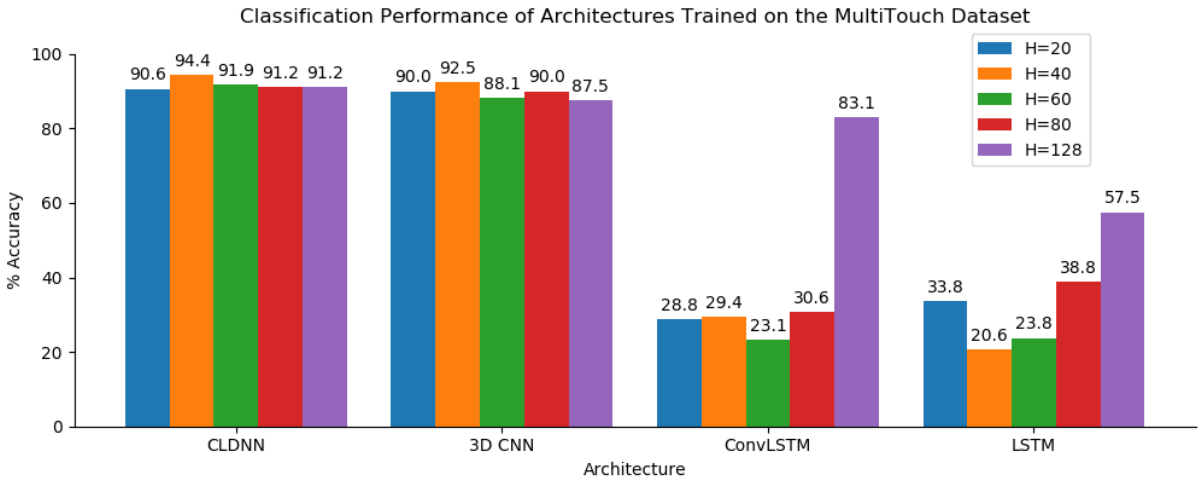

Figure 25. Classification performance of architectures trained on the MultiTouch dataset. Each color represents the hidden dimensionality used.

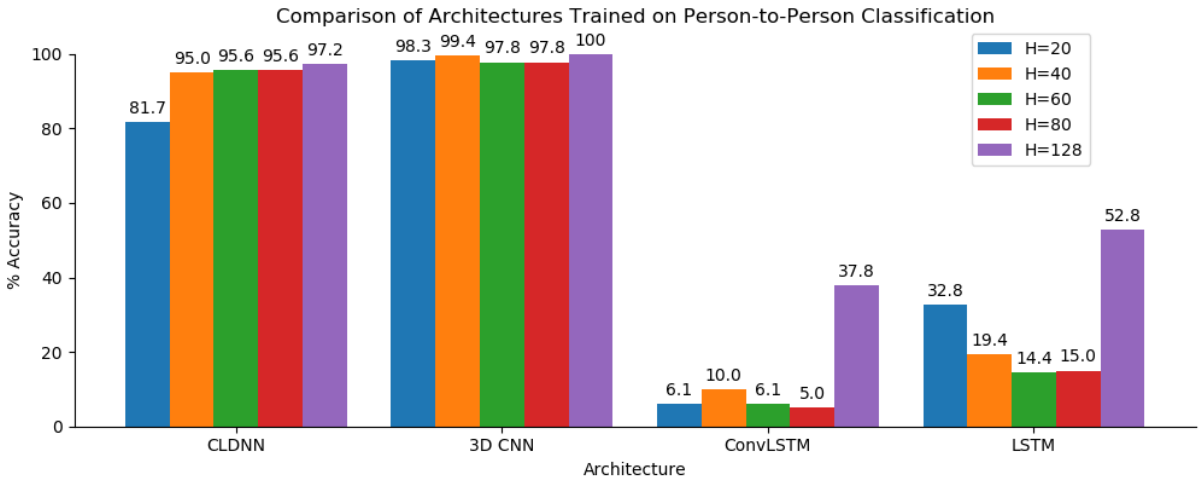

Figure 26. Comparison of architectures trained on a person-to-person variant of the EWActivity dataset. Each color represents the hidden dimensionality used.

6.4 Person-to-Person Classification

For home-automation applications, as well as motor rehabilitation applications for different patients with different specific disabilities, it may be interesting to gain an understanding of which architectures can tell one person's gestures from another person's gestures and pick up on personal similarities. The EWActivity dataset shown repeatedly in this chapter is made up of five gestures from four different subjects, separated into five classes, one

for each gesture. This experiment observes what happens when each subject is given their own class for each gesture, for a total of twenty classes. The results shown in Figure 26 are even more polarizing than previous experiments. CLDNN and 3D CNN performed quite well when distinguishing between subjects, which demonstrates a strong ability to learn personalized features. In contrast, ConvLSTM barely achieved 1 in 20 correct predictions, which is essentially the same as guessing. LSTM did not perform quite as badly but is clearly not a useful architecture for this task. The two successful architectures performed at a similar level to their accuracies on the original training dataset. This implies that personal distinction is easily achievable for motor rehabilitation or home automation with the applications shown here.

6.5 Combined Learning

In an effort to improve the results on the slow and poor variants of the EWActivity dataset, models were trained again where select samples from the training dataset were replaced with samples from each test variant. Those selected samples were simply labeled with the gesture that they represent regardless of their quality. In theory, this gives models more experience learning different forms and characteristics of the same gesture instead of just the consistent particular motions associated with each gesture. The models trained on the combined training dataset were then re-tested on the EWActivity test variants, resulting in significantly better accuracies on both EWActivity-slow (Figure 27) and EWActivity-poor (Figure 28). This training process showed significant improvements compared to the original training group. Previously, the highest achieved accuracy on the EWActivity-slow dataset was a mere 80.8%, with most performances coming in lower than that. The combined training however brought CLDNN up to 99.6% on the slow gestures. ConvLSTM, which usually did not perform well on the EWActivity test variants, scored over 90% on the poor gestures when trained with combined

data. Figure 29 and Figure 30 show the spread of network performance for all EWActivity test variants when trained on the original training data, and the combined data, respectively; comparisons to the original training process for the other two test variants can be found in the Appendix.

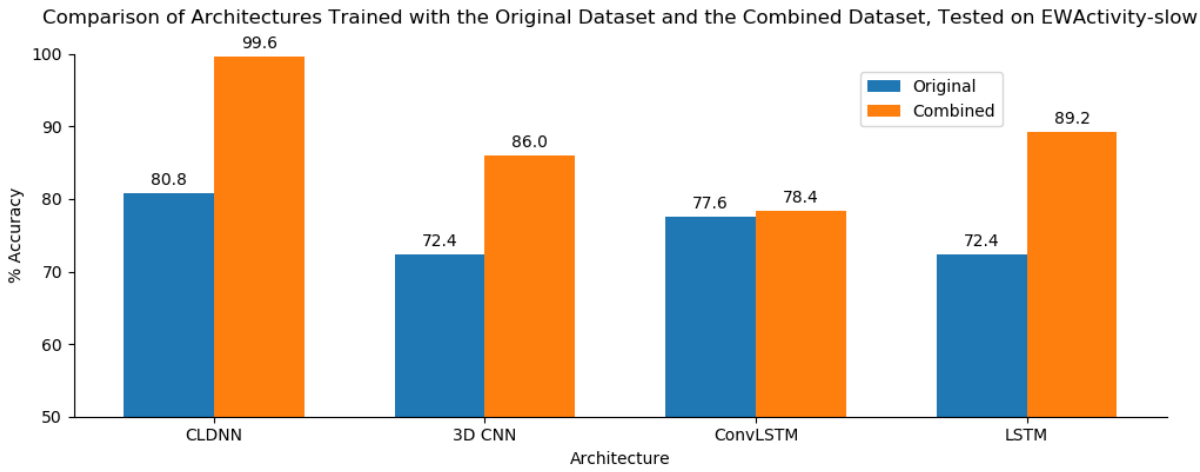

Figure 27. Comparison of architectures trained on the original training set vs. architectures trained on the combined dataset; both are tested on EWActivity-slow dataset.

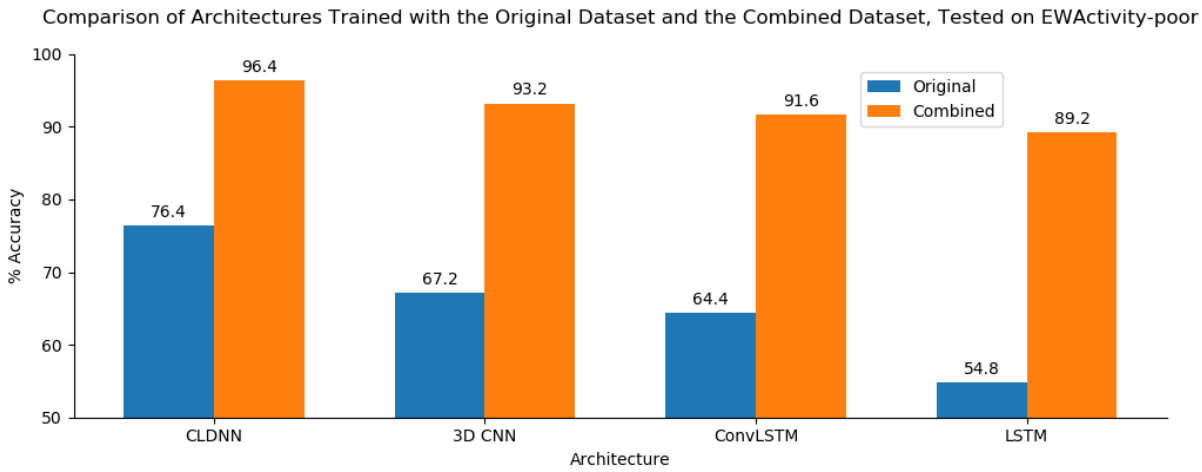

Figure 28. Comparison of architectures trained on the original training set vs. architectures trained on the combined dataset; both are tested on EWActivity-poor dataset.

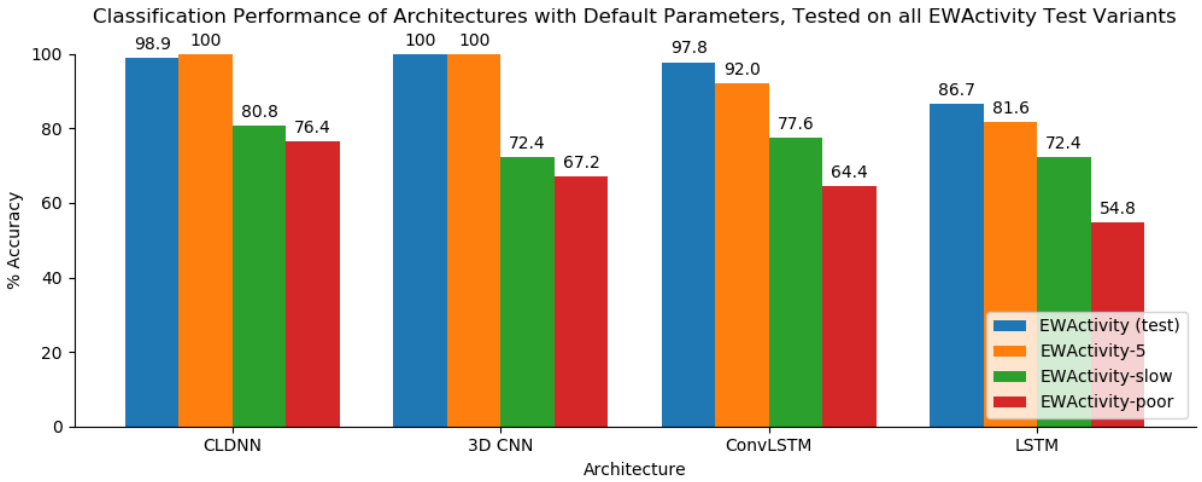

Figure 29. Classification performance of architectures trained on training data that combines data from all EWActivity test variants. Each color represents one of the four EWActivity test variants.

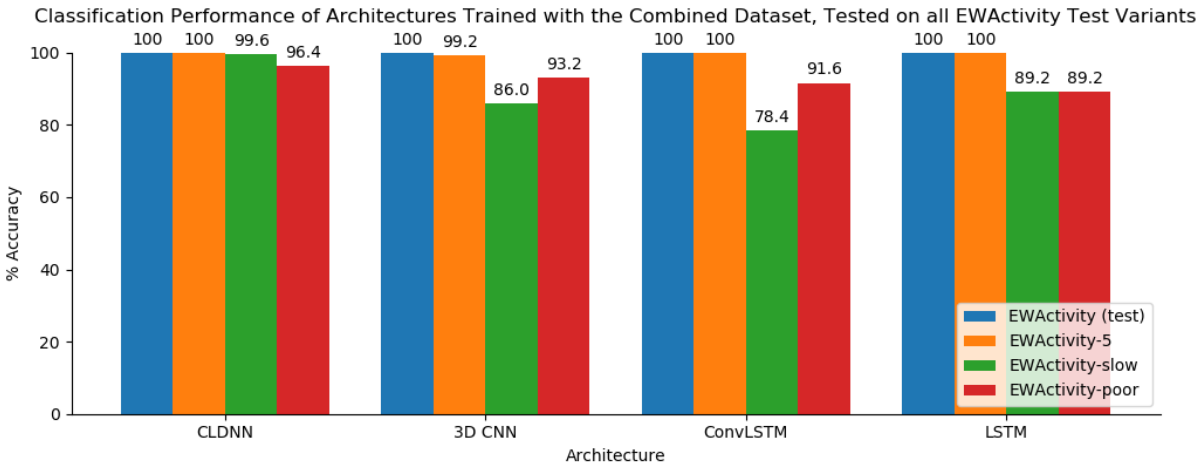

Figure 30. Classification performance of architectures trained on training data that combines data from all EWActivity test variants. Each color represents one of the four EWActivity test variants.

6.6 Other Observations

Sorting by length: A common training practice in deep learning is to shuffle the training dataset after every epoch to regularize the training process. When a sequence-based dataset is used as is the case here, shuffling the training set after every epoch can sometimes lead to very inefficient training if short samples are batched with long samples, as more padding is required.

One solution is to sort the dataset by sequence length initially instead of shuffling after every epoch. However, early experimentation indicated that this had generally little effect on the outcome of training for most models tested here, so epoch shuffling is preferred in favor of the regularization it offers.

Interpreting the Hidden State: For the architectures evaluated here, the output of a recurrent layer is computed by taking the array of hidden state values at all timesteps and finding the maximum value across the time dimension for each individual hidden node. This essentially extracts the strongest feature value from each hidden node throughout the sequence and helps to avoid the issue of weakened features at the end of sequences that are padded with zero frames. Another method that is arguably more common is to simply use the hidden state from the last timestep in the sequence as the output for the recurrent structure. An experiment was performed to identify whether these strategies result in significant differences in performance. Figure 31 compares these two strategies. Architectures labeled with '-last' denote the second strategy described here that differs from the strategy used throughout the other experiments in this thesis. CLDNN did not show a large difference in performance; however, the simple LSTM model improved drastically using the last hidden state strategy instead of the maximal strategy. This demonstrates that these strategies can have large impacts on performance achieved.

Importance of Dropout: All evaluated architectures include dropout to regularize the training process. Dropout simply refers to randomly replacing features (or entire channels in the case of convolution) with a zero value at a fixed probability during training, typically between many layers, which helps to reduce overfitting to the training data. Exact dropout probabilities used are listed in Table 2. Dropout proved to be essential in these experiments due to quantity of training data collected, which is rather low relative to typical deep learning studies. An additional

training experiment was conducted where no dropout was used. This had a measurably negative impact on the performance of the architectures on unseen test sets (**Error! Reference source not found.**) which indicates that dropout is crucial to learning where large volumes of training data are not available.

Interpolation: Convolution requires consistent input size, so the inputs to the 3D CNN architecture are interpolated to precisely 30 frames per sample. This is accomplished dynamically and without cropping by using trilinear interpolation at the input layer (after input dropout). The downside of interpolation is that it cannot be done in real time, unless the designer creates a way to define the start and end point of a sample within an endless sequence. Other architectures are also compatible with interpolated images, but it is not required. Figure 33 shows the performances of architectures that begin by interpolating input sequences to 30 frames. This demonstrates that there is not a major correlation between interpolation and increased performance. Interestingly, ConvLSTM is the exception as it performed considerably better on interpolated sequences than on non-interpolated sequences. Similar comparisons between the other EWActivity test variants can be found in the Appendix.

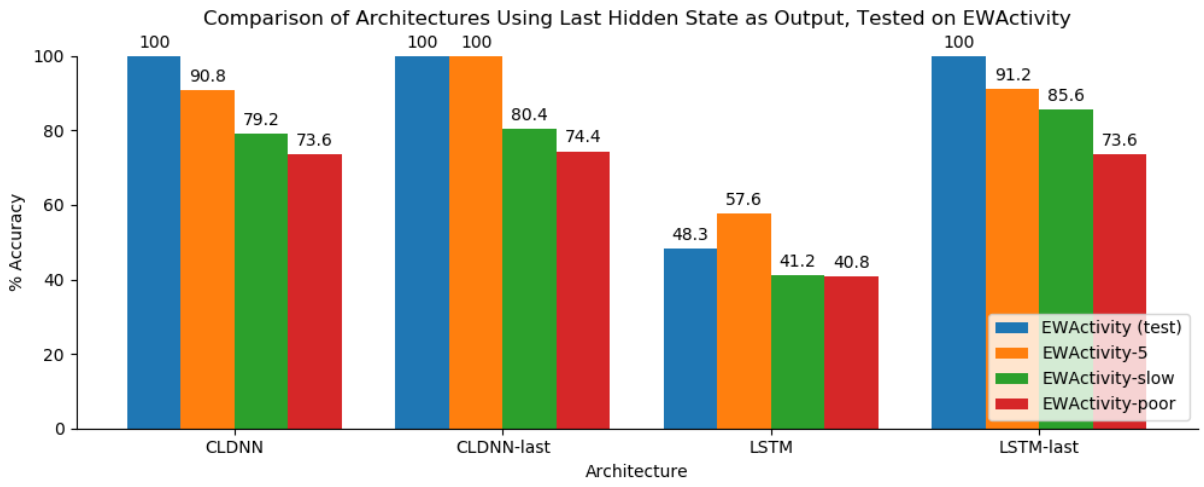

Figure 31. Comparison of architectures that use different output methods at the recurrent layer. Networks that end in '-last' indicate the use of the last hidden state as the recurrent output,

otherwise the approach used is the same as the other experiments. Each color represents one of the four EWActivity test variants.

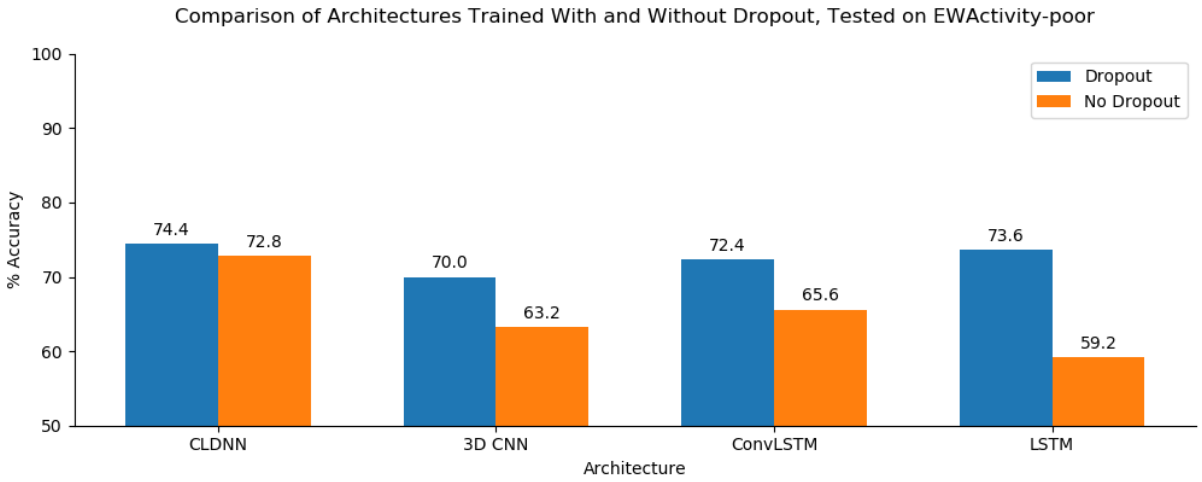

Figure 32. Classification performance of architectures trained with vs. without the use of dropout layers and tested on EWActivity-poor.

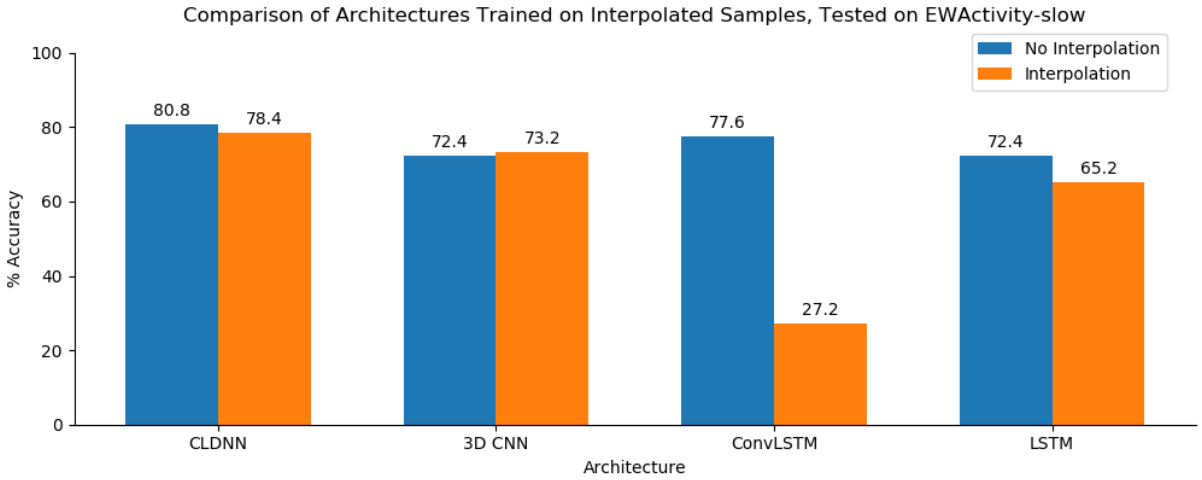

Figure 33. Comparison of architectures trained on interpolated samples and tested on EWActivity-slow.

Chapter 7: Conclusion

7.1 Discussion of Results

While the conclusions reached here can only scientifically be applied to capacitive sensor array-based applications, these conclusions also carry implications about spatiotemporal gesture recognition in other fields such as video, depth-imaging, and different forms of wearable sensors not used here.

The hypothesis that deep convolution combined with LSTM is superior to simple LSTM structures is confirmed by the results shown here. When gestures of a known length are concerned as inputs, 3D convolution excels above other architectures in terms of classification accuracy. However, 3D convolution is only fully possible when a consistently sized input is given, which makes context-awareness more challenging. CLDNN is shown to be sufficient in most classification tasks demonstrated here, and requires much fewer parameters than the other architectures, making it a superior choice for context-aware classification models. Additionally, the performance of CLDNN often seems to improve when the number of hidden nodes is increased.

For most experiments, ConvLSTM fell short in terms of performance. This is likely due to the fact that ConvLSTM was originally developed for sequence prediction tasks rather than classification. ConvLSTM sometimes requires a separate prediction-encoding network to be viable, which is possibly the case here. In any case, it is not abundantly clear why ConvLSTM did so poorly when it has been shown to work well in other applications. However, a few specific experiments done in this study resulted in excellent performance from ConvLSTM, which indicates that it is not inherently unsuitable for this task. Among other things, future work

includes analyzing the ConvLSTM architecture more closely with regards to low-resolution CSA applications, in order to identify what hyperparameters do lead to success and why the architecture is so prone to poor performance in other designs.

ReLU and Tanh both performed well as activation functions, but in many cases, one led to much better results than the other. Further study is required to precisely identify in what scenarios one should be used over the other. Sigmoid, on the other hand, is shown to be a decidedly poor activation function that should not be used for applications related to those that are evaluated here.

7.1.1 Design Process for Motor Rehabilitation

CLDNN and 3D CNN both proved to be successful architectures for most forms of gesture recognition demonstrated here. However, the advantage of CLDNN is that it includes a recurrent framework with two LSTM layers that are capable of remembering information over longer time sequences. With EWActivity-slow as the benchmark, CLDNN typically outperforms 3D CNN in the CSA environment. Additionally, 3D CNN requires a fixed temporal size in order for the input to fit the 3D model. This means that interpolation of some sort is required on sequences of varying length; otherwise, there must be a method for aligning the prediction process with the start of a gesture so that the exact number of frames required can be recorded. This could also be approached with a multi-network system that continuously makes predictions about each consecutive groups of frames with fixed length, using 3D convolution, and including another network that analyzes the stream of predictions to find the current or most recently formed prediction for a sequence of time. Regardless, this is a difficult task for systems that are meant to be context-aware and capable of classifying inputs at any time. This means that there is a decision for the designer to make. If the actions are repetitive enough such that it is easy to

identify when gestures start and when they begin at all times, and if it is assumed that the gesture will typically be performed with a degree of completion or at least the intent to succeed, then 3D CNNs may be the best approach for learning all gestures. Otherwise, if the system requires that a network can control its memory to learn in dynamic contexts, then CLDNN is shown to be the most reliable model evaluated here.

7.1.2 High Classification Accuracy

All architectures performed better on the more-difficult EWActivity test variants when they were supplied with training data that included data from those test variants. This was especially true for EWActivity-poor, where incomplete and partially incorrect gestures were included. The reason for this is simple: rather than learning the exact sequence of a perfect gesture, a network exposed to imperfect samples will better learn the general characteristics of that gesture and will be more capable of recognizing the gesture even when some human error is introduced. Clearly, there is a strong advantage in correctly classifying all kinds of gestures when a wider variety of training data is supplied to the network. This is very useful for applications which require a high classification accuracy such as home automation, adaptive gaming, or other context-aware systems. These systems rely on minimal mistakes to function well, so when training architectures used for those applications, it is important to include a dynamic variety of training data.

7.1.3 Personalization

3D CNN and CLDNN were also shown to be extremely capable of differentiating between different subjects and classifying their mannerisms when performing gestures. This knowledge could be very useful for applications that aim to make decisions about context based

on who is using them. This might also be a step towards identifying the quality of a gesture and whether it was performed adequately or not, by comparing the confidence of the network's prediction with the full confidence of the original prediction on a perfect-classified training sample performed by the same subject. Similarly, ConvLSTM was shown to fall short in this task.

7.2 Future Work

This thesis was conceived from questions raised while investigating the fields of deep learning and gesture recognition. Appropriately, this thesis also raised questions of its own during experimentation, not of all of which can be answered within a single investigation. Therefore, future work will be dedicated to the topics listed in this section.

7.2.1 Inference on MCU

The experiments shown here focus on training networks with a large set of gestures, and then testing the performance of the networks on different, previously unseen gestures from the same dataset. All of this is done within an Anaconda environment, on a PC, with the help of machine learning libraries like PyTorch. While this is useful for deep learning experimentation and prototyping, this approach fails to test the real-world application, which would involve inference on a microcontroller unit.

Inference refers to a machine recording or receiving inputs coming directly from the environment, in real time, and making a prediction or classification based on the learning model that it is equipped with. For example, if a designer wants to implement a custom gesture recognition application in a patient's home, they must implement the chosen network architecture directly on an MCU. This is usually done by directly programming the MCU to

compute all operations required by the network's forward pass, then loading all pre-trained network parameters to the MCU's memory. The process of programming a network's forward pass on an MCU can be quite complex and requires confident knowledge of how every layer of the network is mathematically represented. This is because the machine learning libraries used in development here are not directly compatible with the processors found on most microcontrollers. That being said, the chosen MCU must be versatile and directly programmable, such as one that is equipped with an ARM-processor (e.g., an Arduino Uno), unlike the MCU used in the capacitive sensing applications presented here. Note that the MCU must also be configured to record environmental inputs and translate them to a state that is compatible with the specific network in real time. Finally, the designer must incorporate some sort of user interface (UI) that is human readable, to communicate what the network's inference prediction is.

Generally, inference applications are designed to run continuously, and therefore must also be context-aware, or at least have some method of determining where the start of a gesture is, and when to make a prediction. This is extremely important for evaluating a network's real-life applicability to problems like motor rehabilitation and is considered the direct next step of this work.

7.2.2 Performance Feedback System

Where motor rehabilitation is considered, it is important to identify the accuracy of a gesture and quantify how well the gesture was performed. One of the target applications of this work is a system that can give feedback to the patient on how closely their gesture matched the target gesture, in real time. Additionally, a UI integration to the CSA sensor might be able to

highlight specific areas where the gesture was less accurate. This kind of application would likely be trained on perfect samples as most networks were trained here.

7.2.3 Real Patients

The subjects that performed gestures as a part of the experiments shown here do not have any known form of motor impairment, neurological or otherwise. Therefore, it would be much more useful to repeat these experiments with patients suffering from loss of motor function due to different injuries. In that setting, the "bad" gestures that were artificially created by healthy subjects would be replaced with gestures performed by real patients, and the "good" gestures would still be performed by health subjects as a control group or target.

Additionally, one of the fundamental requirements of deep learning is the huge amount of data that is needed to train a network. While any amount of data can train a network to perform some task, network architectures perform the best in real-world applications when they are trained on a sufficiently large dataset. This makes the network more reliable and results in fewer mistakes in the applications it is designed for by introducing more variance to the training process and reducing the effect of overfitting. Unfortunately, a large setback of this work is the lack of access to a more significant number of subjects for collecting data, due to a pandemic surrounding the time frame of development. In the future, it will be invaluable to collect as much data as possible for training the architectures that will see deployment to real applications.

7.2.4 Wearable CSAs

While the MCU used in these experiments is large, the MSP430FR2676 chip can also be installed on a micro 48x37mm MCU. Furthermore, this chip can be installed on a flexible version of the micro MCU board with only the individual components that are needed. Using

small ribbon cable ports and micro resistors, these boards retain the same parallel-scanning ability and the same number of supported electrodes as the MCU used here. This, combined with prior work our lab has done on soft robotics and wearable sensors (Section 3.1.4), implies that the conclusions made in this thesis may translate well to wearable micro-CSAs that can be worn in clothing. It is very likely the results would differ somewhat with new, flexible sensors in a different application environment, and thus would be worth investigating.

7.2.5 LSTM Peephole Connections

Equation 12 shows the modified LSTM equations when peephole connections are incorporated using the previous cell state vector [83]. One goal of this work that was not achieved was to determine if peephole connections can be useful for CSAs to distinguish between gestures performed at different speeds, which would be very useful for motor rehabilitation.

$$\begin{aligned} i_t &= \sigma(W_{xi}X_t + W_{hi}h_{t-1} + W_{ci} \circ c_{t-1} + b_i) \\ f_t &= \sigma(W_{xf}X_t + W_{hf}h_{t-1} + W_{cf} \circ c_{t-1} + b_f) \\ g_t &= \tanh(W_{xg}X_t + W_{hg}h_{t-1} + b_g) \\ c_t &= f_t \circ c_{t-1} + i_t \circ g_t \\ o_t &= \sigma(W_{xo}X_t + W_{ho}h_{t-1} + W_{co} \circ c_{t-1} + b_o) \\ h_t &= o_t \circ \tanh(c_t) \end{aligned} \tag{12}$$

7.3 Personal Contributions

All code used was written by me except for code generated by CapTIvate Design Center (Section 4.3.1), and a PyTorch implementation of ConvLSTM forked from GitHub (Section 5.4.2) [78]. This includes signal processing, dataset generation, dataset manipulation, PyTorch interpretation, training procedures, testbenches, and customized PyTorch implementations for all

other architectures. All sensor designs were built by me, with the exception of the CSA used in the activity board, which was built by a team member in [57].

References

[1] KD Kochanek, S Murphy, J Xu, and E Arias, "Mortality in the United States, 2016," *NCHD Data Brief,* vol. 293, 2017.

[2] D. Mozzafarian, E.J. Benjamin, A.S. Go, D.K. Arnett, et al., "Heart disease and stroke statistics," *Circulation,* vol. 133*,* no. 4, pp. 38-360, 2016.

[3] Nancy Byl, Jennifer Roderick, Olfat Mohamed, Monica Hanny, et al., "Effectiveness of Sensory and Motor Rehabilitation of the Upper Limb Following the Principles of Neuroplasticity: Patients Stable Poststroke," *Neurorehabilitation and Neural Repair,* vol. 17*,* no. 3, pp. 176-191, 2003.

[4] I. Aprile, M. Rabuffetti, L. Padua, E. Di Sipio, C. Simbolotti, and M. Ferrarin, "Kinematic analysis of the upper limb motor strategies in stroke patients as a tool towards advanced neurorehabilitation strategies: a preliminary study," *Biomed Res Int,* vol. 2014, pp. 636123, 2014.

[5] S. McCombe Waller, and J. Whitall, "Bilateral arm training: why and who benefits?," *NeuroRehabilitation,* vol. 23*,* no. 1, pp. 29-41, 2008.

[6] F. Cincotti, F. Pichiorri, P. Aricò, F. Aloise, et al., "EEG-based Brain-Computer Interface to support post-stroke motor rehabilitation of the upper limb," *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 4112-4115, 2012.

[7] L. E. Sucar, F. Orihuela-Espina, R. L. Velazquez, D. J. Reinkensmeyer, R. Leder, and J. Hernández-Franco, "Gesture Therapy: An Upper Limb Virtual Reality-Based Motor Rehabilitation Platform," *IEEE Transactions on Neural Systems and Rehabilitation Engineering,* vol. 22*,* no. 3, pp. 634-643, 2014.

[8] Eleftheria Vaportzis, Maria Giatsi Clausen, and Alan J. Gow, "Older Adults Perceptions of Technology and Barriers to Interacting with Tablet Computers: A Focus Group Study," *Frontiers in psychology,* vol. 8, pp. 1687-1687, 2017.

[9] Bilal Ghazal, and Khaled Khatib, "Smart Home Automation System for Elderly, and Handicapped People using XBee," *International Journal of Smart Home,* vol. 9, pp. 203-210, 2015.

[10] M. Valles, F. Manso, M. T. Arredondo, and F. del Pozo, "Multimodal environmental control system for elderly and disabled people," *Proceedings of 18th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 516-517 vol.512, 1996.

[11] Jungyoon Kim, Bo Ra Kim, Ilhyang Kang, Myung-Eun Kye, et al., "Using a Tablet Device to Compensate for Underestimation of Cognitive Function due to Impaired Dominant Hand

Function in Stroke Patients," *International Journal of Human‐Computer Interaction,* vol. 33*,* no. 5, pp. 423-427, 2017.

[12] Michelle Annett, Fraser Anderson, Darrell Goertzen, Jonathan Halton, et al., "Using a multi-touch tabletop for upper extremity motor rehabilitation," *Proceedings of the 21st Annual Conference of the Australian Computer-Human Interaction Special Interest Group: Design: Open 24/7*, Melbourne, Australia, 2009.

[13] Wei Niu, Xiaolong Ma, Yanzhi Wang, and Bin Ren, "26ms inference time for resnet-50: Towards real-time execution of all dnns on smartphone," *arXiv preprint arXiv:1905.00571*, 2019.

[14] Erico Tjoa, and Cuntai Guan, "A survey on explainable artificial intelligence (xai): Towards medical xai," *arXiv preprint arXiv:1907.07374*, 2019.

[15] Larry K. Baxter, "Capacitve Sensors: Design and Applications." *John Wiley & Sons*, 1996.

[16] Andreas Holzinger, "Finger Instead of Mouse: Touch Screens as a Means of Enhancing Universal Access," *Universal Access Theoretical Perspectives, Practice, and Experience*, Berlin, Heidelberg, pp. 387-397, 2003.

[17] Ronald Peter Binstead, United States, Patent No. EP0185671B1. U. S. P. a. T. Office, 1989.

[18] Joel Kent, "Touchscreen technology basics & a dew development," *CMOS Emerging Technologies Conference*, pp. 1-13, 2010.

[19] Mayank Shadwani, Shivani Sachan, and Palak Sachan, "Capacitive Sensing & Its Applications." 2018.

[20] Texas Instruments, "Capacitive Sensing Basics - Captivate Technology Guide, 2017 [Revised Aug. 2019]. Available: http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/CapTIvate_Design_Center/latest/exports/docs/users_guide/html/CapTIvate_Technology_Guide_html/markdown/ch_basics.html

[21] Kent Chamberlin, Wayne Smith, Christopher Chirgwin, Seshank Appasani, and Paul Rioux, "Analysis of the charge exchange between the human body and ground: evaluation of "earthing" from an electrical perspective," *Journal of chiropractic medicine,* vol. 13*,* no. 4, pp. 239-246, 2014.

[22] John Carey, "Getting in touch with capacitance sensor algorithms, 2009. Available: https://www.embedded.com/getting-in-touch-with-capacitance-sensor-algorithms/

[23] J. Lee, M. T. Cole, J. C. S. Lai, and A. Nathan, "An Analysis of Electrode Patterns in Capacitive Touch Screen Panels," *Journal of Display Technology,* vol. 10*,* no. 5, pp. 362-366, 2014.

[24] Seung Hwan Lee, Jae Sung An, Seong Kwan Hong, and Oh Kyong Kwon, "A highly linear and accurate fork-shaped electrode pattern for large-sized capacitive touch screen panels," *IEEE Sensors Journal,* vol. 18*,* no. 15, pp. 6345-6351, 2018.

[25] B. Cannon, and C. Brennan, "Electrostatic simulation methodology for capacitive touch-screen panels," *25th IET Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014)*, pp. 216-220, 2014.

[26] Carl Friedrich Gauss, and G. W. Stewart, "Theory of the Combination of Observations Least Subject to Errors." *Society for Industrial and Applied Mathematics*, 1995.

[27] Warren S. McCulloch, and Walter Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics,* vol. 5*,* no. 4, pp. 115-133, 1943.

[28] Frederic Gruau, Darrell Whitley, and Larry Pyeatt, "A comparison between cellular encoding and direct encoding for genetic neural networks, pp. 81-89, 1996.

[29] Yiming Jiang, Chenguang Yang, Jing Na, Guang Li, Yanan Li, and Junpei Zhong, "A Brief Review of Neural Networks Based Learning and Control and Their Applications for Robots," *Complexity*, 2017.

[30] Frank Rosenblatt, "The Perceptron: a Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review,* vol. 65*,* no. 6, pp. 386-408, 1958.

[31] Frank Rosenblatt, "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms." New York: *Spartan*, 1962.

[32] Bernard Widrow, and Marcian Hoff, "Associative Storage and Retrieval of Digital Information in Networks of Adaptive "Neurons" (pp. 160-160). 1962.

[33] A. L. Hodgkin, and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of physiology,* vol. 117*,* no. 4, pp. 500-544, 1952.

[34] Vinod Nair, and Geoffrey E. Hinton, "Rectified linear units improve restricted boltzmann machines," *Proceedings of the 27th International Conference on International Conference on Machine Learning*, Haifa, Israel, 2010.

[35] Jun Han, and Claudio Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," *International Workshop on Artificial Neural Networks*, pp. 195-201, 1995.

[36] Barry L Kalman, and Stan C Kwasny, "Why tanh: choosing a sigmoidal function," *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, pp. 578-581, 1992.

[37] Sebastian Raschka, "Perceptron," *mlxtend*, 2014 [Revised 2019]. Available: http://rasbt.github.io/mlxtend/user_guide/classifier/Perceptron/

[38] Stephen Judd, "On the complexity of loading shallow neural networks," *Journal of Complexity,* vol. 4*,* no. 3, pp. 177-192, 1988.

[39] A. G. Ivakhnenko, "Polynomial Theory of Complex Systems," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. SMC-1*,* no. 4, pp. 364-378, 1971.

[40] Eric B. Baum, and Frank Wilczek, "Supervised learning of probability distributions by neural networks, pp. 52-61, 1988.

[41] Paul J. Werbos, "Applications of advances in nonlinear sensitivity analysis, Berlin, Heidelberg, pp. 762-770, 1982.

[42] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. (1985). Learning internal representations by error propagation: California Univ San Diego La Jolla Inst for Cognitive Science.

[43] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86*,* no. 11, pp. 2278-2324, 1998.

[44] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, "U-net: Convolutional networks for biomedical image segmentation," *International Conference on Medical image computing and computer-assisted intervention*, pp. 234-241, 2015.

[45] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger, "Densely connected convolutional networks," *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700-4708, 2017.

[46] Michael Mozer, "A Focused Backpropagation Algorithm for Temporal Pattern Recognition," *Complex Systems,* vol. 3, 1995.

[47] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur, " Recurrent neural network based language model," *Eleventh annual conference of the international speech communication association*, 2010.

[48] Mahendran Venkatachalam, "Recurrent Neural Networks, 2019. Available: https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce

[49] Sepp Hochreiter, and Jürgen Schmidhuber, "Long Short-Term Memory," *Neural Computation,* vol. 9*,* no. 8, pp. 1735-1780, 1997.

[50] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: continual prediction with LSTM," *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, pp. 850-855 vol.852, 1999.

[51] Felix A. Gers, Nicol N. Schraudolph, and Jürgen Schmidhuber, "Learning precise timing with lstm recurrent networks," *J. Mach. Learn. Res.,* vol. 3*,* no. null, pp. 115–143, 2003.

[52] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[53] Nicolas Ballas, Li Yao, Chris Pal, and Aaron Courville, "Delving deeper into convolutional networks for learning video representations," *arXiv preprint arXiv:1511.06432*, 2015.

[54] J. Donahue, L. A. Hendricks, M. Rohrbach, S. Venugopalan, et al., "Long-Term Recurrent Convolutional Networks for Visual Recognition and Description," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 39*,* no. 4, pp. 677-691, 2017.

[55] Karen Simonyan, and Andrew Zisserman, "Two-stream convolutional networks for action recognition in videos," *Advances in neural information processing systems*, pp. 568-576, 2014.

[56] H. Liu, J. P. Parkerson, and A. Nelson, "Connected Capacitive Sensor Array for Upper-Extremity Motor Rehabilitation," *2018 IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, pp. 17-18, 2018.

[57] H. Liu, E. Sanchez, J. Parkerson, and A. Nelson, "Gesture Classification with Low-Cost Capacitive Sensor Array for Upper Extremity Rehabilitation," *2019 IEEE SENSORS*, pp. 1-4, 2019.

[58] P. Molchanov, S. Gupta, K. Kim, and J. Kautz, "Hand gesture recognition with 3D convolutional neural networks," *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1-7, 2015.

[59] G. Zhu, L. Zhang, P. Shen, and J. Song, "Multimodal Gesture Recognition Using 3-D Convolution and Convolutional LSTM," *IEEE Access,* vol. 5, pp. 4517-4524, 2017.

[60] L. Zhang, G. Zhu, P. Shen, J. Song, S. A. Shah, and M. Bennamoun, "Learning Spatiotemporal Features Using 3DCNN and Convolutional LSTM for Gesture Recognition," *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pp. 3120-3128, 2017.

[61] P. Molchanov, X. Yang, S. Gupta, K. Kim, S. Tyree, and J. Kautz, "Online Detection and Classification of Dynamic Hand Gestures with Recurrent 3D Convolutional Neural Networks," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4207-4215, 2016.

[62] A. Nelson, J. Schmandt, P. Shyamkumar, W. Wilkins, et al., "Wearable multi-sensor gesture recognition for paralysis patients," *SENSORS, 2013 IEEE*, pp. 1-4, 2013.

[63] G. Singh, A. Nelson, S. Lu, R. Robucci, C. Patel, and N. Banerjee, "Event-Driven Low-Power Gesture Recognition Using Differential Capacitance," *IEEE Sensors Journal,* vol. 16*,* no. 12, pp. 4955-4967, 2016.

[64] A. Nelson, G. Singh, R. Robucci, C. Patel, and N. Banerjee, "Adaptive and Personalized Gesture Recognition Using Textile Capacitive Sensor Arrays," *IEEE Transactions on Multi-Scale Computing Systems,* vol. 1*,* no. 2, pp. 62-75, 2015.

[65] Alex Nelson Enrique Sanchez, "Gaming Adaptive Sensing Pads," *In Preparation…*, 2020.

[66] Andreas Frutiger, Joseph T. Muth, Daniel M. Vogt, Yiğit Mengüç, et al., "Capacitive soft strain sensors via multicore‐shell fiber printing," *Advanced Materials,* vol. 27*,* no. 15, pp. 2440-2446, 2015.

[67] Alexander Nelson, Sandy McCombe Waller, Ryan Robucci, Chintan Patel, and Nilanjan Banerjee, "Evaluating touchless capacitive gesture recognition as an assistive device for upper extremity mobility impairment," *Journal of Rehabilitation and Assistive Technologies Engineering,* vol. 5, pp. 2055668318762063, 2018.

[68] Texas Instruments, "MSP430FR267x Capacitive Touch Sensing Mixed-Signal Microcontrollers," 2019 [Revised 2020]. [Online] Available: http://www.ti.com/lit/ds/symlink/msp430fr2676.pdf

[69] Texas Instruments, "CAPTIVATE-PGMR - Captivate Technology Guide, 2017 [Revised Aug. 2019]. Available: http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/CapTIvate_Design_Center/latest/exports/docs/users_guide/html/CapTIvate_Technology_Guide_html/markdown/ch_evm_CAPT_PGMR.html

[70] Texas Instruments, "Technology - Captivate Technology Guide, 2017 [Revised Aug. 2019]. Available: http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/CapTIvate_Design_Center/latest/exports/docs/users_guide/html/CapTIvate_Technology_Guide_html/markdown/ch_technology.html

[71] Texas Instruments, "Design Guide - Captivate Technology Guide, 2017 [Revised Aug. 2019]. Available: http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/CapTIvate_Design_Center/latest/exports/docs/users_guide/html/CapTIvate_Technology_Guide_html/markdown/ch_design_guide.html

[72] PyTorch, "torch.utils.data - PyTorch master documentation, 2020. Available: https://pytorch.org/docs/stable/data.html#torch.utils.data.Dataset

[73] PyTorch, "torch.nn - PyTorch master documentation, 2020. Available: https://pytorch.org/docs/stable/nn.html

[74] L. F. Aguiar, and Bó A. P. L, "Hand gestures recognition using electromyography for bilateral upper limb rehabilitation," *2017 IEEE Life Sciences Conference (LSC)*, pp. 63-66, 2017.

[75] Jacob O. Wobbrock, (2006) "Edgewrite: a versatile design for text entry and control," Carnegie Mellon University.

[76] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks," *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4580-4584, 2015.

[77] SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," *Advances in neural information processing systems*, pp. 802-810, 2015.

[78] Andrea Palazzi, "Implementation of Convolutional LSTM in PyTorch," 2017 [Last updated Mar. 24th, 2020], GitHub. Available: https://github.com/ndrplz/ConvLSTM_pytorch

[79] S. Ji, W. Xu, M. Yang, and K. Yu, "3D Convolutional Neural Networks for Human Action Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 35*,* no. 1, pp. 221-231, 2013.

[80] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri, " Learning spatiotemporal features with 3d convolutional networks," *Proceedings of the IEEE international conference on computer vision*, pp. 4489-4497, 2015.

[81] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton, "On the importance of initialization and momentum in deep learning," *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, Atlanta, GA, USA, 2013.

[82] Ilya Loshchilov, and Frank Hutter, "Sgdr: Stochastic gradient descent with warm restarts, " *arXiv preprint arXiv:1608.03983*, 2016.

[83] Alex Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

Appendix

Table 4. Full names of algorithms and networks referred to by acronym.

Acronym	Algorithm
HMM	Hidden Markov Model
DTW	Dynamic Time Warping
NNC	Nearest Neighbor Classifier
DTC	Decision Tree Classifier
NBC	Naïve Bayes Classifier
CLDNN	Convolutional, Long Short-Term Memory, Deep Neural Network
ConvLSTM	Convolutional Long Short-Term Memory Unit
3D CNN	Three-Dimensional Convolutional Neural Network
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
RNN	Recurrent Neural Network

Table 5: MultiTouch gestures

Gesture	Class Name
Pinch inward (2 fingers)	pinch-2
Pinch inward (5 fingers)	pinch-5
Spread outward (2 fingers)	spread-2
Spread outward (5 fingers)	spread-5
Swipe up (2 fingers)	swipe-2
Swipe up (5 fingers)	swipe-5
Spiral (1 finger, clockwise)	spiral-cw
Spiral (1 finger, counterclockwise)	spiral-ccw

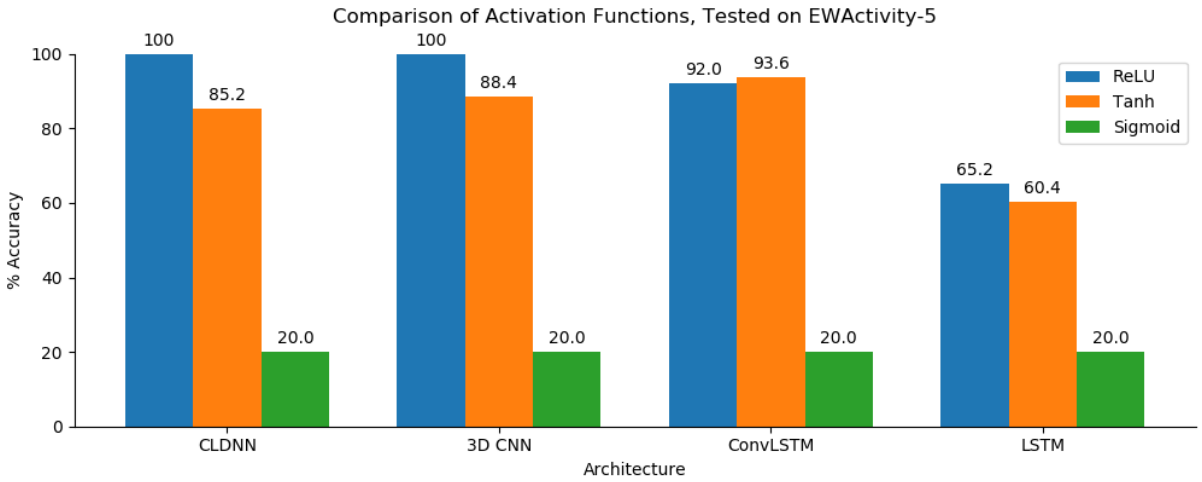

Figure 34. Comparison of activation functions tested on the EWActivity-5 test variant. Each color represents one of the three activation functions evaluated.

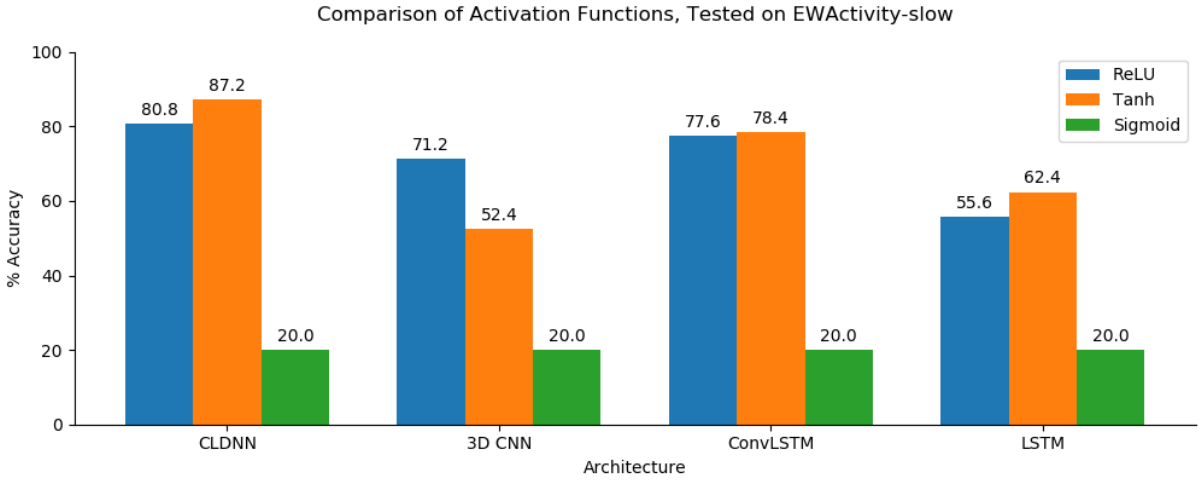

Figure 35. Comparison of activation functions tested on the EWActivity-slow test variant. Each color represents one of the three activation functions evaluated.

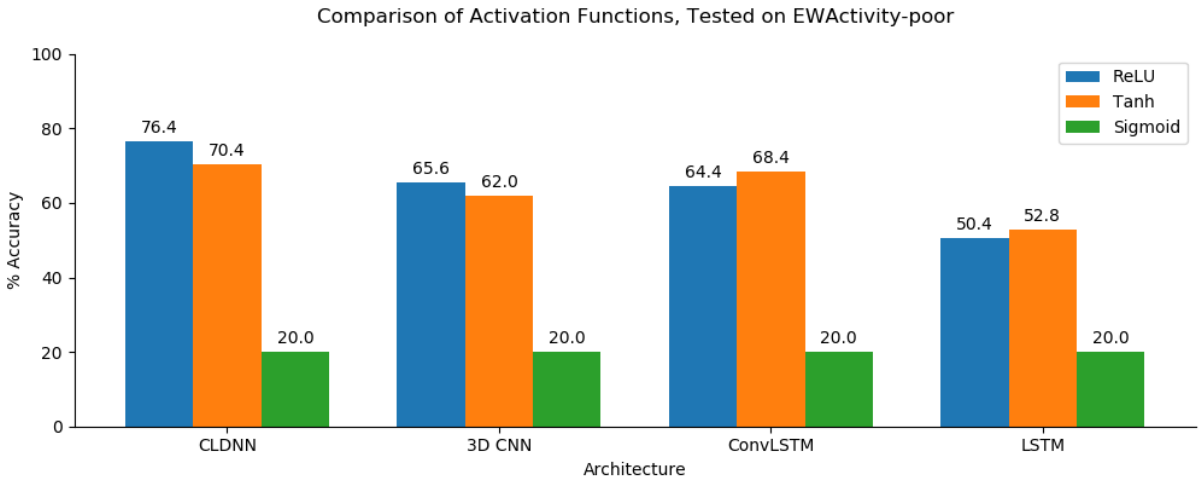

Figure 36. Comparison of activation functions tested on the EWActivity-poor test variant. Each color represents one of the three activation functions evaluated.

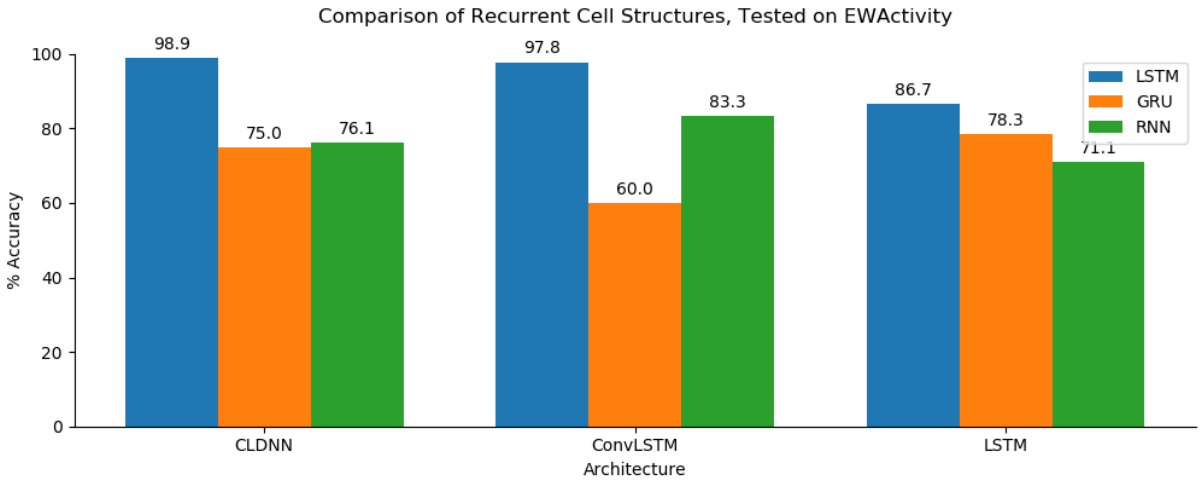

Figure 37. Comparison of recurrent cell structures tested on the EWActivity test set. Each color represents one of the three recurrent cell structures evaluated.

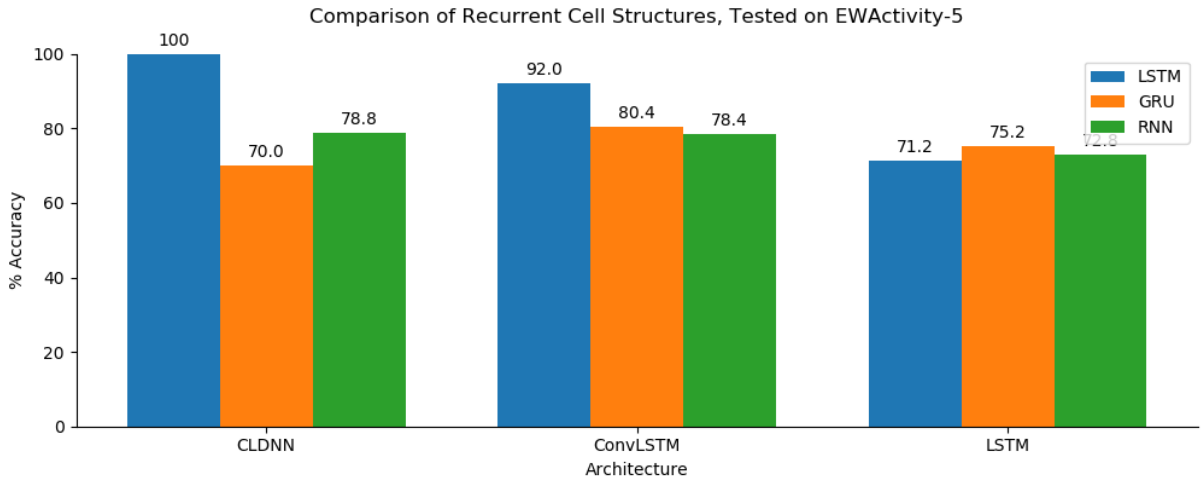

Figure 38. Comparison of recurrent cell structures tested on the EWActivity-5 test variant. Each color represents one of the three recurrent cell structures evaluated.

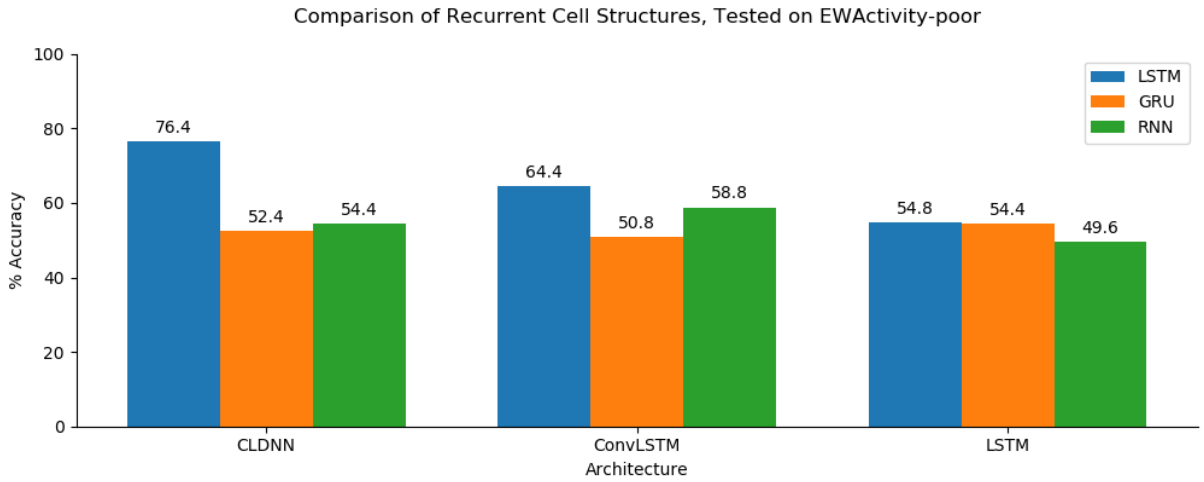

Figure 39. Comparison of recurrent cell structures tested on the EWActivity-poor test variant. Each color represents one of the three recurrent cell structures evaluated.

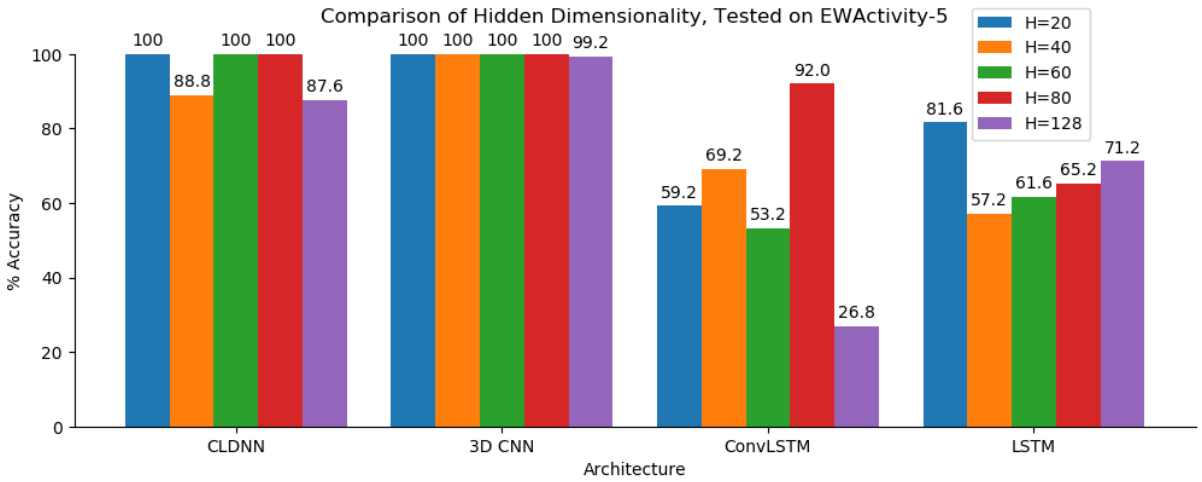

Figure 40. Comparison of hidden dimensionality tested on the EWActivity-5 test variant. Each color represents the number of hidden nodes in the recurrent layers.

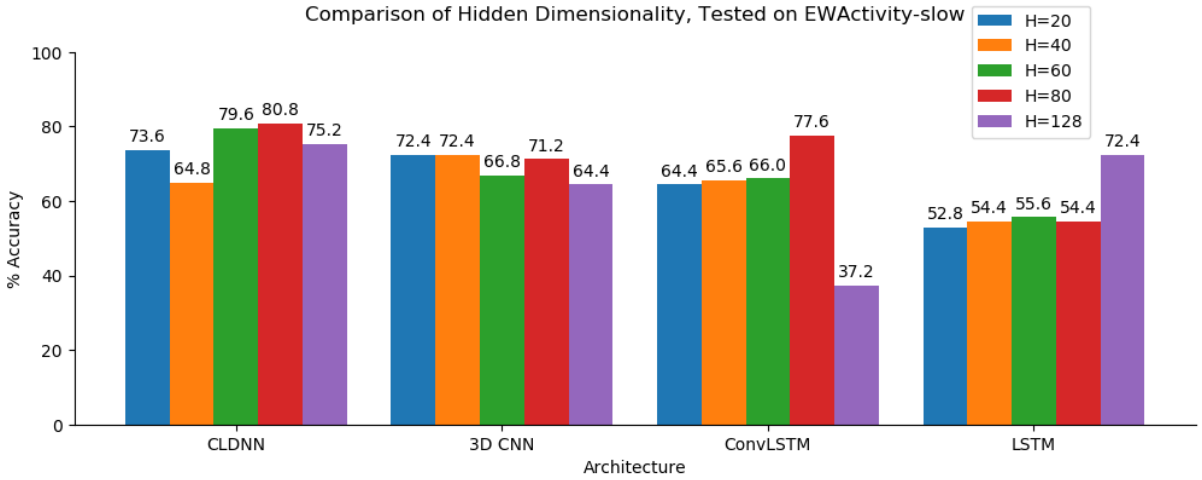

Figure 41. Comparison of hidden dimensionality tested on the EWActivity-slow test variant. Each color represents the number of hidden nodes in the recurrent layers.

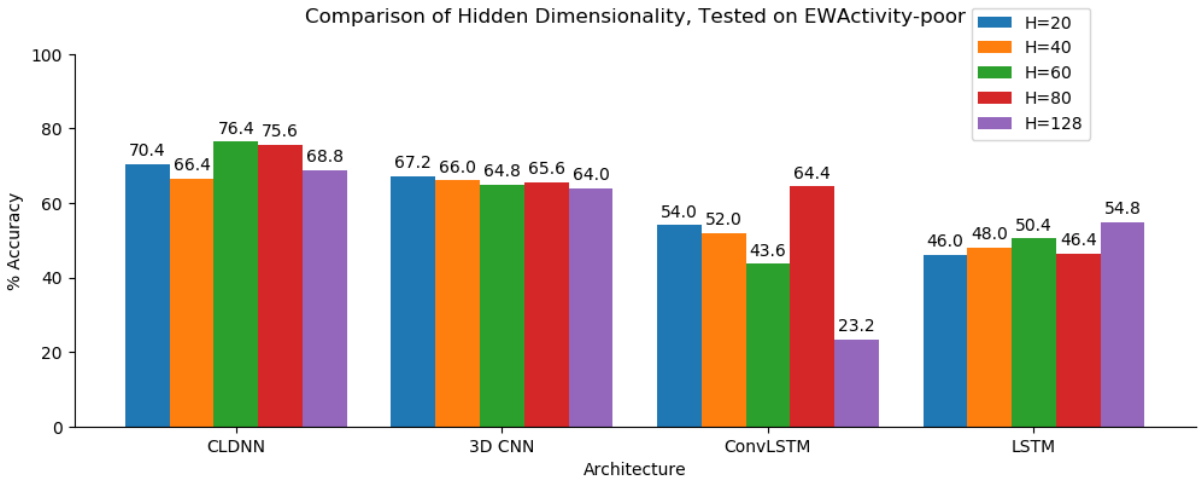

Figure 42. Comparison of hidden dimensionality tested on the EWActivity-poor test variant. Each color represents the number of hidden nodes in the recurrent layers.

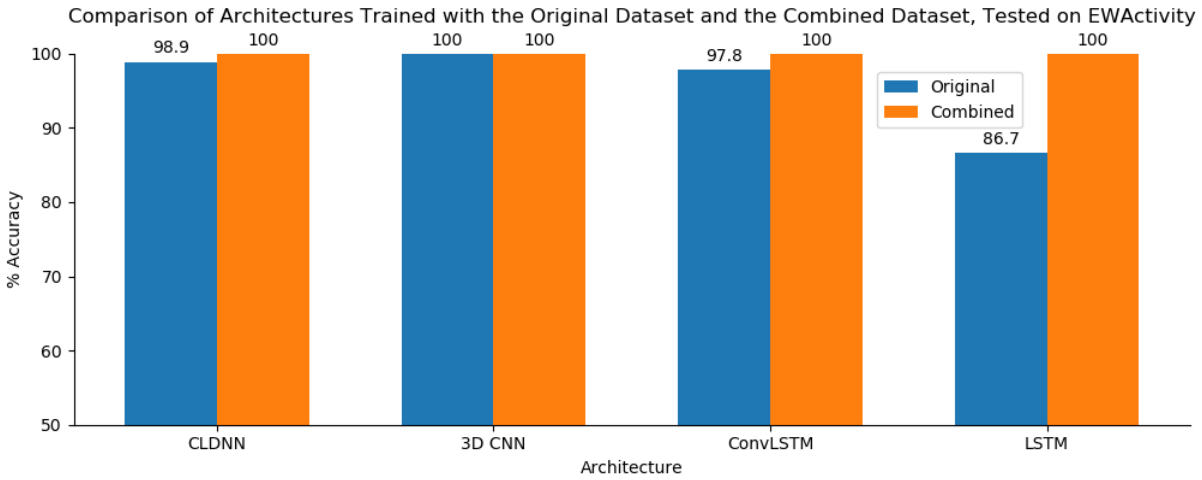

Figure 43. Comparison of architectures trained on the original training set vs. architectures trained on the combined dataset; both are tested on EWActivity test set.

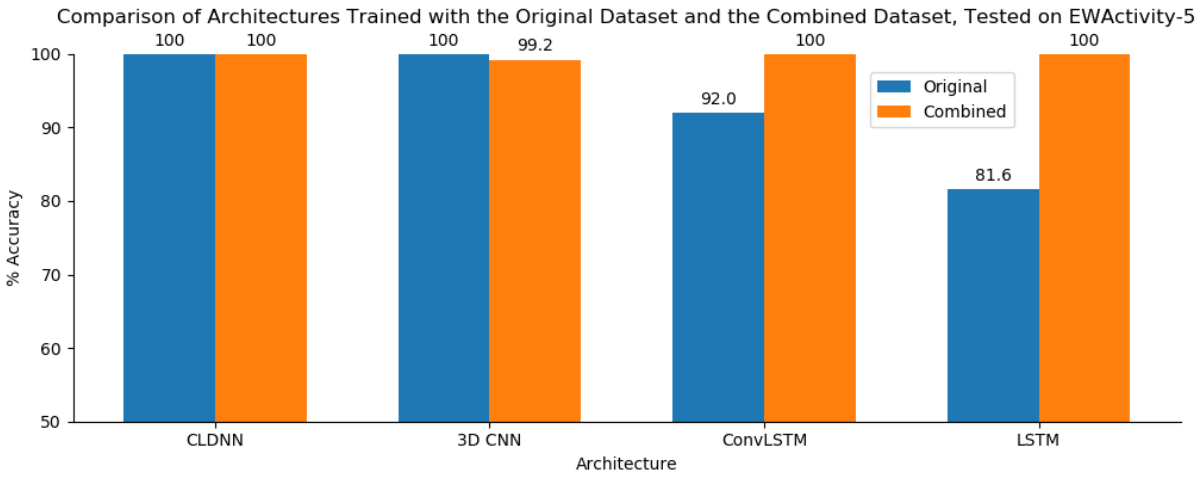

Figure 44. Comparison of architectures trained on the original training set vs. architectures trained on the combined dataset; both are tested on EWActivity-5 test variant.

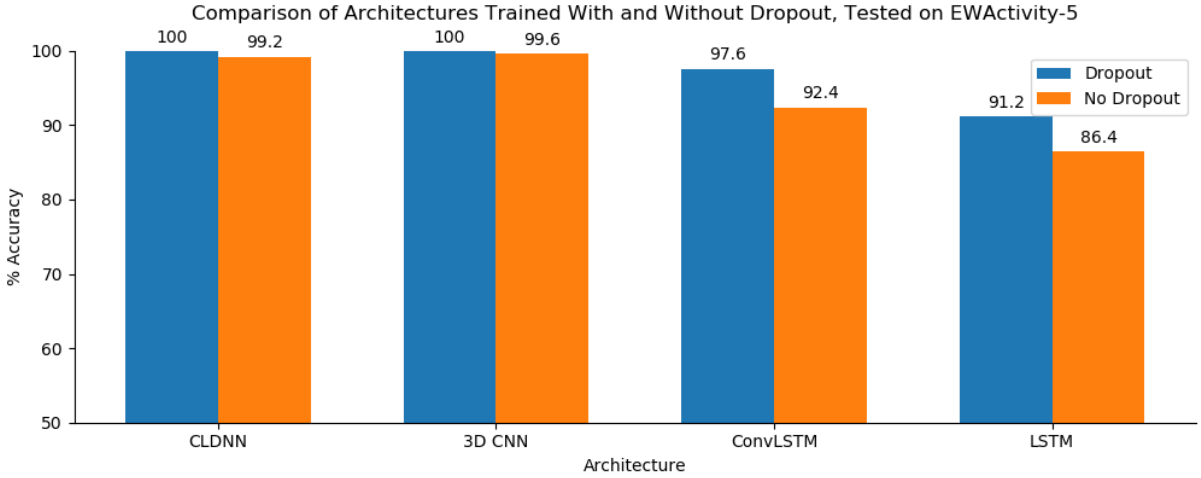

Figure 45. Classification performance of architectures trained with vs. without the use of dropout layers and tested on the EWActivity-5 test variant.

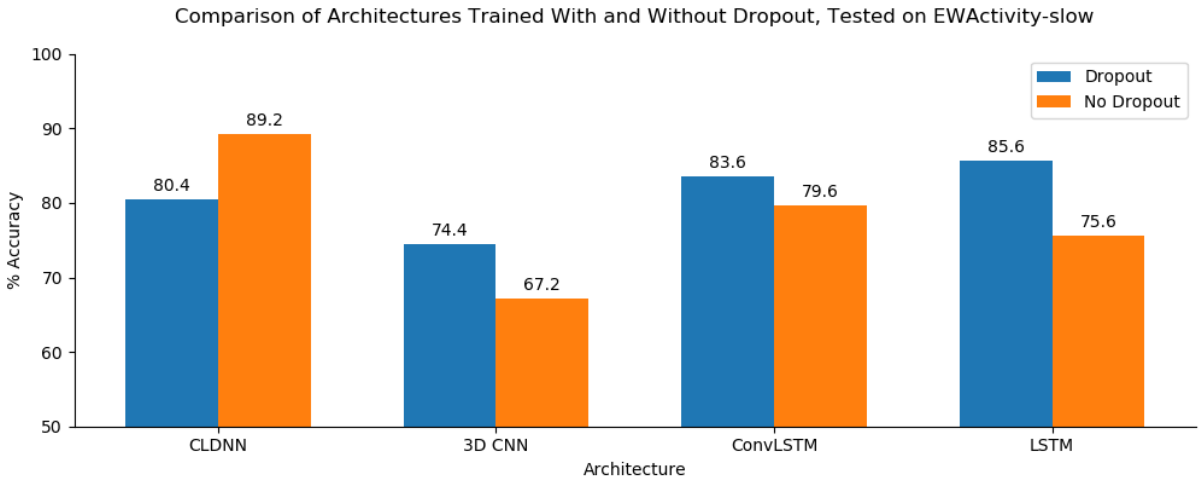

Figure 46. Classification performance of architectures trained with vs. without the use of dropout layers and tested on the EWActivity-slow test variant.

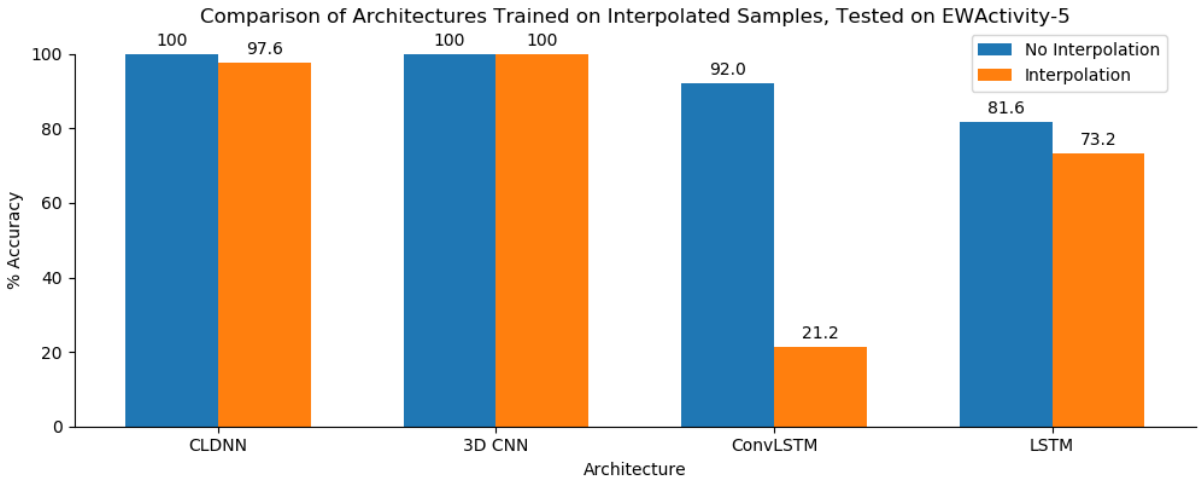

Figure 47. Comparison of architectures trained on interpolated samples and tested on the EWActivity-5 test variant.

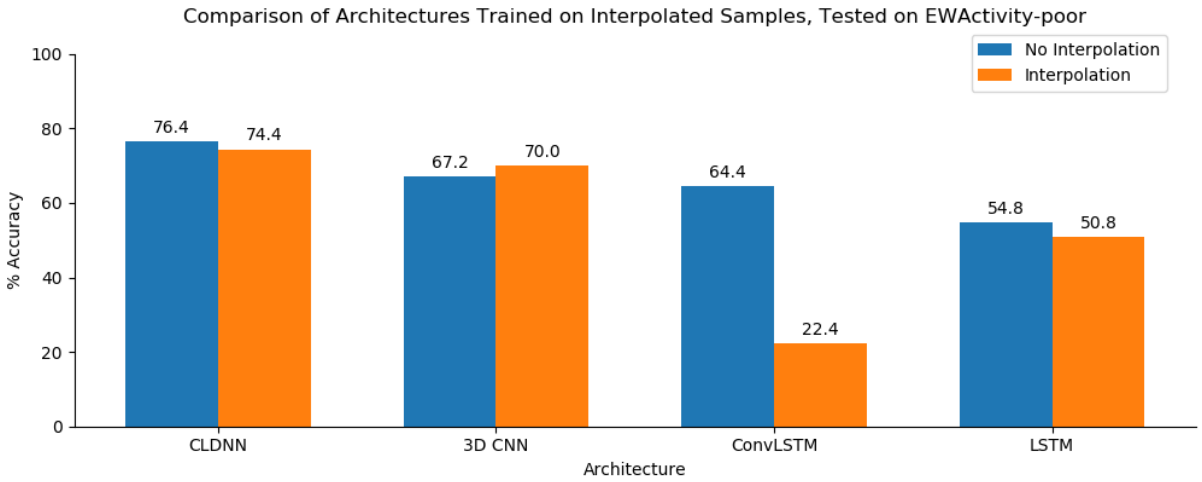

Figure 48. Comparison of architectures trained on interpolated samples and tested on the EWActivity-poor test variant.