sEMG Classification with Convolutional Neural Networks

A Multi-Label Approach for Prosthetic Hand Control

Alexander Olsson

February 5, 2018

LUND
UNIVERSITY

Master's Thesis in
Biomedical Engineering

Faculty of Engineering LTH
Department of Biomedical Engineering

Supervisor: Nebojša Malešević

Acknowledgments

I would like to thank the Department of Biomedical Engineering (BME) for granting me the opportunity to perform my master thesis, as well as for supplying the necessary (and expensive) hardware and software. In particular, I would like to thank my supervisor Nebojša Malešević and my examiner Christian Antfolk not only for providing invaluable advise, but also for enduring the tedious process of providing me with their sEMG data. Lastly, I would like to extend my gratitude to Filip Johannesson, fellow student and friend, for taking time and effort to create an extraordinarily useful interface for hand movement visualization.

Abstract

In myoelectric prosthesis design, there is often a trade-off between control robustness and range of executable movements. As a low movement error rate is necessary in any real application, this often results in a quite severe limitation on the dexterity of the user. One possible remedy for this could come from the use of multi-label machine learning methods, where complex hand movements can be expressed as the sum of several simple movements.

I investigate how effective state of the art deep learning methods are at classifying HD-sEMG signals. Notable weight is put on extracting multi-label information from both the spatial and temporal signal domain of EMG signals by use of convolutional neural networks (CNN). In addition, to investigate the feasibility of reducing the number of necessary electrodes, a novel method for quantifying channel importance is proposed.

I show that multi-label classification performance can rival that of classical single-label methods, even with a large set of labels. Despite the general stochasticity of sEMG signals, no manual feature engineering is necessary and a very short time window is sufficient for accurate classification.

Glossary

Batch In neural network training; a subset of all training data which is passed through the network before its parameters are updated.

Epoch In neural network training; one complete use (forward and backward pass) of all available batches.

Feature In this thesis, the output of a transformation of data that is (hopefully) more descriptive/useful than the input.

Hyperparameter In machine learning; a parameter whose value is set prior to the training/fitting of the model.

Iteration In neural network training; the forward and backward pass of one batch. One epoch contains iterations equal to the number of batches.

Tensor A data structure of arbitrary dimensionality; a generalization of vectors and matrices into 3D space and beyond.

Brief Notes on Notation

In this thesis, the use of boldface when denoting a variable, e.g. $\mathbf{X}$, signifies that it is a vector, matrix or tensor of higher dimensionality, i.e. not a scalar. When denoting individual, scalar elements of tensors, parentheses with coma separated indices will be used and boldface will be omitted, e.g. $X(j, k, l)$. When only one index is used, such as $X(i)$, this does not necessarily mean that $\mathbf{X}$ is of one dimension, but that only the linear index is relevant; for example $X(i) = X(j, k, l)$, $i \in 1..I$, $I = \sup(j) \cdot \sup(k) \cdot \sup(l)$. One-indexing is consistently used for all tensors. When treating functions with multidimensional range, braces replaces parentheses, e.g. $f(\mathbf{X})\{i\}$, to eliminate confusion with multiplication and in all cases leaving subscript for (meta)indexation and superscript for (element-wise) exponentiation.

Contents

Chapter 1

Introduction

When contemplating prosthetic hand design, one of the first and most salient problems to emerge is the issue of control. While the human hand can carry out an enormous breadth of tasks without much apparent effort on the part of its owner, this is not currently the case for even the state of the art of prostheses. Separate from the domain of mechanical prosthesis design; an endeavour with increasingly sophisticated results, there are many angles from which the task of making it possible for the wearer to command such a contraption can be approached. One such approach which has garnered much attention in recent times is the use of *myoelectric* devices; a class of prostheses controlled by the electrical activity of the muscles in the residual arm of the prospective user. While having been usefully employed for many decades by now, they still struggle with many problems; mainly the trade-of between control effort, movement robustness and dexterity.

One set of possible solutions to these problems has come from the rapidly evolving academic field of machine learning. By recording the surface electromyogram (sEMG) with a matrix of electrodes along the arm of a subject, an image of sorts (where each pixel corresponds to an electrode sample) can be acquired. This data can be used to train a classifier to be used for hand posture recognition. The recognized poses, provided the recognition is robust enough, could subsequently be used to instruct the mechanical behaviour of a prosthesis. Furthermore, with the use of multi-label classifier algorithms, data can be assigned many different combinations of classes simultaneously. This is of interest in recognition of hand and wrist postures and movements, as they inherently posses many independent degrees of freedom. One type of classifier that has received much attention for its unprecedented accuracy, specifically in the field of image recognition, is the convolutional neural network (CNN). Its success with images, as well as its promising ability to operate in multi-label cases, makes it of special interest.

1.1 Objectives

The main goal of this thesis is to investigate how effective a convolutional neural network could be at classifying high density surface electromyography (HD-sEMG) signals collected from the arm of a subject. Distinctive weight is put on investigating if it is possible to use multiple, non-mutually exclusive classes without unacceptable loss of performance. Finding answers to this large question will entail investigating smaller issues, such as:

- Finding a good method for EMG measurements to ensure maximal information extraction and hence classification accuracy.
- Figuring out what features, or rather preprocessing steps in general, are desirable for good results at later stages in the algorithm.
- Deciding what architecture should be selected for the convolutional neural network.
- Investigating whether it possible and helpful to extract information present in the transient behaviour of the EMG signal for classification purposes.

1.2 Earlier Work

In a paper from earlier this year, Yu Du et al.[1] achieved surprisingly good results by using a convolutional neural network to classify single time instant sEMG images. With this they showed that time window based feature extraction might not be mandatory, as was previously thought by many [2]. The authors of [3] were successful in applying their own network on the NinaPro database of sEMG recordings with a much more sparse electrode setup, and achieved quite impressive results. Among others, many of the additional methods novel to the specific problem considered here are introduced in [4], [5] and [6].

Chapter 2

Theory

This chapter aims to present the general theoretical background by which the methods employed in this thesis are inspired. Initially, some time is spent delving into the properties and morphology of the electromyographic signals. In order to reinforce understanding for the specific task at hand, an overview of neural networks and their principles and structure is presented. Particular weight is put on explaining the relevant layers, loss functions and optimizers. Finally, in order to emphasize the possibilities and challenges of multi-label learning, a section is dedicated to explaining some pertinent performance measures and their interpretation.

2.1 Surface electromyography

The *electromyogram* (EMG) [7],[8] is a bioelectrical signal which represents the electrical activity in skeletal musculature. It is generated by the neuromuscular activation of motor units during muscle contraction; the more motor units recruited the higher the amplitude. The signal is acquired by measuring the difference in electrical potential between points in or along the muscle of interest, and EMG is therefore always measured by multiple channels. Electromyography can be divided into two main categories; intramuscular EMG (iEMG), where measurements are taken via needles inserted into the muscle, and surface EMG (sEMG), where measurements are taken via electrodes on the skin along the muscle. While less invasive, sEMG suffers from some considerable drawbacks compared to iEMG. Notably, the signal at each recording site is composed of a superimposition of the activations of nearby muscle fibers, which consequently lowers spatial resolution. The high impedance of skin tissue also severely decreases signal strength compared to iEMG. In this thesis, only sEMG is taken or considered.

While sEMG is useful for diagnosing in a plethora of clinical situations, such applications lie far beyond the scope of this work (although similar procedures as those outlined in this thesis could be used for assessing features for detecting e.g. stroke). What is relevant is the fact that forearm sEMG has been shown to predict hand grip forces [9] and because of this been utilized to control hand prostheses. Extension and flexion of the digits is mainly controlled by the extensor digitorum communis (EDC) and the flexor digitorum profundus (FDP) muscle situated at the radial and ulnar surface, respectively, of the forearm. It is therefore not surprising that information about hand movement is encoded in the sEMG signal, however deeply concealed. An example of a sEMG channel acquired in this thesis work is shown below in fig. 2.1.

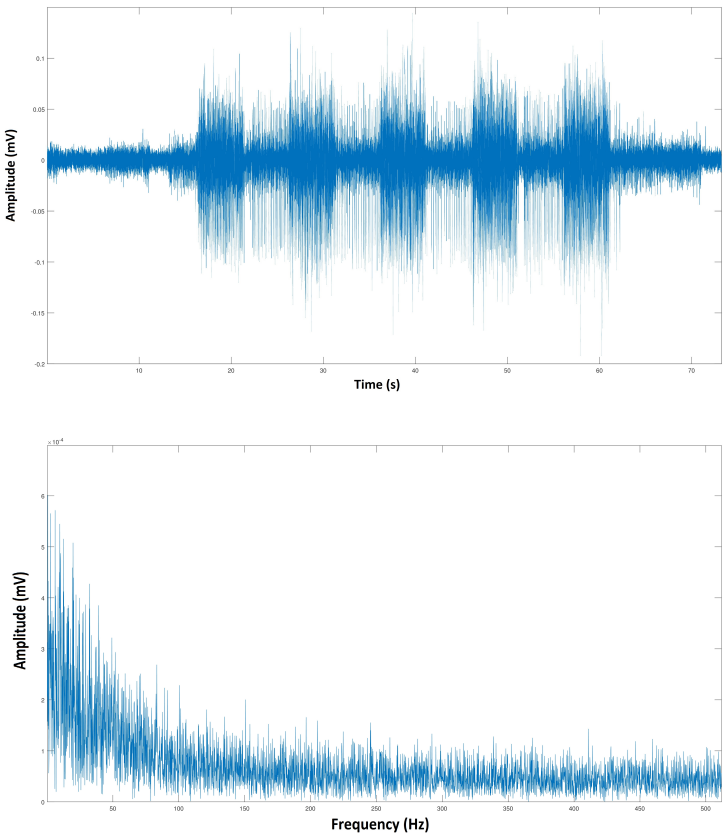


Figure 2.1: (top) A differential sEMG channel measured from the forearm during 5 repetitions of little finger flexion. (bottom) Its frequency content, computed via FFT.

2.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a set of computationally distributed machine learning algorithms originally inspired by biological systems [10], although the similarities are seldom much more than conceptual. In order to understand the design of a neural network, it is helpful to contemplate its purpose. While artificial neural networks possess utility in a wide range of applications, their in some sense most primal area of usage, as well as the one under consideration here, is *classification*.

2.2.1 Classification in Machine Learning

With notation partially inspired by [5], consider a list of observations in the form of the pairings $(\mathbf{X}_1, \mathbf{y}_1), (\mathbf{X}_2, \mathbf{y}_2), \ \dots \ , (\mathbf{X}_n, \mathbf{y}_n)$, where $\mathbf{X}_i \in \mathbb{R}^{\mathbf{d}}$, henceforth called *input data*, is a tensor of arbitrary but consistent (w.r.t. i) size $\mathbf{d}$ and $\mathbf{y}_i \in \{0, 1\}^q$ is called its *class* or sometimes its *label set* depending on the mutual exclusivity of its elements (see section 2.3). q is the number of possible labels, and $(\mathbf{X}_i, \mathbf{y}_i)$ is henceforth collectively called an *example*. If one assumes that each $\mathbf{X}_i$ is sampled from an unknown probability distribution $\chi(\mathbf{X}; \mathbf{y}_i)$ dependent upon the corresponding $\mathbf{y}_i$, there exists at least one optimal transformation $\phi(\cdot)$, in the sense that the probability $P(\phi(\mathbf{X}_i) = \mathbf{y}_i)$ is maximized $\forall i$. The problem of approximating this transformation to a satisfactory degree based on the observations and applying it is called *classification*; one of the main tasks with which contemporary machine learning is concerned. In the problem to be solved in this thesis, the input data set $\mathbf{X}$ represents HD-sEMG measurements acquired from the arm of a subject, and the label set $\mathbf{y}$ is a representation of the ongoing hand movement at time of measurement. More details concerning this modeling are presented in chapter 3. With the problem delineated, we can now move on to the specific case of artificial neural networks.

2.2.2 Neurons and Layers

The most basic element of an ANN is the *artificial neuron* [11]. The artificial neuron is a function that takes an argument $\mathbf{x}$ of arbitrary dimensionality and produces a scalar value x', calculated according to eq. 2.1.

$$x' = f(\sum_{k=1}^{n} x(k)w(k) + b) \tag{2.1}$$

where x' is the output (also known as the *axon* or more commonly, *activation*), $x(k)$ and $w(k)$ is the k:th input and weight, respectively (the term $a(k)w(k)$ is known as a *dendrite*), b is the *bias* and $f(\cdot)$ is the *activation*

function. The biological terminology is presented for increased intuition and will not be used further in this thesis. We shall later return to the choice of activation function, but for now it suffices to say that it in some way limits to small inputs from producing to large output i.e. *activating* the neuron. A graphical representation of the artificial neuron is presented below in fig. 2.2 together with its biological counterpart.

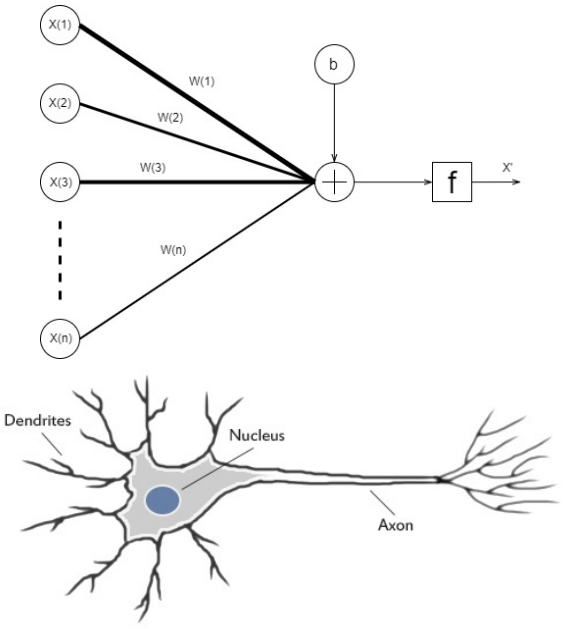


Figure 2.2: Artificial (top) and biological (bottom) neuron [12].

In order to obtain an actual network (or if one wants to be pedantic, a *feed-forward network*), neurons are grouped together into *layers*, and the output of each layers is used as input for the subsequent layer. Whilst we shall later see (e.g. section 2.2.6.1) that not all layers are constructed exclusively from the neuron in eq. 2.1, this is initially useful as a mental model. The first layer is the *input layer*, whose size and content is always the same as the input data $\mathbf{X}$ discussed previously, and the final layer is the *output layer*, whose size is always the same as the label set $\mathbf{y}$. The layers in-between are usually referred to as *hidden layers*. To use the network as a classifier, the problem is now to tune the weights and biases (and possibly other learnable parameters) of the network in order to make it approximate the transformation $\phi(\cdot)$ from section 2.2.1 to a satisfactory degree. Before methods for approaching this task are presented, a taxonomy of relevant layers is needed.

2.2.2.1 The Fully Connected Layer

The *fully connected* (FC) layer, sometimes called the *dense* layer, is perhaps intuitively the most straight-forward type of layer, but by the same token possibly the most computationally expensive. The output of this layer is computed by connecting every element in the input to every element in the output: given an input tensor $\mathbf{X}$, the output tensor $\mathbf{X}'$ is calculated via a linear combination of inputs as in eq. 2.2:

$$X'(i) = \sum_{k=1}^{n}(W(i,k)X(k)) + b(i)\ ,\ i \in 1..m \iff \mathbf{X}' = \mathbf{W}\mathbf{X} + \mathbf{b} \quad (2.2)$$

Where $n = |\mathbf{X}|$ is the number of input elements, $m = |\mathbf{X}'|$ is the number of output elements, $W(i,k)$ is the learnable weight connecting the k:th input to the i:th output and $b(i)$ is the learnable bias of the i:th output. A graphical model of the fully connected network layer is presented below in fig. 2.3. In this work, as well as in others, an activation function is often subsequently applied to every output. In the model described in section 2.2.2, we can realize that each element in $\mathbf{X}'$ is generated by its own neuron with a possibly unique set of weights and bias. The size m of $\mathbf{X}'$ that can be selected for the layers is arbitrary and not dependent on $\mathbf{X}$; the numbers of parameters in the layer is $n \cdot m + m = m(n+1)$, $n \cdot m$ weights and m biases. The final layer of all networks used in this thesis (and often in other works) is a fully connected layer with output size equivalent to the label set size q.

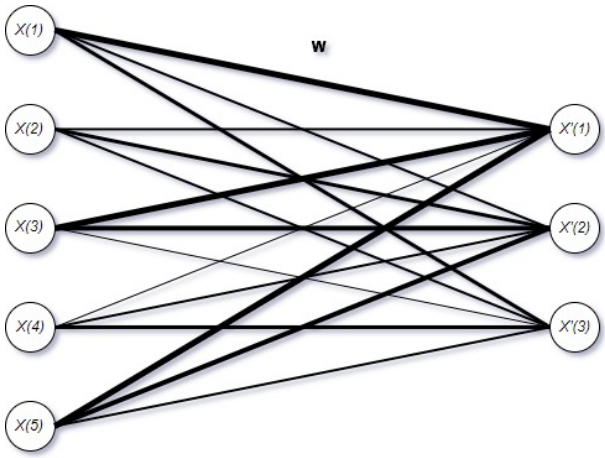


Figure 2.3: Example of a fully connected layer. In this case, the input tensor $\mathbf{X}$ is of size $[1,1,5]$ and the output tensor $\mathbf{X}'$ is of size $[1,1,3]$. The varying thickness of the connections signifies the varying magnitudes of elements in $\mathbf{W}$. Depiction of biases is omitted for clarity.

2.2.2.2 The Convolutional Layer

The *convolutional* layer [13],[14] generates its output tensor, as the name implies, by convolving the input tensor with one or more kernels of fixed size. The importance and usefulness of this layer has given rise to the convention that networks containing it are collectively known as *convolutional neural networks* (CNN). The type of convolutional layer under consideration here is the two-dimensional variant, where the kernels are of size $[h, w, d]$ (h and w are hyperparameters, while d is the depth of the input tensor), although the principle generalizes to higher dimensions without issue. The archetypal use for this type of layer is in image analysis and computer vision, as fully connected layers do not scale well with full images due to their size and the curse of dimensionality [15]. Along with the the kernel size (also denoted the *receptive field* refering to the related property of sensory neurons in biology), this layer requires other hyperparameters: Zero padding $\mathbf{P} = [P_y \ P_x]$, stride $\mathbf{S} = [S_y \ S_x]$ and filter depth F. $\mathbf{P}$ determines if elements with value 0 are added along the spatial (width and height) edges of the input tensor and if so how many along each dimension, while $\mathbf{S}$ determines the length the kernel moves along the spatial dimensions with each step when calculating the output. The filter depth F (not to be confused with the input tensor depth d) represents the number of kernels in the layer. To construct the output tensor, the results (called the *activation maps*) from convolving (i.e. filtering) the input tensor with each kernel are stacked along the depth dimension. A visual representation of a simple convolutional layer can be viewed below in fig. 2.4. A more stringent mathematical definition of the convolutional layer is presented below in eq. 2.3.

$$X'(i,j,k) = \sum_{l=0}^{h-1} \sum_{m=0}^{w-1} \sum_{n=1}^{d} \Big(W_k(l+1, m+1, n) X(S_y i + l, S_x j + m, n) \Big) + b_k \tag{2.3}$$

Where $\mathbf{X}'$ is the output tensor, $\mathbf{X}$ is the (possibly zero-padded) input tensor, $\mathbf{W}_k$ is the learnable k:th kernels weights and b_k is its bias; $k \in 1..F$. The operation can be interpreted as the kernels 'sliding' over the input tensor; each kernel producing a single depth slice of the output tensor. Contrary to the fully connected layer, the size of the output of this layer strongly depends on the size of the input. In the relevant 2D image related case, if $\mathbf{X}$ is of spatial size $\mathbf{V_x}$, the spatial size of the output tensor $\mathbf{X}'$ is $\mathbf{V_{x'}} = \frac{\mathbf{V_x} - [h \ w]^T + 2\mathbf{P}}{\mathbf{S}} + 1$, while having depth F. A common strategy, and one which will be used throughout this thesis, is to set the zero padding to a value which preserves the spatial dimensions of the input.

Worth noting is the fact that the convolutional layer fundamentally **cannot represent any transform that the fully connected layer in principle cannot**. The convolutional layer is in essence a fully connected layer with *weight sharing*; i.e. the same weight are repeated iteratively over the

entirety of the input tensor. This rationing of weight memory is motivated by the assumption that if a feature is relevant to calculate at one spatial index, it is also worth calculating at every other spatial index.

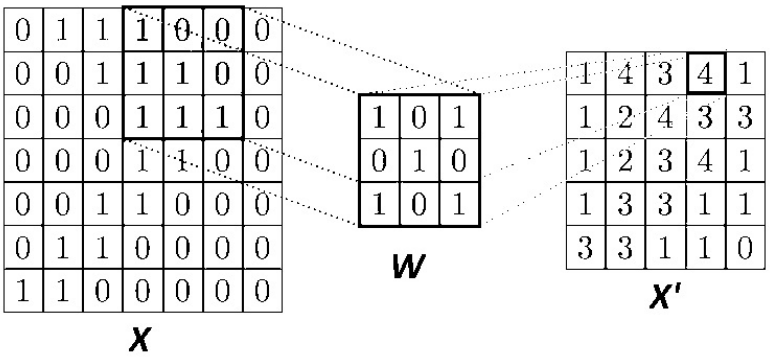


Figure 2.4: Example [16] of a convolutional layer in action. The bias is zero and hyperparameters are set as $h = w = 3$, $\mathbf{S} = \mathbf{1}$, $\mathbf{P} = \mathbf{0}$ and $F = 1$.

2.2.2.3 The Batch Normalization Layer

The *batch normalization* (batchNorm) layer [17] is a proposed solution to the problem known as *covariate shift* [18]. In essence, in large networks the probability distribution functions of input tensors in later layer depends heavily on the weights of earlier layer, possibly generating instabilities in activations during training. This leads to the need for smaller learning rates (see section 2.2.5 on optimizers) and thus slows training down. To combat this, the batchNorm layer normalizes its input tensor over the current batch. The exact transform is given below in eq. 2.4.

$$X'(i) = \gamma\left(\frac{X(i) - \hat{\mu}_i}{\sqrt{\hat{\sigma}_i^2 + \epsilon}}\right) + \beta \tag{2.4}$$

Where $X'(i)$ is the i:th output element, $X(i)$ is the i:th input element, $\hat{\mu}_i$ and $\hat{\sigma}_i^2$ is the mean and variance of the i:th input element (estimated over the current batch) and γ and β are learnable parameters. ϵ is a small static value introduced just to avoid output overflow for small variances. When the network is used for prediction after training, $\hat{\mu}_i$ and $\hat{\sigma}_i^2$ are calculated from the entirety of the training set. Use of this layer can allegedly speed up training and prevent training divergence, which was also observed during the work performed for this thesis .

2.2.2.4 The Residual Block

The *residual block* [6] is not a layer per se, but rather a technique to improve the performance of very deep (i.e. many-layered) networks by way of skipping. In principle, a deep network should never perform worse than any shallower subset network, as the excess layers could assume identity mappings. In practice however, the optimization methods presented in section 2.2.5 do not always manage to reach this configuration. The proposed solution for this is to introduce a shortcut between the input $\mathbf{X}$ and output $F(\mathbf{X})$ of an arbitrary layer (FC, convolutional, etc.), given below in eq. 2.5 with a graphical representation in fig. 2.5.

$$\mathbf{X}' = F(\mathbf{X}) + \mathbf{X} \tag{2.5}$$

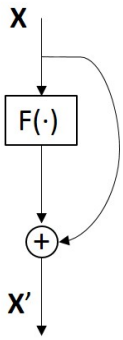


Figure 2.5: The residual block.

Intuitively, this means that the layer $F(\cdot)$ no longer needs to figure out how to transform the data into something more descriptive, but rather what to *add* to the data in order to increase its descriptive qualities. For example, in the case when the layer is redundant and should perform an identity mapping, it is empirically simpler to optimize $F(\cdot)$ towards zero rather than to make it perform an identity mapping for all inputs. It is, of course, also possible to connect this shortcut around multiple layers, i.e. a block, instead of just one as shown above.

2.2.3 Activation Functions

Recall that in eq. 2.1, the artificial neuron contains an activation function $f(\cdot)$. At the network level of analysis, this is conceptualized as an activation layer, or a *nonlinearity*. This name aptly captures the importance of this layer: without any intermingled activation layers, we can deduce from

eq. 2.2 and 2.3 that **every sequence of FC and convolutional layers connected in cascade is equivalent to a single FC layer**. Because of this, a network devoid of activation layers just performs linear regression which, while computationally simple, does not have very high discriminatory capabilities. If this is not a satisfactory explanation, one can note that the activation layer models the role of somata in biological brains; the most powerful known neural networks to date credited for (among other things) writing this thesis, and thus probably worthwhile to mimic. Below follows a short account of the activation layers used in this thesis.

2.2.3.1 The Rectifier Linear Unit

The *rectifier linear unit* (ReLU) layer is described by the relation below in eq. 2.6.

$$f(\mathbf{X})\{i\} = f(X(i)) = \max(X(i), 0) = \begin{cases} X(i) \text{ if } X(i) > 0 \\ 0 \text{ if } X(i) < 0 \end{cases} \tag{2.6}$$

The function is applied element-wise, thus treating the inputs independently and preserving shape. While simple to understand and easy to compute, it has outperformed earlier, more complicated alternatives [19].

2.2.3.2 The Softmax Layer

The *softmax* function layer [20] applied to an input $\mathbf{X}$ is given by eq. 2.7.

$$f(\mathbf{X})\{i\} = \frac{e^{X(i)}}{\sum_{j=1}^{I} e^{X(j)}} \tag{2.7}$$

Where I is the number of elements in $\mathbf{X}$. It is also known as the *normalized exponential*, as $f : \mathbb{R}^J \longrightarrow [0,1]^I$ and $\sum_i^I f(\mathbf{X})\{i\} = 1$. Its output can be considered as a categorical probability distribution, and it can thence be (and in this thesis is) used as the last layer (prior to the loss function) in a multiclass classification network. In such a setup, the output probabilities are implicitly assumed to be mutually exclusive, and as such it offers little use in a multilabel situation.

2.2.3.3 The Sigmoid and tanh Layer

The *sigmoid* function [20] layer can be considered as a element-wise version of the softmax function. It is given below in eq. 2.8.

$$f(\mathbf{X})\{i\} = f(X(i)) = \frac{1}{1 + e^{X(i)}} = \frac{e^{X(i)}}{1 + e^{X(i)}} \tag{2.8}$$

Just like with the softmax function, it holds that $f : \mathbb{R}^J \longrightarrow [0,1]^J$, but **not** necessarily $\sum_i^J f(\mathbf{X})\{i\} = 1$. Consequently, it can represent the probabilities of the presence of certain classes/labels without the assumption of mutual exclusivity. As such, it is useful for multilabel classification. In contrast to the softmax layer, the sigmoid layer can also be effective further back in the network with the same purpose as the ReLU layer, although not used for that purpose in this thesis.

The *hyperbolic tangent* (tanh) function [20] layer is closely related to the sigmoid function and is defined in eq. 2.9.

$$f(\mathbf{X})\{i\} = \frac{\sinh(X(i))}{\cosh(X(i))} = \frac{e^{X(i)} - e^{-X(i)}}{e^{X(i)} + e^{-X(i)}} \tag{2.9}$$

This can be recognized as a sigmoid function that is scaled and has an offset. While this might convey an illusion of equivalence, the tanh activation layer has some advantages [21] related to backpropagation (section 2.2.5). As $f(\mathbf{X})\{i\} \in [-1,1]$, the output of the tanh layer cannot be interpreted as a probability in the same way as the two previous presented layers unless the scaling is reversed. In this thesis, it is exclusively used as output activation in conjunction with BP-MLL loss (section 2.2.4.2).

2.2.4 Loss functions

In order to actually learn a network to perform something approximating $\phi(\cdot)$ from section 2.2.1 instead of some arbitrary transformation in the class of functions $\hat{\phi} : \mathbb{R}^{\mathbf{d}} \longrightarrow [0,1]^q$, the first step needed is to introduce some sort of error measure. This is the *loss function* $C = \mathcal{L}(\mathbf{s}, \mathbf{y}) = \mathcal{L}(\hat{\phi}(\mathbf{X}), \mathbf{y}) = \mathcal{L}(\mathbf{X}, \mathbf{W}, \mathbf{y})$, where $\mathbf{s} = \hat{\phi}(\mathbf{X})$ is the output of the last activation layer (prior to the loss calculating 'layer') of the network. $\mathbf{s}$ is also called the network *label score* or just *score*, and $s(i)$ can usually be interpreted as the networks guessed probability for the presence of the ith label; $i \in 0..q$. $\mathbf{y}$ is the true label set/class pertaining to the input data $\mathbf{X}$. $\mathbf{W}$ is the collection of all learnable parameters in the network (e.g. the weights and biases of FC and convolutional layers) and C is the *cost*. For a loss function to be effective, it should generate small values of C when $\|\mathbf{y} - \hat{\phi}(\mathbf{X})\|$ is small. In addition, in order for the network to be trainable, $\mathcal{L}$ must be *differentiable* w.r.t. every element in $\mathbf{W}$, i.e. $\exists \frac{\delta \mathcal{L}(\mathbf{X},\mathbf{W},\mathbf{y})}{\delta W(i)} \forall i$. This is perhaps the most distinct reason why the classification accuracy (or rather the classification error) is not practical as a loss function for gradient-based learning strategies such as neural networks. Out of the plethora of available loss functions, two were used in this thesis: cross-entropy loss and BP-MLL loss, defined below.

2.2.4.1 Cross-Entropy Loss

The *cross-entropy* loss function [22], with variable names conserved since the previous section, is given by eq. 2.10.

$$C_{CE} = \mathcal{L}(\mathbf{s}, \mathbf{y}) = \sum_{j=1}^{q} y(j) \log(\frac{1}{s(j)}) = -\sum_{j=1}^{q} y(j) \log(s(j)) \tag{2.10}$$

The cross-entropy function has many quite profound interpretations in information and coding theory relating to the concept of mutual information [22], but these lie beyond the scope of this thesis.

2.2.4.2 BP-MLL loss

The *backpropagation multi-label learning* loss function (BP-MLL) [5] is an improvement over the basic sum-of-squares error, specifically designed for multi-label neural networks. Its definition (different from the inventors', but more in line with the general notation of this thesis) for label set $\mathbf{y}$ and network score $\mathbf{s}$ can be viewed in eq. 2.11 below.

$$C_{BP-MLL} = \frac{\sum_{i=1}^{q} \sum_{j=1}^{q} \Big(y(i) \cdot (1 - y(j)) \cdot e^{-(s(i)-s(j))} \Big)}{\Big(\sum_{i=1}^{q} y(i) \Big) \Big(\sum_{i=1}^{q} 1 - y(i) \Big)} \tag{2.11}$$

In short, for every pairing of a relevant and an irrelevant label, the function increases the total loss by an amount **inversely proportional to the exponential of the difference between their respective label score**. This generates a severe penalty in the case when the network gives higher scores to irrelevant labels than to relevant ones. The cost is finally normalized by dividing by the number of such pairings. It is worth noticing that as the layer prior to this loss is the tanh layer and not softmax/sigmoid, the elements of $\mathbf{s}$ used in eq. 2.11 must be scaled as $\frac{\mathbf{s}+\mathbf{1}}{2}$ in order to be appropriately interpreted as label probabilities.

2.2.5 Optimizers

With a loss function $\mathcal{L}$ defined, one could justly intuit that improving the networks performance has at least something to to with reducing its output w.r.t. $\mathbf{W}$ over a large amount of input data. One mathematically coherent way of doing this is called the *backpropagation* algorithm [21], briefly summarized here. In order to train the network, all parameters are initialized in some manner. The network is then applied to input data, generating some cost via the loss function. If input data is sent in a batch, the cost is calculated as the average over the batch. As we explicitly know the operations

the network performs in each layer, the derivative of the cost **w.r.t. every parameter in the network individually**, i.e. $\nabla_{\mathbf{W}}(C)\{i\} = \frac{\delta \mathcal{L}(\mathbf{W},\mathbf{X},\mathbf{y})}{\delta W(i)}$, can be calculated via the chain rule of calculus. How this gradient is subsequently used to update the network parameters differs between optimizers, of which the employed examples are presented below.

2.2.5.1 Stochastic Gradient Descent

The stochastic gradient descent (SGD) algorithm [23] as an iterative optimization method predates neural networks by quite a margin. From elementary multivariate calculus, it is known that the gradient of a function constitutes the direction of fastest increase in argument space. The SGD exploits this fact by iteratively updating the parameters by translating them in the opposite direction of the gradient, as seen in eq. 2.12.

$$\mathbf{W}_{(k+1)} = \mathbf{W}_{(k)} - \alpha \cdot \left(\nabla_{\mathbf{W}} \mathcal{L}(\mathbf{W}, \mathbf{X}_k, \mathbf{y}_k) \Big|_{\mathbf{W}=\mathbf{W}_{(k)}} \right) \tag{2.12}$$

Where k is the iteration number and α is a hyperparameter called the *learning rate*, directly proportional to the parameter updating step size. Keep in mind, $(\mathbf{X}_k, \mathbf{y}_k)$ does not necessarily represent a single training example but rather a batch in which case the loss gradient is evaluated as the batch average. The updating is repeated for a certain number of steps, more often than not determined by a static hyperparameter or until some measure of convergence has been achieved. When the list of available training batches is exhausted, it will loop around and initialize a new epoch.

In this thesis, the original SGD algorithm is never applied. Instead an improvement usually denoted stochastic gradient descent with *momentum* (SGDM) [24] is used. The mathematical formulation of its update rule is given by eq. 2.13.

$$\begin{aligned} \mathbf{v}_{(k+1)} &= m \cdot \mathbf{v}_{(k)} + \alpha \cdot \left(\nabla_{\mathbf{W}} \mathcal{L}(\mathbf{W}, \mathbf{X}_k, \mathbf{y}_k) \Big|_{\mathbf{W}=\mathbf{W}_{(k)}} \right) \\ \mathbf{W}_{(k+1)} &= \mathbf{W}_{(k)} - \mathbf{v}_{k+1} \end{aligned} \tag{2.13}$$

Where $m \in]0, 1]$ is an additional hyperparameter called *momentum*, and $\mathbf{v}$ is called the *velocity*. The terminology is inspired by the Newtonian case of a weight traversing a potential field with shape equivalent to the loss function, in which the loss gradient would be proportional to the force at each time step. This method gives the list of learnable parameters 'inertia' as it traverses the (multidimensional) loss surface, making it more robust to the stochastic fluctuations common in regular SGD.

2.2.5.2 The Adam Algorithm

The Adam (adaptive moment estimation) algorithm [4] is an iterative optimization method building on the success of AdaGrad [25] and RMSProp [26]. Its update rule(s) are formulated below in eq. 2.14.

$$\begin{aligned}
\mathbf{g}_{(k+1)} &= \nabla_{\mathbf{W}} \mathcal{L}(\mathbf{W}, \mathbf{X}_k, \mathbf{y}_k)\Big|_{\mathbf{W}=\mathbf{W}_{(k)}} \\
\mathbf{m}_{(k+1)} &= \beta_1 \cdot \mathbf{m}_{(k)} + (1-\beta_1) \cdot \mathbf{g}_{(k+1)} \\
\mathbf{v}_{(k+1)} &= \beta_2 \cdot \mathbf{v}_{(k)} + (1-\beta_2) \cdot \mathbf{g}^2_{(k+1)} \\
\hat{\mathbf{m}}_{(k+1)} &= \frac{\mathbf{m}_{(k+1)}}{1-\beta_1^{k+1}} \ , \ \hat{\mathbf{v}}_{(k+1)} = \frac{\mathbf{v}_{(k+1)}}{1-\beta_2^{k+1}} \\
\mathbf{W}_{(k+1)} &= \mathbf{W}_{(k)} - \alpha \cdot \frac{\hat{\mathbf{m}}_{(k+1)}}{\sqrt{\hat{\mathbf{v}}_{(k+1)}} + \epsilon}
\end{aligned} \tag{2.14}$$

Where $\mathbf{g}$ is the loss function gradient used previously and $\hat{\mathbf{m}}$ and $\hat{\mathbf{v}}$ are real-time running, bias-corrected estimates of the first and second moments (mean and variance) of each element in $\mathbf{g}$. As usual, ϵ is just a small number introduced to avert numerical instabilities. $\beta_1, \beta_2 \in [0, 1[$ are hyperparameters called *exponential decay rates* for the first and second moments, respectively. The algorithm is initialized by setting $\mathbf{m}_{(1)} = \mathbf{v}_{(1)} = \mathbf{0}$. The step size is thus not statically set to the learning rate α, but updated in accord with the statistical properties of the ever-changing gradient.

The main appeal of this method, at least in this thesis, is its robustness when faced with the often time non convex properties of the loss function. In the volatile optimization environment that is ANNs, the regular SGD and even SGDM can be very sensitive to the choice of α. If to large, the training runs the risk of diverging; if overly small the training is slow and might even get stuck in the irregularities of the loss function 'surface' and never converge. The Adam algorithm generally circumvents this problem by course correcting appropriately, and is consequently sufficiently effective for a much wider range of hyperparameter values. In addition, as step size decreases when the gradient does so, the risk of 'overshooting' the optimum is reduced.

2.2.6 Overfitting Countermeasures

When developing ANNs, and machine learning algorithms in general, a very common and often necessary approach is to split the available data (with corresponding label sets) into a training set and a test set. The training set is used to actually train the network according to the methods presented thus far. However, the networks performance on the training set is not a fair measure of its proficiency. For example, as shown in [27], adequately deep

networks can achieve perfect results (e.g. ≈ 0 cost) on randomly labeled and even randomly generated, completely unstructured training data, without at all generalizing to novel examples. Consequently, the network is applied to a test set once training is finished in order to evaluate how well the network performance generalizes to data it has never encountered.

If the training has gone on for a large number of iterations and the network contains sufficiently many learnable parameters, i.e. has enough *expressivity*, it is likely that the network performs well on the training set, but badly on the test set. This problem is usually referred to as *overfitting*, and is caused by the fact that the network model has, with its many learnable parameters, memorized the exact intricacies of the examples in the training set instead of the general properties of the unknown, underlying distribution assumed to generate all data. This can be a massive issue and becomes more salient the more expressive a network becomes. Below follows an outline of the approaches utilized in this thesis to combat the problem of overfitting.

2.2.6.1 Dropout Layer

For a given input tensor, the dropout layer [28] iterates through all of its elements and randomly sets some of them to zero. The probability that such a operation is performed is a hyperparameter inherent to the layer, often set to $P = 0.5$. To compensate for the lack of activation passed on to subsequent layers, the output is scaled up by a factor $\frac{1}{P}$. As the inclusion of this layer prevents the network from systematically relying on specific neurons, this is a very simple technique which can drastically reduce the tendency of the network to overfit.

When the training is finished and the network is to be used for prediction and evaluation, all dropout layers are ignored.

2.2.6.2 L2 Regularization

A common approach in many optimization problems is that of *regularization*. This is done by appending the cost generated by the loss function with a regularization term $R(\mathbf{W})$, in simple terms thought of as representing the complexity of the model under consideration. In ANNs, using the terminology from section 2.2.4, this can be expressed as in eq. 2.15.

$$C_R = \mathcal{L}_R(\mathbf{X}, \mathbf{W}, \mathbf{y}) = \mathcal{L}(\mathbf{X}, \mathbf{W}, \mathbf{y}) + R(\mathbf{W}) \tag{2.15}$$

Where $\mathcal{L}_R$ is the regularized cost function. In this thesis, only the L2 regularizer, also known as *weight decay*, was applied. Its formulation is

given below by eq. 2.16.

$$R(\mathbf{W}) = \lambda \cdot \sum_{i=1}^{|\mathbf{W}^*|} \|W^*(i)\|^2 \tag{2.16}$$

Importantly, $\mathbf{W}^*$ is the vector containing all learnable parameters in the network, **excluding biases**. $|\cdot|$ denotes set cardinality, while $\|\cdot\|$ is the absolute value operator. λ is a hyperparameter called the *regularization parameter*. It is also important to note that, as mentioned under section 2.2.5, the loss is evaluated as the batch average during training. If a batch size larger than one is used, $R(\mathbf{W})$ should be divided by this batch size before being added to the total cost.

Intuitively, the addition of this term is a way to coerce the optimizer to prefer small values for the learnable parameters; all else being equal, $\mathbf{W}^*$ will tend towards the origin of the learnable parameter space. If allowed to be pretentious, one could view this as enforcing something resembling the famous heuristic of Occam's razor on the trained model. This counteracts overfitting by giving overly convoluted parameter configurations an added cost, but also provides an additional advantage in that it continuously perturbs the training. In this way, the optimization process is unlikely to get 'stuck' in the small local minimas of the loss function.

2.2.6.3 Holdout Validation

Even with the previous countermeasures imposed, the network is still very likely to eventually reach an overfitted state after a certain (albeit large) number of iterations have passed. Because of this, some way of performing *early stopping* once this stage is reached would be quite useful. A naive approach would be to regularly pause the training to investigate whether the test data performance has decreased since the last pause, as this is an indicator of imminent overfitting. If this is the case, the training would be halted. Alas, while this at first glance seems to actually stop overfitting, it only transfers the problem. Now the choice of hyperparameters and even the network architecture itself runs the risk of being **manually 'overfitted'** by their creator, as the network will never be tried on truly novel input data.

In order to work around this issue, one could introduce a third set of input data laterally to the training and test set; the *validation set*. The procedure is then to pause the training at regular, set intervals, known as the *validation frequency*, and evaluate the average cost of the entire validation set. If an increase persists through a certain number of pauses, known as the *validation patience*, the training is aborted. While thinning the amount of data available for training and testing, this strategy circumvents the concerns of manual overfitting.

2.3 Multi-Label Machine Learning

There are many situations in which a multi-label model is much more useful than a single-label one. Consider, for example, identifying what objects are present in an image (e.g. dog **and** car), or medical diagnosis from biometrical data (e.g. cancer **and** cystic fibrosis). As is argued in chapter 1, hand movement prediction is not an exception to this rule. In preceding sections of this chapter, an effort has been made to describe the theory independent of whether the network will work with a single-label problem or a multi-label problem. There are however some details that could benefit from clarification. Recalling the framework delineated in section 2.2.1, a *single-label classification* problem is one where the label set $\mathbf{y} \in \{0, 1\}^q$ for each example contains **one and only one** 'relevant' (i.e. equal to 1) element. If $q > 2$, the problem is called *multiclass classification* and if $q = 2$ it is known as *binary classification*. *Multi-label classification* on the other hand describes the case when no such requirement of mutual exclusivity exists; every combination of relevant and irrelevant labels is allowed.

An issue that sets classifiers purposed for the single-label and multi-label case apart is how to generate a *prediction* $\mathbf{p}$ from the class scores $\mathbf{s}$. In the single-label case, the predicted label is simply assumed to be that with highest score, as no more than one can be present simultaneously. In this thesis, the jth label in a multi-label environment is assumed as predicted if it falls above some threshold t, i.e. $p(j) = \mathbb{1}(s(j) > t)$, where $\mathbb{1}(\cdot)$ is the indicator function.

In recent times, many effective multilabel machine learning methods have been constructed by modifying ubiquitous single-label methods; prominent examples include ML-kNN [29], multi-label naive bayes [30] and rank-SVM [31]. While some attempts have been to do the same with CNNs (e.g. [32]), the task is still somewhat open-ended and dependent on application. In this thesis, two main approaches will be attempted: by transforming the multilabel problem into q binary classification problems via q different networks, and by modifying the network architecture and cost function to accommodate the multi-label problem statement. Note that in the case of multiple single-label binary networks, no continous score value is used for each label and thus no thresholding is necessary to generate predictions.

2.3.1 Performance Measures

Once the network training has concluded and the resulting model is to be tested, some way to evaluate performance is needed. In single-label classification, the *accuracy* $a \in [0, 1]$, defined as the ratio of correctly classified examples to the total number of examples, is used for this purpose as it is intuitive and in some sense completely describes exactly what we want

the classifier to do. It is, however, not a wholly trivial task to contrive ways to quantify the ability of a trained multi-label classifier (e.g. the CNN). This conundrum stems mostly from the fact that, contrary to the single-label case, a label set prediction performed by the classifier can be *partially correct* in its predictions by way of correctly predicting some labels but not others. In the following sections, some effort is made to describe what measures of error and performance are put to use in this thesis, as it will be essential for interpreting the results presented in chapter 4. In all cases, N is the test set cardinality and $\mathbb{1}(\cdot)$ is the indicator function. $\mathbf{y}_i$ and $\mathbf{p}_i = \mathbb{1}(\mathbf{s}_i > t) = \hat{\phi}(\mathbf{X_i})$ are the correct label set and prediction, respectively, of the i:th test example.

2.3.1.1 Subset Accuracy

The *subset accuracy*, or *exact match rate* [33], is the metric perhaps most related to the regular accuracy of single label classifiers. It measures the fraction of test examples in which the classifier correctly predicts **all** labels, and is calculated via eq. 2.17.

$$\text{Subset accuracy} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{1}(\mathbf{y}_i = \mathbf{p}_i) \tag{2.17}$$

In contrast to the accuracy of a single-label accuracy, its baseline in case of random guessing is not $\frac{1}{q}$, but $\frac{1}{2^q}$.

2.3.1.2 Hamming Loss

Hamming loss [33] is a metric which operates on each label independently by measuring the ratio of wrongly predicted individual labels to total number of labels $q \cdot N$. Mathematically, it can be stated as in eq. 2.18

$$\text{Hamming loss} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{q}\sum_{j=1}^{q}\mathbb{1}\big(y_i(j) \neq p_i(j)\big) \tag{2.18}$$

It is a more lenient metric than the subset accuracy but, because it does not take label dependencies into account, it might give a faulty perception of performance. For example, a classifier might systematically misclassify one particular label but still generate an acceptable Hamming loss.

2.3.1.3 Jaccard Index

The *Jaccard index* [33] is a statistic for quantifying set similarity. Is is quite closely related to the Hamming loss, and is in the notation used so far

defined by eq. 2.19.

$$\text{Jaccard index} = \frac{\sum_{i=1}^{N}\sum_{j=1}^{q}\mathbb{1}\Big(y_i(j)=1 \wedge p_i(j)=1\Big)}{\sum_{i=1}^{N}\sum_{j=1}^{q}\mathbb{1}\Big(y_i(j)=1 \vee p_i(j)=1\Big)} \tag{2.19}$$

A Jaccard index of 1 means that all individual labels predicted to be relevant truly are and vice versa. A score of 0 means the classifier never predicts a label to be relevant when it actually is. Thus, the Jaccard index is not directly improved by labels correctly predicted to be irrelevant (although a label incorrectly predicted to be relevant always reduces it).

2.3.1.4 Precision and Recall

In binary classification, specifically information retrieval, a common measure of performance is *precision* and *recall* combined together into what is known as the *precision-recall curve* [33]. While the multi-class classification problem as such is not binary it can, just as in the case of Hamming loss and Jaccard index, be seen as $q \cdot N$ binary instances; one for each label across all testing examples. To begin with, we define

tp: *True positives*, the number of relevant labels (correctly) predicted as relevant across all test examples.

fp: *False positives*, the number of irrelevant labels (incorrectly) predicted as relevant across all test examples.

tn: *True negatives*, the number of irrelevant labels (correctly) predicted as irrelevant across all test examples.

fn: *False negatives*, the number of relevant labels (incorrectly) predicted as irrelevant across all test examples.

With this, we define precision and recall as in eq. 2.20.

$$\begin{aligned}\text{Precision} &= \frac{tp}{tp+fp}\\ \text{Recall} &= \frac{tp}{tp+fn}\end{aligned} \tag{2.20}$$

Put into words, precision is the fraction of labels predicted as relevant which are actually relevant, while recall is the fraction of actually relevant labels which are predicted to be so. In the single network multi-label cases implemented in this thesis, some classification threshold $t \in]0,1[$ (section 2.3) is always required to transform the output of the final softmax of sigmoid layer to a prediction. Typically, a strict (high) threshold generates good (high) precision and poor (low) recall, while a lenient (low) threshold does the opposite. To visualize this trade-of, the precision-recall curve is created by parametrically plotting the precision and recall at different thresholds.

Chapter 3

Methodology

In this chapter, the practical details of the work performed are outlined. It is inaugurated by a short overview of the work flow in order to give the reader some notion of the direction the work followed. It is followed by a comprehensive review of the tools, data sets and acquisition setup utilized. Lastly, a more detailed description of the specific CNNs and training regimens applied in this thesis is included. The results achieved by following the processes traced here are presented in chapter 4.

3.1 Work Overview

Every sub-task of the main objective in this thesis can be visualized as the simple flow chart presented in fig. 3.1. Firstly, a high density sEMG signal is measured from the arm of a subject while said subject performs a series of hand movements determined prior to the experiment. Via a sequence of processing steps, this signal is transformed into a series of 'images' (more appropriately thought about as tensors), each paired with its relevant label set. Such an image represents the bioelectrical state at the surface of the arm; at a single time instant for the first methods used, and at a series of consecutive time instants later in the thesis. The data is split into training, testing and validation sets which are used to train and evaluate a CNN. What varies between tasks is the details of the steps, e.g. preprocessing methodology and model complexity. Below follows a short summation of what was done to investigate the questions posed in section 1.1, presented in chronological order.

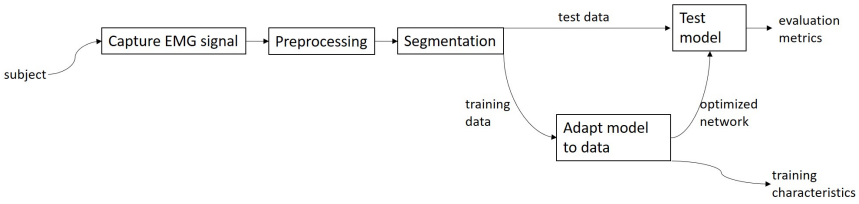


Figure 3.1: High level flow chart describing all processes designed in this thesis.

Initially, a simple recording was acquired from a single test subject. This recording was structured as a series of images, where each image represents the sEMG measured at a single time instant. For these introductory measurements, together with an external data set, a single-label model was assumed. As a proof of concept, the data was processed with a simple network before moving on to a more complex one.

As this proved quite successful, the decision was made to increase the complexity and adopt a multi-label approach. The data acquisition, preprocessing and of course the neural network architecture itself had to be reconstructed for these new circumstances. In total, measurements from 3 subjects were used in this phase in order for the results to possess some degree of generality. As in the previous step, all data used was structured as single time instant images.

Lastly, additional alterations were made to the methodology in order to investigate whether the multi-label network could utilize the time information inherent to the signal by using time-windowed data. For this, the data acquired for the single time instant, multi-label case was reused.

3.2 Tools

3.2.1 MATLAB

MATLAB is a well known proprietary programming language developed by MathsWork for numerical computing. For this thesis, use was made of the *Deep Learning Toolbox*, specifically the class *SeriesNetwork*, for creating single-label neural networks. In all stages of work, MATLAB was used as the primary tool for filtering, structuring and miscellaneous preprocessing.

3.2.2 TensorFlow

TensorFlow [34] is an open source library designed with machine learning as its primary purpose. It is capable of running on GPUs and uses the NVIDIA CUDA Deep Neural Network library (cuDNN) which delivers GPU accelerated implementations of common neural network routines. In this thesis, its Python version was used specifically for the increased customizability it bestows upon the user compared to MATLAB; a necessary feature for implementing the multi-label networks presented later in this chapter.

3.2.3 CapgMyo Data Set

During the early stages of work, before original data had been collected, use was made of the CapgMyo DB-a data set[1], create specifically for the work done in [1]. This is a (single-)labeled sEMG data set with 8 different hand movement, available in preprocessed or raw format. Here, only the preprocessed version was put to use.

3.3 Acquisition Setup

The acquisition protocol remained mostly constant throughout this thesis. Al measurements are recorded with an OT Bioelettronica Quatrocento amplifier, with two 8×8 electrode matrices, coated in conductive gel, placed on the extensor digitorum communis and the flexor digitorum profundus muscles on the right arm of the able-bodied subject, as shown in fig. 3.2.

Figure 3.2: Electrode placement

[1] available from http://zju-capg.org/myo/data/

The arm of the subject is initially placed in a specifically designated stand (fig. 3.3) capable of measuring individual finger and wrist forces. These measurements are never used in the actual classification, but are shown in real time to the subject on a screen as feedback for easier muscle control.

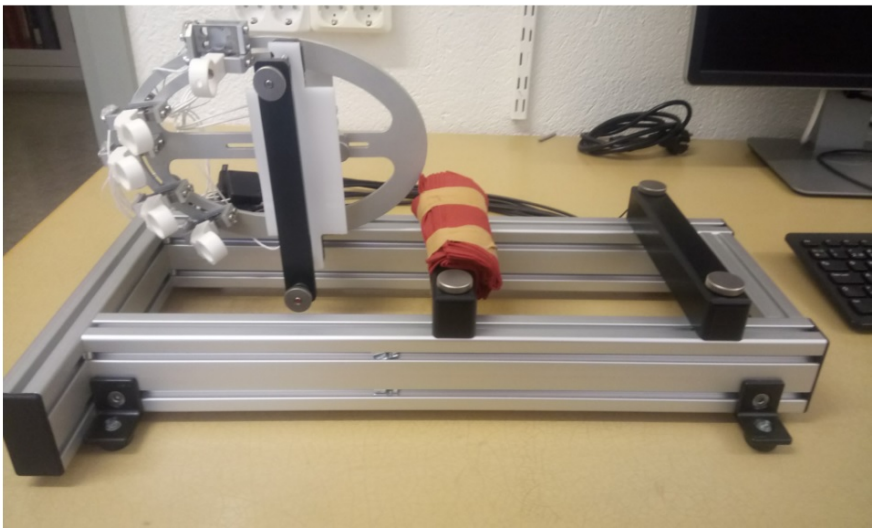


Figure 3.3: Stand for hand placement during measurement.

During acquisition, a LabVIEW VI is used to show the subject a sequence of texts describing the movement he or she is supposed to perform while simultaneously measuring the sEMG signal. The exact sequence of movements differs between experiments, but at every moment the currently shown movement is recorded and paired with its concurrent sEMG measurement. Every movement is repeated 5 times for 5s each, with 5s of rest between every repetition. A sound cue is played at the onset of each movement following rest. During acquisition, the the sEMG sampling frequency was chosen as 2048 samples/s, as a higher rate likely would not contribute with anything other than to duplicate already recorded moments. A lower rate would run the risk of thinning the amount of available input data, and in the extreme case spoil information hidden in the farther parts of the spectrum.

16 labels, representing the main degrees of hand and lower arm freedom, are consistently used throughout acquisition. The label set is composed of flexion and extension of all fingers ($5 \times 2 = 10$ DoF's), thumb abduction and adduction (2 DoF's), wrist flexion, extension, pronation and supination (4 DoF's). A visual representation of these can be viewed on the following page in fig. 3.4, and their numerical encoding can be viewed in tab. 3.1.

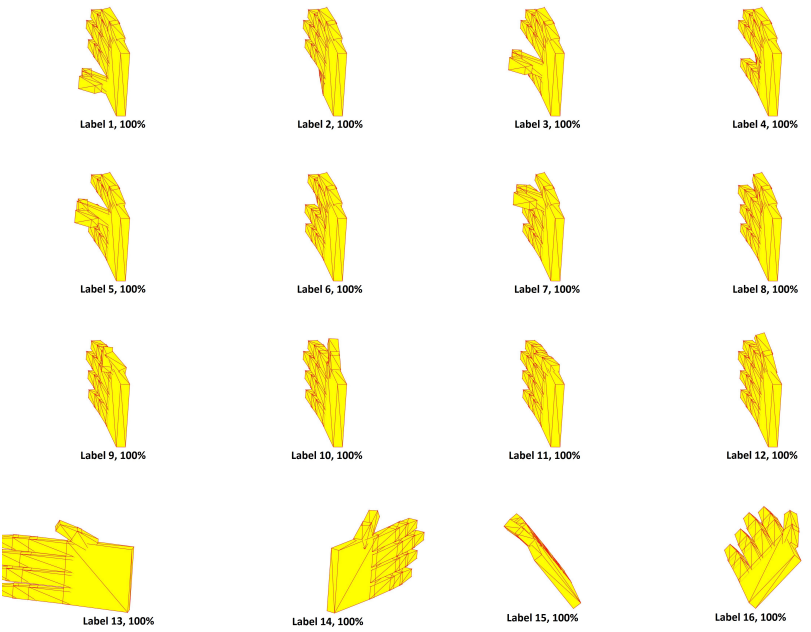


Figure 3.4: Visualization of the hand movement 'basis'

Movement	Relevant label	Corresponding label set **y**
Rest	N/a	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
Little finger flexion	1	[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
Little finger extension	2	[0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
Ring finger flexion	3	[0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0]
Ring finger extension	4	[0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0]
Middle finger flexion	5	[0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0]
Middle finger extension	6	[0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0]
Index finger flexion	7	[0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0]
Index finger extension	8	[0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0]
Thumb flexion	9	[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0]
Thumb extension	10	[0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0]
Thumb abduction	11	[0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0]
Thumb adduction	12	[0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0]
Wrist flexion	13	[0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0]
Wrist extension	14	[0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0]
Wrist pronation	15	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0]
Wrist supination	16	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]

Table 3.1: Hand movement encoding

Despite being measured, the rest state (i.e. time instants with all-zero label set) is never used explicitly in the subsequent CNN training and testing. This is motivated by the fact that rest detection is exceedingly simple, and as such might contribute undeservedly to the evaluated performance of the network(s). While this possibly biases the classification, it could easily be solved in a real application by including a rest state detector prior to the network.

3.3.1 Single-Label

For the single-label case, only 1 recording was collected using a single test subject. As it is implicitly assumed that only one label can be relevant at each time instant, the only movements the subject were instructed to perform are the ones presented in tab. 3.1.

3.3.2 Multi-Label

For the multi-label case, 3 different recordings from 3 different test subjects were collected. Initially, the same movements as in the single-label case were recorded. Of course, in contrast to the single-label case, it is also of interest to record combinations of the basic hand movements, as there might exist interlabel interactions for the networks to potentially learn. These combinations are simple to encode using the same scheme as in tab. 3.1; if one for instance wants to record the label set pertaining to the execution of little and ring finger flexion simultaneously, it would be denoted as $\mathbf{y}_{1+3} = \mathbf{y}_1 + \mathbf{y}_3 = [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$. As there exists 2^{16} such combinations, it would clearly be folly to record every single one of them; instead a subset judged to be somewhat representative were selected in the interest of time. Beyond the single label movements, every 2-label combination were recorded, **excluding movements containing finger extension together with finger flexion, or with two digits separated by 1 or more intermediate digits**. In addition, some 3, 4 and 5 label combinations of great importance were manually selected, namely all fingers extension, all fingers flexion (*Palmar grasp*), all finger flexion excluding the thumb, Palmar grasp + wrist pronation, index finger extension + other digits flexion (pointing), thumb flexion + index finger flexion + middle finger flexion (3-digit pinch), 3-digit pinch + wrist pronation, thumb flexion + index finger flexion (key grasp) and key grasp + wrist pronation. Together with the single-labeled movements, this makes 66 distinct movements in total, generating an aggregate experiment duration of approximately $1h$ per test subject. A complete list of every movement of this protocol with corresponding label encoding is available in appendix A.

3.4 Preprocessing

Once acquired, all signals are filtered with a 2:nd order digital band stop Butterworth filter with lower and upper half-power frequencies set to $\frac{48 \text{ Hz}}{2048 \text{ S/s}} \approx 0.0234$ and $\frac{52 \text{ Hz}}{2048 \text{ S/s}} \approx 0.0254$, respectively, to remove power line interference. An additional 20:th order digital band pass Butterworth filter with half-power frequencies $\frac{20 \text{ Hz}}{2048 \text{ S/s}} \approx 0.0098$ and $\frac{380 \text{ Hz}}{2048 \text{ S/s}} \approx 0.1855$ is applied to suppress low and high frequency noise (e.g. movement artifacts). The filtering is done independently to each of the $2 \cdot 8 \cdot 8 = 128$ individual sEMG channels. The next step is to structure these 128 sEMG channels into 'images' (i.e. spatially structured matrices) with corresponding label sets for a CNN to operate on. This was by necessity done by a slightly different method depending on whether more than one time instant was taken into consideration for a single input data instance or not.

3.4.1 Single Time Instant Recordings

The method for structuring input data is equivalent in the single-label and multi-label case. Given the acquisition setup presented above in 3.3, the neuromuscular state of the arm at time i of sampling is represented by 2×64 numerical values with 16 bits of precision. These are reshaped into 2 8×8 matrices and concatenated vertically, generating a matrix $\mathbf{X}_i^{raw} \in \mathbb{R}^{16 \times 8}$. To generate an image $\mathbf{X}_i$ in the traditional sense, each such matrix $\mathbf{X}_i^{raw}$ is **individually** linearly rescaled to the range $[0, 255]$ and rounded to the nearest integer, where the largest and smallest elements of $\mathbf{X}_i^{raw}$ are transformed into 0 and 255, respectively, in $\mathbf{X}$. As such, a certain value of an element of $\mathbf{X}_i$ might not represent the same measured sEMG voltage for different sampling times i. Such a deliberate reduction in resolution might seem unwarranted, but actually proved to increase network performance.

Each image is thereafter paired with its corresponding label set $\mathbf{y}_i$, determined via the procedure described above in section 3.3.2, together constituting a single example $(\mathbf{X}_i, \mathbf{y}_i)$ with the notation introduced in section 2.2.1. To eliminate the subject reaction time as well as transient signal behaviour, examples originating from the first or last second of a movement repetition are discarded before proceeding, generating a total of $(5\text{s} - 2\text{s}) \cdot F_s = 6144$ examples per repetition, where $F_s = 2048\text{S/s}$ is the sampling frequency. Across all repetitions, this supplies $16 \cdot 5 \cdot 6144 = 491520$ and $66 \cdot 5 \cdot 6144 = 2027520$ examples in total in the single-label and multi-label case, respectively.

Some hand-picked examples of images generated by following the above steps are shown in fig. 3.5.

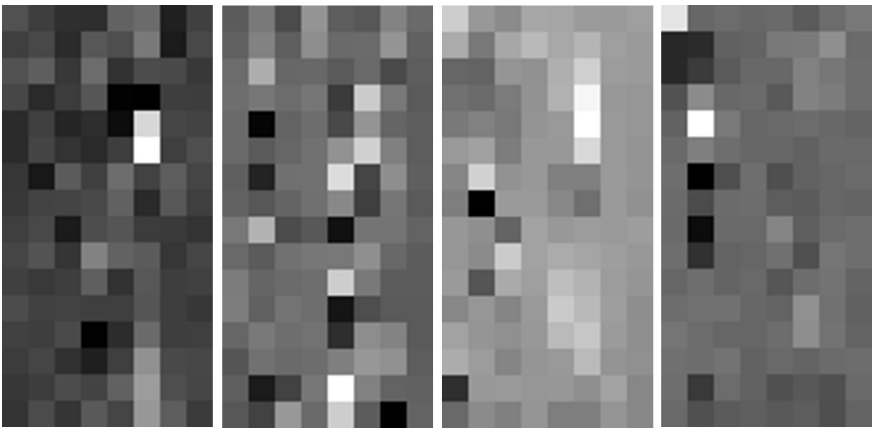


Figure 3.5: An example of sEMG-HD images to be used in classification. From left to right, the images belong to labels 1, 3, 4 and 5. In contrast to many other classification tasks, it might not be apparent how to go about manually classifying these.

3.4.2 Time-Domain Depth Recordings

For the later multi-label networks implemented in this thesis, the network input data $\mathbf{X}$ no longer represents a single time instant sample, but a sequence of 24 consecutive samples, lasting for a duration of $\frac{24-1}{F_s} \approx$11ms. The input data is thus generated by stacking 24 images, created the same way as previously, along the depth dimension, creating a tensor $\mathbf{X}_i \in \mathbb{N}^{16\times8\times24}, (1 \leq \mathbf{X}_i(j,k,l) \leq 255)\forall(j,k,l)$. The full set of such tensors is created by a sliding window with 50% overlap, or equivalently a step size of 12 samples. This entails that each time instant sample will appear in two different, consecutive tensors but at different depth. Just as earlier, the first and last second of measurements from each repetition is discarded, which results in the total number of available tensors becoming $2 \cdot \frac{2027520}{24} - 1 = 168959$. In order to determine the corresponding label set $\mathbf{y}$ for each tensor, a label-wise majority vote is taken, although this is not strictly necessary in this setup as every window only extends over a uniform sequence of label sets.

3.5 Single-Label Networks

For both single-label CNNs considered below, the loss function in use was cross-entropy loss, and for optimization the SGDM algorithm with $\alpha = 0.1$, $m = 0.9$ and L2 normalization with $\lambda = 0.0001$ was adopted (hyperparameters determined empirically). Both networks were separately trained and

tested on the CapgMyo data set and on the original data collected as described previously in section 3.3.1. The training was done in batch with a batch size of 1000, for 30 epochs with CapgMyo data and for 100 epochs for the original data. The exception to this was training of the first simple network, which lasted for 20 epochs for both data sets. For easier convergence, the learning rate α was reduced by a factor 10 after the 10th, 20th, 30th and 40th epoch. The available examples were split into a training and testing set of equal cardinality; no validation data was used in this initial experiment. Partitioning into training and testing sets was done by splitting each label temporally, so that the first half of measurements for each movement constituted the training data, and the second half testing data. For CapgMyo, this entailed using the first half of the sessions (1-5) for training and the second half of session (6-10) for testing. Only data from subject 1 was used here. Prior to training, the test set is shuffled once, so that every batch contains an acceptable mix of labels. As the networks perform single-label classification, only accuracy was considered as a performance measure.

3.5.1 Network Architecture

The first, very simple CNN to be tested was implemented in MATLAB, and contained only the input layer, a convolutional layer with 64 3×3 filters (stride 1, zero padding 1), a ReLU activation layer, a fully connected layer 16 outputs for and the final softmax layer. As such, this network contains only a single hiddel layer. Training of this network lasted for 20 epochs. For use with CapgMyo, the number of fully connected outputs had to be replaced with 8.

After this initial modest attempt, a more extensive CNN, with architecture inspired by [1], was created. An illustration of this network, for this work realized in MATLAB, is shown in fig. 3.6. Secondary to the input layer, 4 blocks of the form conv-batchNorm-ReLU follows; the first two convolutional layers with 64 kernels of width and breadth 3 and the last two with kernels of width and breadth 1. While a 1×1 kernel might seem to just perform pixel-wise rescaling, this is not the case as the kernel is fully connected along the depth dimension. These blocks are succeeded by 3 blocks of the form dropout-FC-batchNorm-ReLU, where the fully connected layers are of output size 512, 512 and 128. Lastly, as is usually the case for single-label networks, comes a fully connected layer of the same output size as the label set (8 for CapgMyo and 16 for the original data) together with a softmax activation layer. During training, this is followed by the cross-entropy loss which in turn feeds into the SGDM by backpropagation. During testing, this is followed by prediction, where the index of the largest element in the softmax output (or equivalently, the output of the last fully connected layer) is the network label prediction.

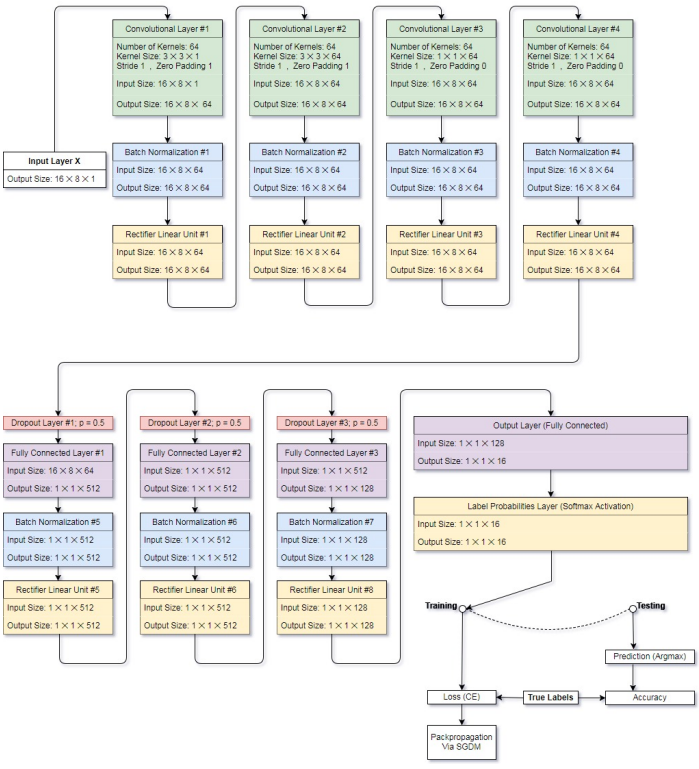


Figure 3.6: Flow chart of the architecture of the second single-label CNN. The total number of learnable parameters is 4572496.

3.5.2 Pixel Contribution Quantification

If the methods devised in this thesis is to be used in a real prosthesis control application, it would be of interest to be able to reduce the number of sEMG channels necessary. This is motivated both by the reduction in processing power needed to perform the classification, but also by the infeasibility for a portable device to measure 128 channels with the same fidelity as the stationary setup employed here. Because of this, a measure of channel importance, denoted as the *Pixel Contribution Matrix* $\mathbf{Q}$ is proposed here. Once the network has finished training, the training set is fed into the network once again. $\mathbf{Q}$ is then calculated for each image via eq. 3.1.

$$Q(x,y) = \sum_{j=1}^{q} \|\hat{\phi}(\mathbf{X})\{j\} - \hat{\phi}(\mathbf{X}^*_{x,y})\{j\}\| \tag{3.1}$$

Here, $\mathbf{X}^*_{x,y}$ is the input image $\mathbf{X}$, but modified so that $X(x,y) = 0$. As in chapter 2, q is the number of labels, here equal to 8 for CapgMyo and

16 for the original data, and $\hat{\phi} : \mathbb{R}^{\mathbf{d}} \longrightarrow [0,1]^q$ is the trained network output prior to classification, i.e. the activations of the softmax layer. As $\mathbf{Q}$ varies depending on input $\mathbf{X}$, its element-wise average is calculated over the entirety of the training data. To investigate how the removal of less important channels affect network performance, 50% of the pixels of each image in both the training and testing set were set to 0. To determine which pixels were selected for removal, a binary mask was created via element-wise thresholding of the averaged $\mathbf{Q}$, were the threshold is set to its median. After element-wise multiplication with this mask, the network is retrained with the reduced training set, and new performance measures (i.e. accuracy) are obtained from the reduced testing set. As this is only tangentially relevant to the main goals of this thesis, and some work would have to be done to adapt this method for multi-labeled time windowed data, this procedure was only applied to the second, multi-layered single-label CNN, but for both the CapgMyo and original data.

3.6 Multi-Label Networks

For classifying multi-labeled movements, two approaches were taken: multiple binary single-label CNNs and single multi-label CNN. ALL networks were developed, trained and tested with the data acquired during the same measurement sessions (see section 3.3.2) from 3 different subjects. In contrast to previously, a validation set was incorporated, as described in section 2.2.6.3. Measurements from the 2:nd and 3:rd repetition of each movement were used for training, from the 4:th for testing and from the 5:th for validation. The 1:st repetition of each movement was discarded, as it for all subjects happened once or more that the subject did not switch movement and instead performed the previous movement through the first repetition of the subsequent movement.

3.6.1 Single Time Instant Architectures

3.6.1.1 Multiple Networks

The first approach to expanding the model into the domain of multi-label classification was done in MATLAB with 16 separate networks, each identical to fig. 3.6 with the exception of the output size of the final fully connected layer, which was reduced to 2. Each network is trained separately, and with the sole purpose of detecting the relevance or irrelevance of a single label each. In this setup, the same training hyperparameters as in section 3.5 were used, but with the number of epochs reduced to 10. The validation frequency was set to once per epoch, with a validation patience of 3.

3.6.1.2 Single Network

The multi-label single network models utilized here were both implemented in Python, making use of the TensorFlow library, as MATLAB did not offer the relevant functionality needed. The first multi-label CNN was implemented very much akin to the second single-labeled network; and illustration is available in fig. 3.7. The main differences are the residual blocks, and naturally the choice of final activation layer. For optimization, both the cross-entropy and BP-MLL were tested, with sigmoid and tanh activation, respectively. In training, the validation frequency was set to every second epoch with a validation patience of 5. If no early stopping was performed in this way, the maximum number of allowed epochs was set to 30. For optimization, the Adam algorithm with $\alpha = 0.03$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ was selected together with L2 regularization with $\lambda = 0.0065$ and a batch size of 1000. As earlier, these values were arrived at empirically. During testing, all measures introduced in section 2.3.1 are calculated. As the classification depend on the chosen activation threshold, performance measures were computed for a range of uniformly sampled thresholds from the interval $[0, 1]$ in the case of cross-entropy loss and sigmoid activation, and $[-1, 1]$ in the case of BP-MLL and tanh activation. The results are used to plot the precision-recall curve, while the other measures are reported as the ones computed from the threshold that yielded maximum subset accuracy.

3.6.2 Time Domain Features Architecture

For the multi-label CNN to be able to formulate spatiotemporal features from the data, some slight modifications were made to the network in fig. 3.7. The full extent of these modification can be viewed in the network flowchart illustration in fig. 3.8. Most notably, the number of kernels in the first layer was increased from 64 to 128, and by necessity the depth of these kernels were increased from 1 to 24. The motivation for this interference is that as the number of elements in the input data $\mathbf{X}$ has increased, it should follow that a greater number of features is needed to represent it in a descriptive way. The choice of hyperparameters and evaluation procedure here was the same as in the single time instant scenario; the single difference was the choice of batch size and maximum number of allowed epochs, here incremented to 3000 and 250, respectively.

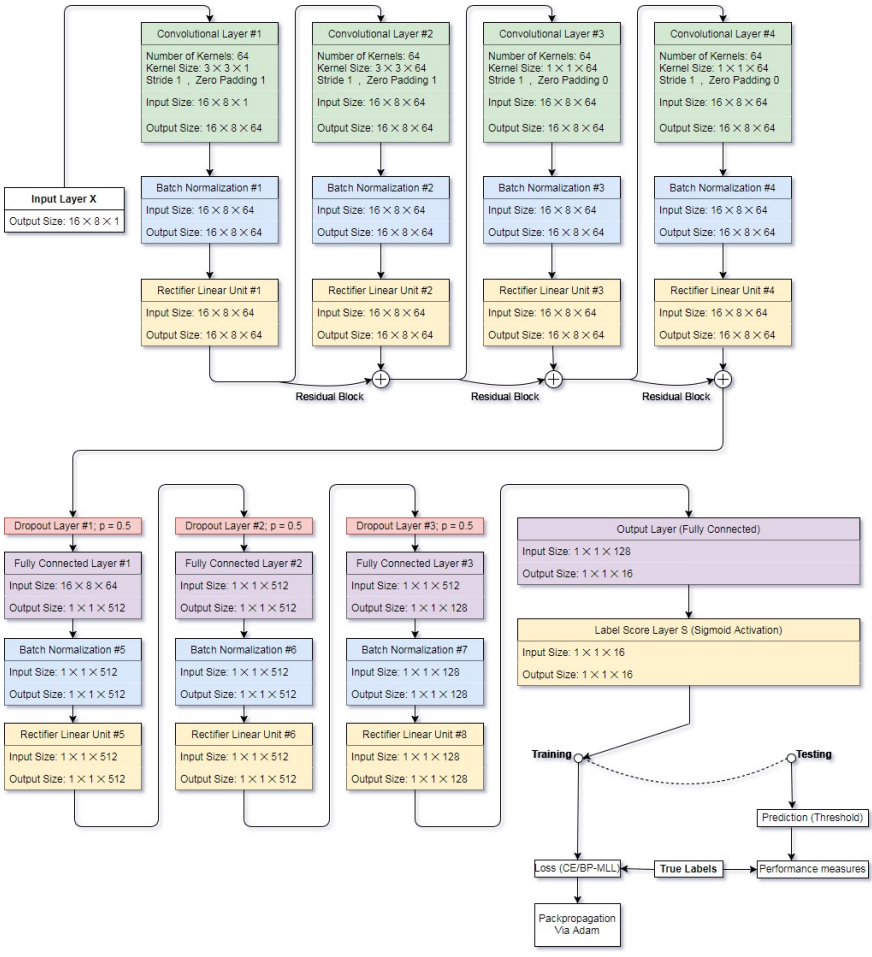


Figure 3.7: Flow chart of the architecture of the single time instant multi-label CNN. The total number of learnable parameters in this model is 4572496.

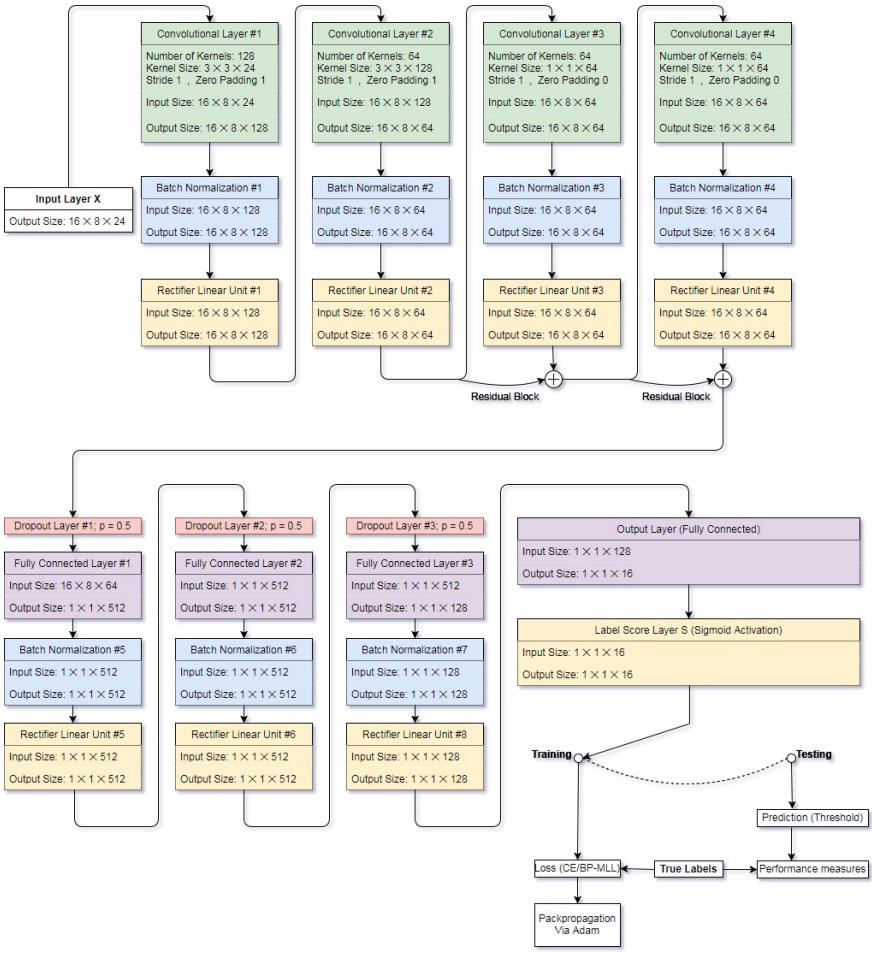


Figure 3.8: Flow chart of the architecture of the time window multi-label CNN. The total number of learnable parameters in this model is 4636560.

Chapter 4

Results

This chapter contains the results obtained from following the procedures outlined in chapter 3, intermingled with some sparse commentary. For further discussion and interpretation of the results, proceed to chapter 5.

4.1 Single-Label

Here follows the results yielded from training and testing both of the single-label networks as described in section 3.5. As the first, single hidden layer CNN acted solely as a proof of concept, graphs of its training progress are omitted.

4.1.1 CapgMyo Data

With the single hidden layer CNN trained and tested with the CapgMyo data set, an accuracy of 0.845 was achieved. When the network was expanded to the second, deeper version (fig. 3.6) but applied to the same data, the accuracy was increased to 0.895. The training progress (loss and accuracy) of this network is shown in fig. 4.1.

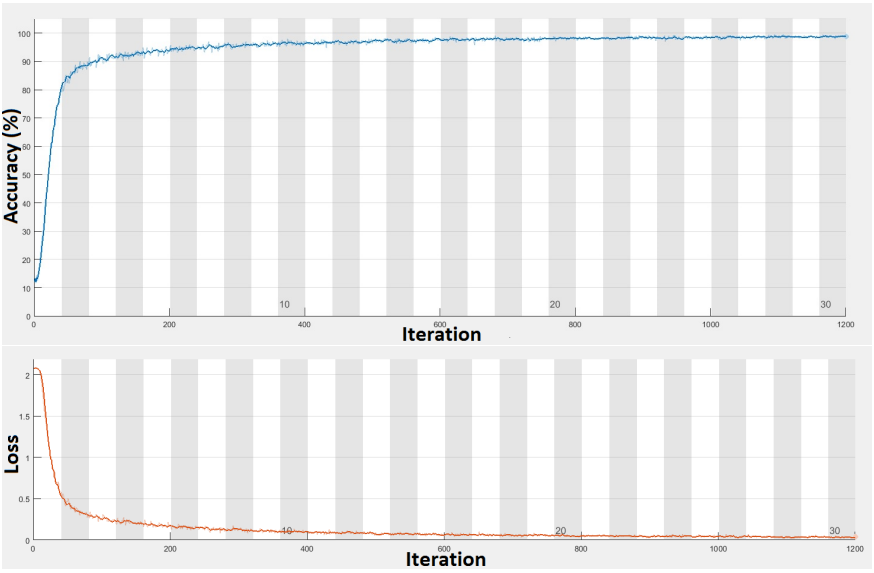


Figure 4.1: (top) The iteration accuracy during training of the second single-label CNN with CapgMyo data. (bottom) The corresponding averaged batch loss during training.

4.1.1.1 Pixel Contribution Quantification

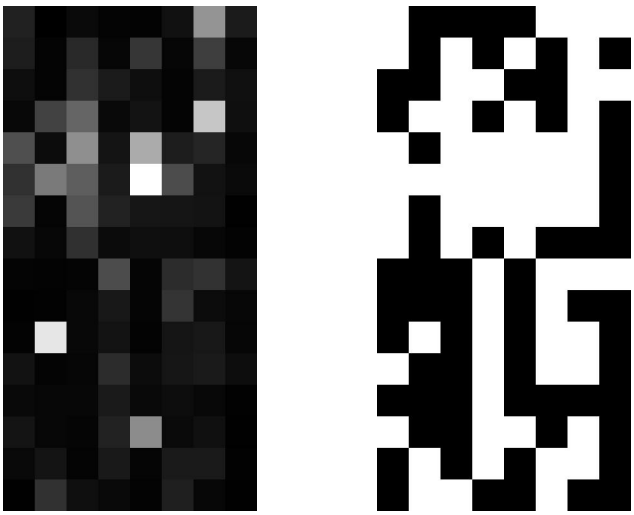


Figure 4.2: (left) Pixel contribution matrix **Q** obtained from the CapgMyo data. (right) The 50% removal binary mask created from median thresholding of **Q**. Black pixels are set to zero before retraining and retesting.

A gray-scale image representation of the pixel contribution matrix **Q**, defined in eq. 3.1 and obtained from the network in fig. 3.6 trained on CapgMyo data, is shown in fig. 4.2 alongside the corresponding binary mask. When the CapgMyo data was filtered with this mask and the network retrained and retested with identical setup as for the pristine data, an accuracy of 0.831 was obtained.

4.1.2 Original Data

When trained and tested on the original data, the single hidden layer CNN achieved an accuracy of 0.6214. With the same data, the network in fig. 4.1 improved this to a final accuracy of 0.742. The training progress of this later case can be viewed in fig. 4.3.

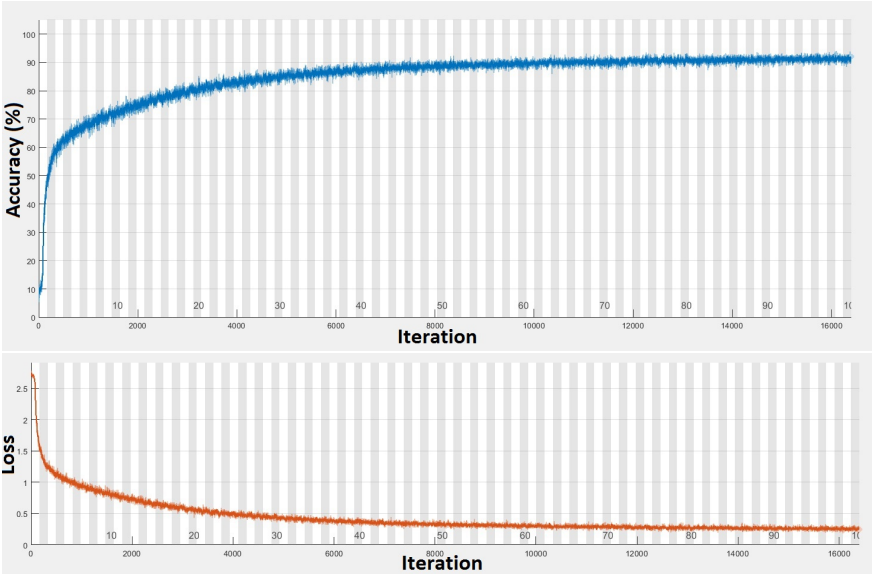


Figure 4.3: (top) The iteration accuracy during training of the second single-label CNN on original data. (bottom) The corresponding averaged batch loss during training.

4.1.2.1 Pixel Contribution Quantification

Just as for the CapgMyo data, the pixel contribution matrix **Q** with corresponding binary mask were computed and are shown in fig. 4.4. The accuracy after training and testing with reduced data amounted to 0.669.

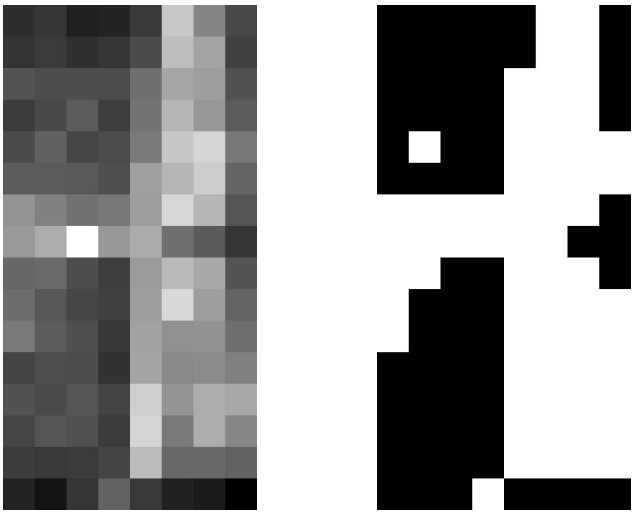


Figure 4.4: (left) Pixel contribution matrix **Q** obtained from the original data. (right) The 50% removal binary mask created from median thresholding of **Q**. Black pixels are set to zero before retraining and retesting.

4.2 Multi-label

In the following sections, results obtained from use of the different multi-label CNNs are presented. In both the single time instant and time window single network models, results from training with cross-entropy loss and BP-MLL are presented separately and for each subject individually. To keep the layout comprehensive, only representative examples of training progress plots and precision-recall curves, one from each method, are shown here. For a systematic exhibition of all such graphs obtained, see appendix B.

4.2.1 Multiple Networks

The results from using 16 separate binary classification networks are visible in tab. 4.1. As there is no tunable sensitivity threshold in these networks, no precision-recall curve can be generated.

Subject	1	2	3
Average final epoch	7.81	7.25	8.69
Subset accuracy	0.223	0.449	0.349
Hamming loss	0.097	0.060	0.075
Jaccard Index	0.400	0.606	0.538
Precision	0.672	0.807	0.731
Recall	0.497	0.709	0.671

Table 4.1: Performance measures for the multiple binary networks approach. As there are 16 separate networks, the final epoch is reported as the average over all networks.

4.2.2 Single Time Instant Network

4.2.2.1 Cross-Entropy Loss

Here follows the results obtained from utilizing the multi-label single time instant CNN depicted in fig. 3.7 trained with cross-entropy loss. Tab. 4.2 contains all performance measures for each subjects, while fig. 4.5 depict a representative precision-recall curves from subject 2. Fig. 4.6 show the training progress of the network the same subject. The full set of resulting images pertaining to this specific method can be viewed in fig. B.1-B.6 of appendix B.

Subject	1	2	3
Final epoch	20	30	30
Optimal threshold	0.34	0.42	0.42
Subset accuracy	0.229	0.503	0.342
Hamming loss	0.105	0.057	0.085
Jaccard Index	0.479	0.686	0.56
Precision	0.596	0.790	0.680
Recall	0.615	0.770	0.651

Table 4.2: Performance measures of the multi-label single time instant CNN trained with cross-entropy loss. Here final epoch denotes the epoch during which the training was aborted, by validation checking or by reaching the maximum number of allowed epochs. Optimal threshold is the threshold for the final activation layer which generated the highest subset accuracy.

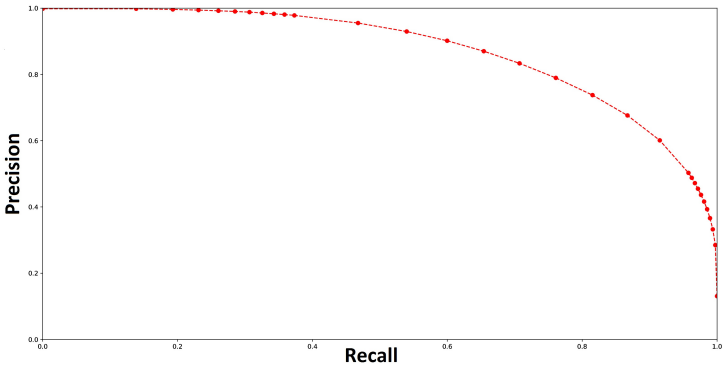


Figure 4.5: Precision-recall curve acquired from the multi-label single time instant CNN trained with cross-entropy loss on data for subject 2.

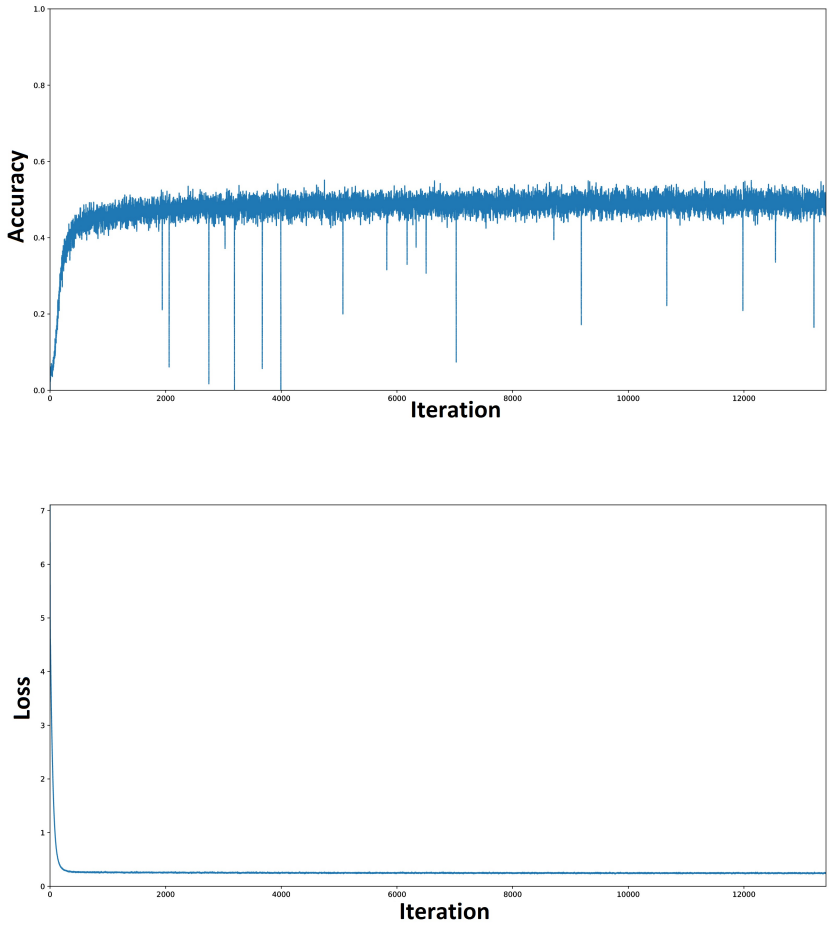


Figure 4.6: (top) Multi-label single time instant CNN iteration accuracy when training with cross-entropy loss on data from subject 2 (bottom) Corresponding iteration loss

4.2.2.2 BP-MLL loss

In this section, the results from utilizing the multi-label single time instant CNN depicted in fig. 3.7 with BP-MLL selected as loss function for training. In the same way as in the previous section, tab. 4.3 contains the performance measures, fig. 4.7 present a representative precision-recall curve and fig. 4.7 show the CNN training progress for the same subject. All subject-specific plots are compiled in fig. B.7-B.12 in appendix B. Note that the optimal

threshold domain is $[-1, 1]$ as the final activation layer is of the tanh kind.

Subject	1	2	3
Final epoch	20	30	30
Optimal threshold	0.86	0.90	0.92
Subset accuracy	0.182	0.412	0.272
Hamming loss	0.103	0.0.065	0.089
Jaccard Index	0.45	0.652	0.542
Precision	0.629	0.762	0.674
Recall	0.514	0.733	0.625

Table 4.3: Performance measures of the multi-label single time instant CNN trained with BP-MLL loss. Here final epoch denotes the epoch during which the training was aborted, by validation checking or by reaching the maximum number of allowed epochs. Optimal threshold is the threshold for the final activation layer which generated the highest subset accuracy.

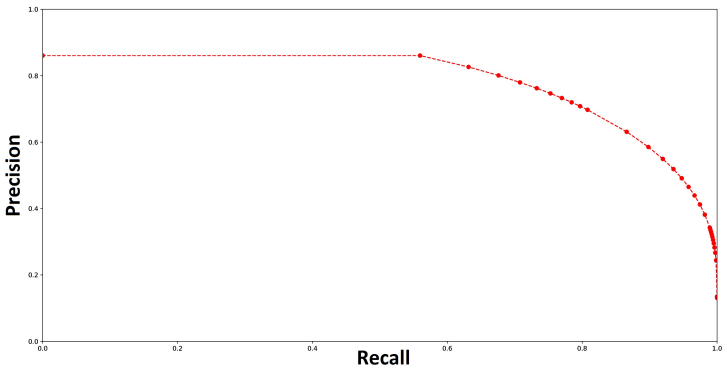
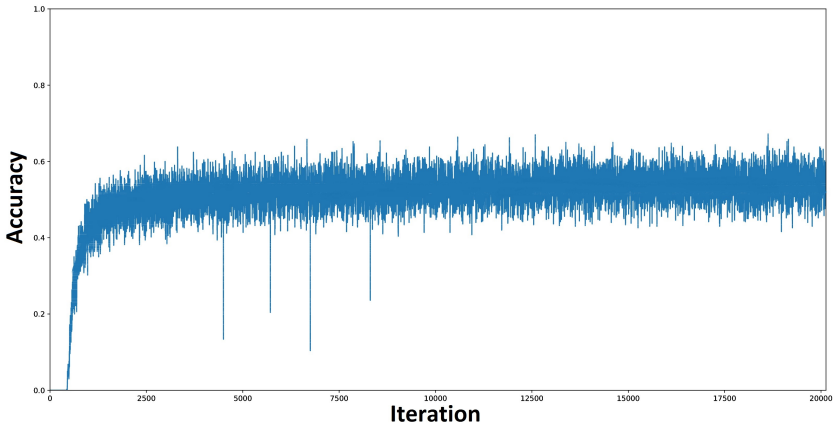


Figure 4.7: Precision-recall curve acquired from the multi-label single time instant CNN trained with BP-MLL loss on data for subject 2.



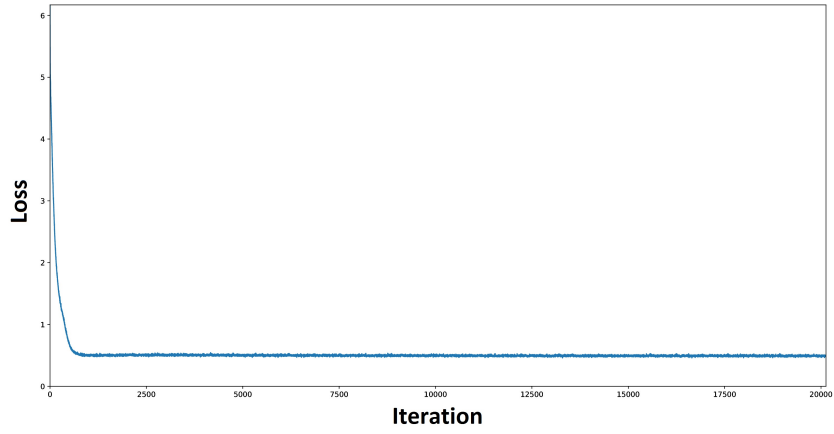


Figure 4.8: (top) Multi-label single time instant CNN iteration accuracy when training with BP-MLL loss on data from subject 2. (bottom) Corresponding iteration loss

4.2.3 Time Domain Features Network

4.2.3.1 Cross-Entropy Loss

Presented below are the results from using the multi-label time depth CNN from fig. 3.8 trained with cross-entropy loss, as described in section 3.6.2. The joint results from all subjects are presented in tab. 4.4. Corresponding precision-recall curves are shown in fig. B.13-B.3 and the training progress that led to these results are shown in fig. B.16-B.18 in appendix B. Representative samples of a precision-recall curve and a training progress plot are shown below in fig. 4.9 and fig. 4.10, respectively.

Subject	1	2	3
Final epoch	250	250	250
Optimal threshold	0.42	0.50	0.50
Subset accuracy	0.577	0.806	0.759
Hamming loss	0.055	0.024	0.030
Jaccard Index	0.727	0.871	0.846
Precision	0.792	0.913	0.891
Recall	0.781	0.905	0.875

Table 4.4: Performance measures of the multi-label time depth CNN trained with cross-entropy loss. Here final epoch denotes the epoch during which the training was aborted, by validation checking or by reaching the maximum number of allowed epochs. Optimal threshold is the threshold for the final activation layer which generated the highest subset accuracy.

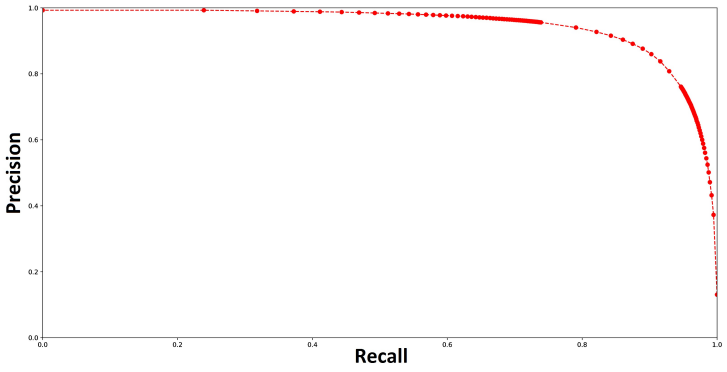


Figure 4.9: Precision-recall curve acquired from the multi-label time depth CNN trained with cross-entropy loss on data for subject 2.

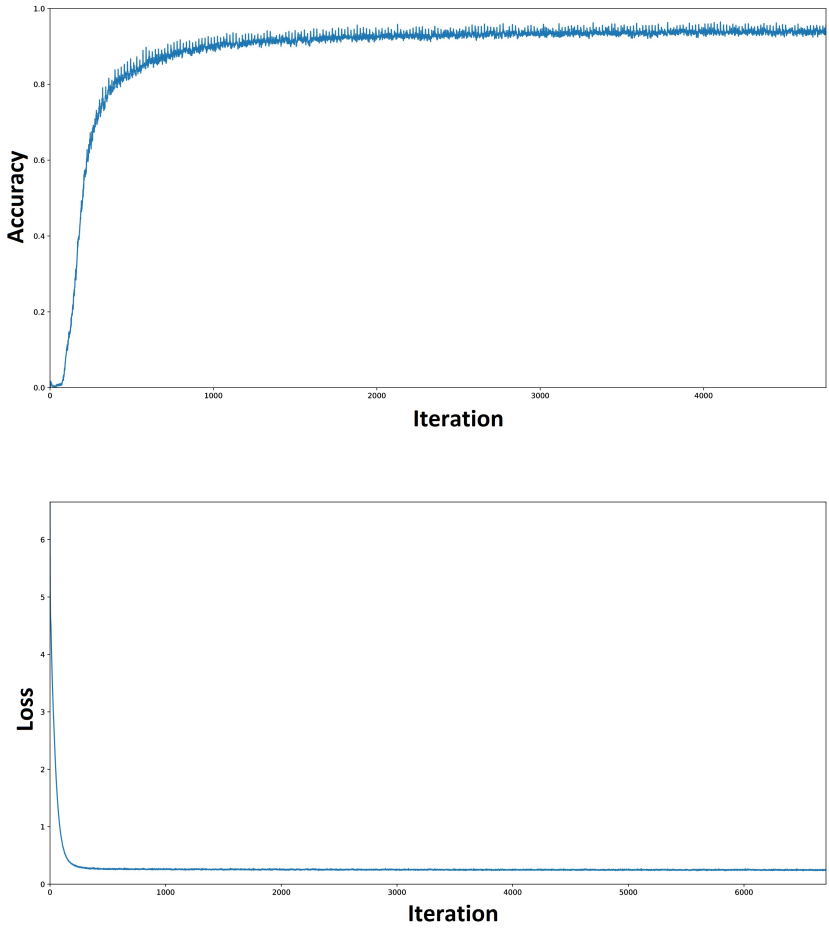


Figure 4.10: (top) Multi-label time depth CNN iteration accuracy when training with cross-entropy loss on data from subject 2 (bottom) Corresponding iteration loss.

4.2.3.2 BP-MLL Loss

In this chapter-ending section, the results from the CNN in fig. 3.8 trained with BP-MLL loss are described. Tab. 4.5 contains the compiled performance measures, fig. B.19-B.21 shows the precision recall curves of each subject while finally fig. B.22-B.24 depicts their respective training progress. As previously, representative examples are shown in fig. 4.11 and fig. 4.12.

Subject	1	2	3
Final epoch	250	250	250
Optimal threshold	0.93	0.95	0.94
Subset accuracy	0.502	0.700	0.681
Hamming loss	0.060	0.031	0.037
Jaccard Index	0.697	0.825	0.809
Precision	0.774	0.890	0.858
Recall	0.762	0.866	0.860

Table 4.5: Performance measures of the multi-label time depth CNN trained with BP-MLL loss. Here final epoch denotes the epoch during which the training was aborted, by validation checking or by reaching the maximum number of allowed epochs. Optimal threshold is the threshold for the final activation layer which generated the highest subset accuracy.

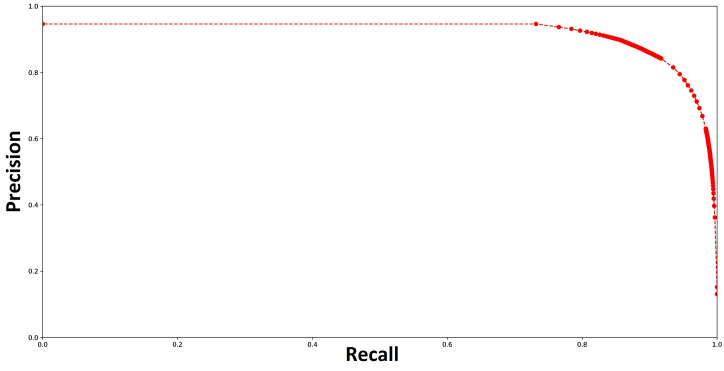
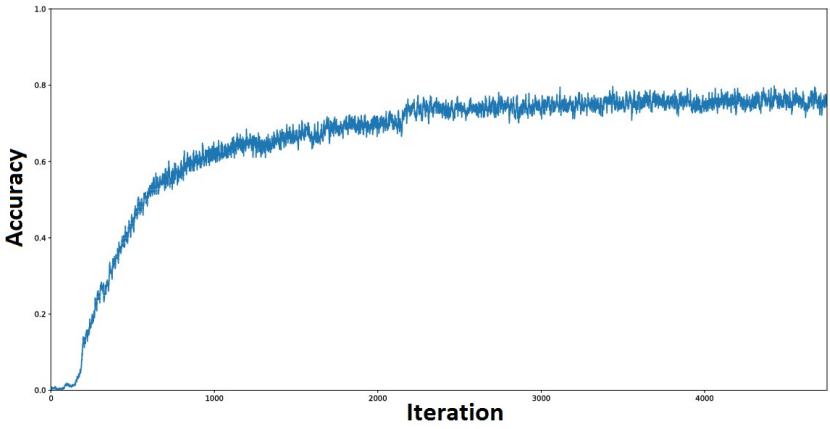


Figure 4.11: Precision-recall curve acquired from the multi-label time depth CNN trained with BP-MLL loss on data for subject 2



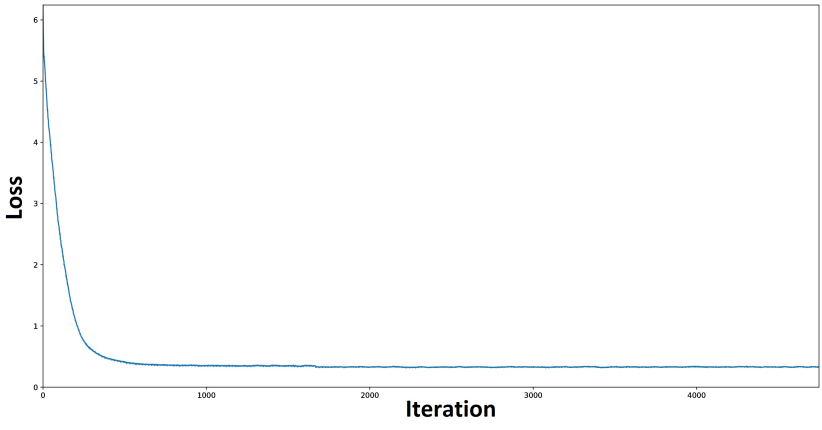


Figure 4.12: (top) Multi-label time depth CNN iteration accuracy when training with BP-MLL loss on data from subject 2 (bottom) Corresponding iteration loss

Chapter 5

Conclusions and Discussion

The aim of the work carried out here was to investigate the possible application of multi-label convolutional neural networks for HD-sEMG classification. This was done with the specific application of hand prostheses in mind. While single-label classification of the individual digit movements might be useless for practical purposes (a prosthetic hand that can only move a single finger at the time is not very helpful), it still offers insight into the feasibility of the multi-label endeavour. As is shown in section 4.1, the sEMG signal clearly possesses extractable patterns relating to the current digit or wrist state, even at the single time-instant level. Exactly what these patterns are remains unknown, and would require further physiological inquiry to find out. It was also shown that the descriptive qualities of the HD-sEMG signal is not uniformly distributed across the input channels, and an approach with fewer electrodes could prove feasible in a real application: In sections 4.1.1.1 and 4.1.2.1, the reduction of accuracy when removing the channels below the 50:th importance percentile amounted to 6.4% and 7.3%, respectively. In fig. 4.4, a clear pattern emerged where the rightmost pixels, corresponding to electrodes placed further towards the radial and ulnar side of the forearm for each matrix, contributed the most in classification. No equally clear pattern emerged from CapgMyo data, which used a completely different electrode setup.

In multi-label classification, inter-subject variations in performance measures proved quite large, as can be seen in tab. 4.2, 4.3, 4.4 and 4.5. This conforms well to the reported experience of the subjects, where data from those who reported making more mistakes were associated with worse performance measures. From these tables, it can in all cases be shown that the network actually learns interlabel dependencies in the raw data, as

subset accuracy $>$ (1 $-$ Hamming loss)^{16}, i.e. the CNN adapts its prediction depending on which set of labels it estimates is present. From the same tables, it is readily apparent that expanding the data along the time dimension improved performance quite significantly. There are some ambiguities regarding the choice of loss function, as the data is somewhat inconclusive. While cross-entropy loss always generated superior performance measures for all subjects, precision recall curves show slightly more robustness for BP-MLL, most saliently for subject 3 (compare fig. B.15 and fig. B.21). In addition, when comparing the training characteristics in fig. B.4-B.6, B.10-B.12, B.16-B.18 and B.22-B.24 with the obtained subset accuracy, it is observed that the cross-entropy loss based training systematically overfits the CNN, although not to the extent required to trigger validation abortion. This is however not the case for BP-MLL loss based training, which achieves similar results on both the training and testing data. In an intersession environment, when a long time elapses between training and testing, BP-MLL loss could prove superior. Compared to the results of multiple networks approach (tab. 4.1), the single network approach proved systematically superior. This is postulated to be caused by the inability to learn how the relevance of some labels might affect the patterns associated with the relevance of other labels, despite the multiple networks approach being almost 16 times more computationally expensive compared to the single network counterparts.

It is not a trivial task to compare the results obtained here and those obtained from similar endeavours elsewhere. To the best knowledge of yours truly, there exists no published work in which multi-label classification has been applied to EMG signals, let alone via convolutional neural networks. To be able to attain any notion of how the approaches attempted here fare, one must turn to the more extensive literature on single-label EMG classification. Conventional wisdom dictates that as classification accuracy decreases rapidly with the the number of possible classes [3], it is only fair to compare classification methods of similar class set cardinality. The number of unique 'classes' possible here is equal to $2^{16} = 65536$, massively more than that of other works found. Another issue, as discussed in section 2.3.1, is what measures to compare. Subset accuracy might skew perception negatively, as even failed testing examples might be partially correct. Even at this disadvantage, the best methods employed here could generate subset a accuracy of more than 0.8; highly comparable to the results of the works of others, e.g. [35],[36] and [37].

5.1 Future Work

While the networks applied in this thesis proved quite effective, their architecture is still somewhat arbitrary. In the space of possible and plausible CNNs, it is not at all obvious that the networks proposed here are close to optimal. Hence, it could be of interest to investigate a *genetic algorithm* [38] approach, in which many different networks are iteratively tried and the superior ones extended. Granted, this would require significant computing power to be done in a reasonable time span.

One aspect that could be expanded upon is the measurement setup, specifically electrode placement. As was shown in this thesis, different channels contribute differently to the performance of the classifier. Because of this, it could be of interest to systematically vary electrode placement and evaluate which setups prove most effective.

One problem which was ignored in this thesis was whether networks trained with data from one session could be used at a different session, at a different time with slightly different electrode placement. For a network to be able to function is such an environment, it is likely some fine tuning would have to be done for each session, and some methodology for performing this would be necessary.

A related approach taken to the one presented in this thesis would be to move from classification to *regression* of individual muscle forces. This would eliminate the impact of erroneous subject behaviour, as the muscle forces would be measured separately and no assumption of currently relevant labels would be needed

Chapter 6

Bibliography

[1] Y. Du, W. Jin, W. Wei, Y. Hu, and W. Geng, "Surface EMG-based inter-session gesture recognition enhanced by deep domain adaptation," *Sensors*, vol. 17, no. 3, p. 458, 2017.

[2] A. Phinyomark, C. Limsakul, and P. Phukpattaranont, "A novel feature extraction for robust emg pattern recognition.," *Jornal of Computing*, vol. 1, 2009.

[3] M. Atzori, M. Cognolato, and H. Müller, "Deep learning with convolutional neural networks applied to electromyography data: A resource for the classification of movements for prosthetic hands.," *Frontiers in Neurorobotics*, vol. 10, no. SEP, 2016.

[4] D. Kingma and J. Ba, "Adam: A method for stochastic optimization.," *arXiv*, 2014.

[5] M.-L. Zhang and Z.-H. Zhou, "Multi-label neural networks with applications to functional genomics and text categorization.," *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, 2006.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition.," *2016 IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[7] K. R. Mills, "The basics of electromyography.," *Journal Of Neurology, Neurosurgery, And Psychiatry*, vol. 76 Suppl 2, pp. ii32 – ii35, 2005.

[8] R. H. Chowdhury, M. B. I. Reaz, M. A. B. M. Ali, A. A. A. Bakar, K. Chellappan, and T. G. Chang, "Surface electromyography signal processing and classification techniques," *Sensors (Basel, Switzerland) 13(9)*, p. 12431–12466, 2013.

[9] M. J. Hoozemans and J. H. van Dieën, "Prediction of handgrip forces using surface emg of forearm muscles.," *Journal of Electromyography and Kinesiology*, vol. 15, pp. 358 – 366, 2005.

[10] T. Kohonen, "An introduction to neural computing.," *Neural Networks*, vol. 1, pp. 3 – 16, 1988.

[11] W. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity.," *Bulletin of Mathematical Biophysics*, vol. 5, no. 4, p. 115, 1943.

[12] Image modified from `http://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-1/7`. Accessed: 2017-12-12.

[13] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems 2* (D. S. Touretzky, ed.), pp. 396–404, Morgan-Kaufmann, 1990.

[14] 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.

[15] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and L. Qianli, "Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review.," *International Journal of Automation and Computing*, vol. 14, no. 5, pp. 503 – 519, 2017.

[16] Image modified from `https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html`. Accessed: 2017-12-14.

[17] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift.," *arXiv*, 2015.

[18] H. Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function.," *Journal of Statistical Planning and Inference*, vol. 90, pp. 227 – 244, 2000.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks.," *Communications of the ACM*, vol. 60, no. 6, pp. 84 – 90, 2017.

[20] C. M. Bishop, *Pattern recognition and machine learning.* Information science and statistics, Springer, 2006.

[21] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop.," *Neural Networks: Tricks of the Trade*, pp. 9 – 48, 2012.

[22] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[23] L. Bottou, "Online algorithms and stochastic approximations," in *Online Learning and Neural Networks* (D. Saad, ed.), Cambridge, UK: Cambridge University Press, 1998. revised, oct 2012.

[24] D. Rumelhart, R. Williams, and G. Hinton, "Learning representations by back-propagating errors.," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[25] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *JOURNAL OF MACHINE LEARNING RESEARCH*, vol. 12, pp. 2121 – 2159, 2011.

[26] T. Tieleman and G. Hinton, "Lecture 6.5 - rmsprop, coursera: Neural networks for machine learning.," 2012.

[27] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning requires rethinking generalization.," *arXiv*, 2016.

[28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting.," *Journal of Machine Learning Research. Vol. 15*, pp. 1929–1958, 2014.

[29] M.-L. Zhang and Z.-H. Zhou, "Ml-knn: A lazy learning approach to multi-label learning.," *Pattern Recognition*, vol. 40, pp. 2038 – 2048, 2007.

[30] M.-L. Zhang, J. M. Peña, and V. Robles, "Feature selection for multi-label naive bayes classification.," *Information Sciences*, vol. 179, pp. 3218 – 3229, 2009.

[31] A. Elisseeff and J. Weston, "A kernel method for multi-labelled classification," in *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, (Cambridge, MA, USA), pp. 681–687, MIT Press, 2001.

[32] Y. Wei, W. Xia, M. Lin, J. Huang, B. Ni, J. Dong, Y. Zhao, and S. Yan, "Hcp: A flexible cnn framework for multi-label image classification.," *IEEE Transactions on Pattern Analysis and Machine Intelligence, Pattern Analysis and Machine Intelligence, IEEE Transactions on, IEEE Trans. Pattern Anal. Mach. Intell*, p. 1901, 2016.

[33] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks.," *Information Processing and Management*, vol. 45, pp. 427 – 437, 2009.

[34] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Good-

fellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems.," *arXiv*, 2016.

[35] W. Geng, Y. Du, W. Jin, W. Wei, Y. Hu, and J. Li, "Gesture recognition by instantaneous surface emg images.," *Scientific Reports*, vol. 6, p. 36571, 2016.

[36] C. Amma, T. Krings, J. Böer, and T. Schultz, "Advancing muscle-computer interfaces with high-density electromyography," in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, (New York, NY, USA), pp. 929–938, ACM, 2015.

[37] M. Atzori, A. Gijsberts, C. Castellini, B. Caputo, A.-G. M. Hager, S. Elsig, G. Giatsidis, F. Bassetto, and H. Müller, "Electromyography data for non-invasive naturally-controlled robotic hand prostheses," *Scientific data*, vol. 1, p. 140053, 2014.

[38] M. Mitchell, *An introduction to genetic algorithms.* Complex adaptive systems, Cambridge, Mass. : MIT Press, cop. 1996, 1996.

Appendices

Appendix A

Multi-Label Acquisition Protocol

In this appendix, the full list of movements recorded for multi-label presentation is collected. As described in sections 3.3.1 and 3.3.2, each movement is performed for 5s, repeated 5 times with 5s of rest in-between.

Movement	Corresponding label set $\mathbf{y}$
Rest	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
Little finger flexion	[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
Little finger extension	[0,1,0,0,0,0,0,0,0,0,0,0,0,0,0]
Ring finger flexion	[0,0,1,0,0,0,0,0,0,0,0,0,0,0,0]
Ring finger extension	[0,0,0,1,0,0,0,0,0,0,0,0,0,0,0]
Middle finger flexion	[0,0,0,0,1,0,0,0,0,0,0,0,0,0,0]
Middle finger extension	[0,0,0,0,0,1,0,0,0,0,0,0,0,0,0]
Index finger flexion	[0,0,0,0,0,0,1,0,0,0,0,0,0,0,0]
Index finger extension	[0,0,0,0,0,0,0,1,0,0,0,0,0,0,0]
Thumb flexion	[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0]
Thumb extension	[0,0,0,0,0,0,0,0,0,1,0,0,0,0,0]
Thumb abduction	[0,0,0,0,0,0,0,0,0,0,1,0,0,0,0]
Thumb adduction	[0,0,0,0,0,0,0,0,0,0,0,1,0,0,0]
Wrist flextion	[0,0,0,0,0,0,0,0,0,0,0,0,1,0,0]
Wrist extension	[0,0,0,0,0,0,0,0,0,0,0,0,0,1,0]
Wrist pronation	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]
Wrist supination	[0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]
Little finger flexion + Ring finger flexion	[1,1,0,0,0,0,0,0,0,0,0,0,0,0,0]
Little finger flexion + Thumb down	[1,0,0,0,0,0,0,0,1,0,0,0,0,0,0]
Little finger flexion + Thumb abduction	[1,0,0,0,0,0,0,0,0,0,1,0,0,0,0]

Little finger flexion + Thumb adduction	[1,0,0,0,0,0,0,0,0,0,1,0,0,0,0]
Little finger flexion + Wrist flextion	[1,0,0,0,0,0,0,0,0,0,0,1,0,0,0]
Little finger flexion + Wrist extension	[1,0,0,0,0,0,0,0,0,0,0,0,1,0,0]
Little finger flexion + Wrist pronation	[1,0,0,0,0,0,0,0,0,0,0,0,0,1,0]
Little finger flexion + Wrist supination	[1,0,0,0,0,0,0,0,0,0,0,0,0,0,1]
Ring finger flexion + Middle finger flexion	[0,0,1,0,1,0,0,0,0,0,0,0,0,0,0]
Ring finger flexion + Thumb down	[0,0,1,0,0,0,0,0,1,0,0,0,0,0,0]
Ring finger flexion + Thumb abduction	[0,0,1,0,0,0,0,0,0,0,1,0,0,0,0]
Ring finger flexion + Thumb adduction	[0,0,1,0,0,0,0,0,0,0,0,1,0,0,0]
Ring finger flexion + Wrist flextion	[0,0,1,0,0,0,0,0,0,0,0,0,1,0,0]
Ring finger flexion + Wrist extension	[0,0,1,0,0,0,0,0,0,0,0,0,0,1,0]
Ring finger flexion + Wrist pronation	[0,0,1,0,0,0,0,0,0,0,0,0,0,0,1]
Ring finger flexion + Wrist supination	[0,0,1,0,0,0,0,0,0,0,0,0,0,0,0]
Middle finger flexion + Index finger flexion	[0,0,0,0,1,0,1,0,0,0,0,0,0,0,0]
Middle finger flexion + Thumb down	[0,0,0,0,1,0,0,0,1,0,0,0,0,0,0]
Middle finger flexion + Thumb abduction	[0,0,0,0,1,0,0,0,0,0,1,0,0,0,0]
Middle finger flexion + Thumb adduction	[0,0,0,0,1,0,0,0,0,0,0,1,0,0,0]
Middle finger flexion + Wrist flextion	[0,0,0,0,1,0,0,0,0,0,0,0,1,0,0]
Middle finger flexion + Wrist extension	[0,0,0,0,1,0,0,0,0,0,0,0,0,1,0]
Middle finger flexion + Wrist pronation	[0,0,0,0,1,0,0,0,0,0,0,0,0,0,1]
Middle finger flexion + Wrist supination	[0,0,0,0,1,0,0,0,0,0,0,0,0,0,0]
Index finger flexion + Thumb down	[0,0,0,0,0,0,1,0,1,0,0,0,0,0,0]
Index finger flexion + Thumb abduction	[0,0,0,0,0,0,1,0,0,0,1,0,0,0,0]
Index finger flexion + Thumb adduction	[0,0,0,0,0,0,1,0,0,0,0,1,0,0,0]
Index finger flexion + Wrist flextion	[0,0,0,0,0,0,1,0,0,0,0,0,1,0,0]
Index finger flexion + Wrist extension	[0,0,0,0,0,0,1,0,0,0,0,0,0,1,0]
Index finger flexion + Wrist pronation	[0,0,0,0,0,0,1,0,0,0,0,0,0,0,1]
Index finger flexion + Wrist supination	[0,0,0,0,0,0,1,0,0,0,0,0,0,0,0]
Thumb down + Thumb abduction	[0,0,0,0,0,0,0,0,1,0,1,0,0,0,0]
Thumb down + Thumb adduction	[0,0,0,0,0,0,0,0,1,0,0,1,0,0,0]
Thumb down + Wrist flextion	[0,0,0,0,0,0,0,0,1,0,0,0,1,0,0]
Thumb down + Wrist extension	[0,0,0,0,0,0,0,0,1,0,0,0,0,1,0]
Thumb down + Wrist pronation	[0,0,0,0,0,0,0,0,1,0,0,0,0,0,1]
Thumb down + Wrist supination	[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0]
Wrist flextion + Wrist pronation	[0,0,0,0,0,0,0,0,0,0,0,0,1,0,1]
Wrist flextion + Wrist supination	[0,0,0,0,0,0,0,0,0,0,0,0,1,0,0]
Wrist extension + Wrist pronation	[0,0,0,0,0,0,0,0,0,0,0,0,0,1,1]
Wrist extension + Wrist supination	[0,0,0,0,0,0,0,0,0,0,0,0,0,1,0]
Extend all fingers	[0,1,0,1,0,1,0,1,0,1,0,0,0,0,0]
All fingers flexion (without thumb)	[1,0,1,0,1,0,1,0,0,0,0,0,0,0,0]
All fingers extension	[0,1,0,1,0,1,0,1,0,1,0,0,0,0,0]
Palmar grasp	[1,0,1,0,1,0,1,0,1,0,0,0,0,0,0]
Pronation with the Palmar grasp	[1,0,1,0,1,0,1,0,1,0,0,0,0,0,1]

Pointing: index-ext, all-flex	[1,0,1,0,1,0,0,1,1,0,0,0,0,0,0,0]
3-digit pinch	[0,0,0,0,1,0,1,0,1,0,0,0,0,0,0,0]
3-digit pinch with pronation	[0,0,0,0,1,0,1,0,1,0,0,0,0,0,1,0]
Key grasp with pronation	[0,0,0,0,0,0,1,0,1,0,0,0,0,0,1,0]

Appendix B

Training Plots and Precision-Recall Curves

Here follows the training progress plots and resulting precision-recall curves of all test subjects, some of which were omitted from chapter 4 to conserve space.

Single Time Instant Network

Cross-Entropy Loss

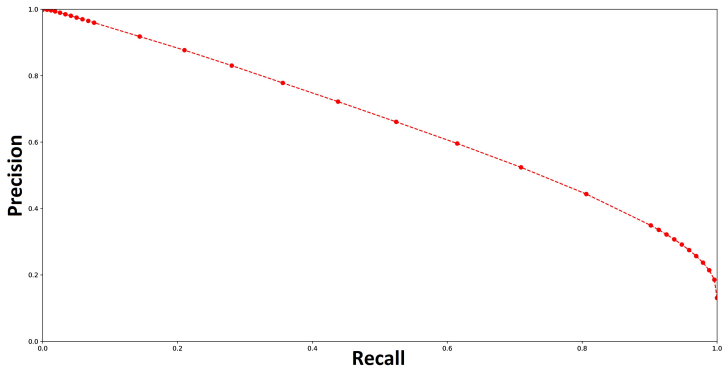


Figure B.1: Precision-recall curve acquired from the multi-label single time instant CNN trained with cross-entropy loss on data for subject 1.

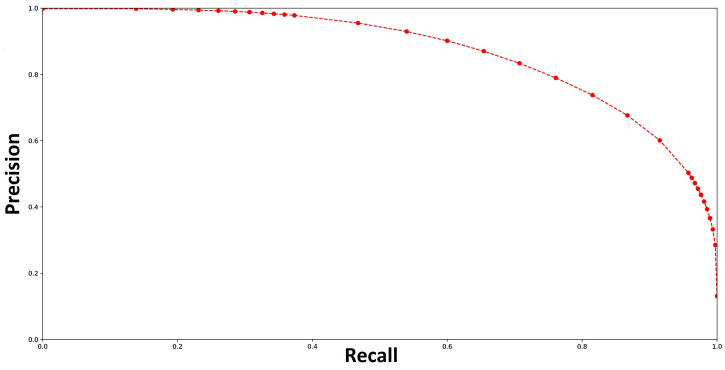


Figure B.2: Precision-recall curve acquired from the multi-label single time instant CNN trained with cross-entropy loss on data for subject 2.

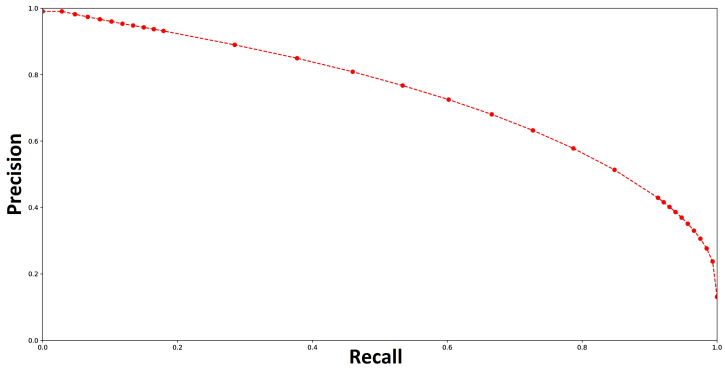


Figure B.3: Precision-recall curve acquired from the multi-label single time instant CNN trained with cross-entropy loss on data for subject 3.

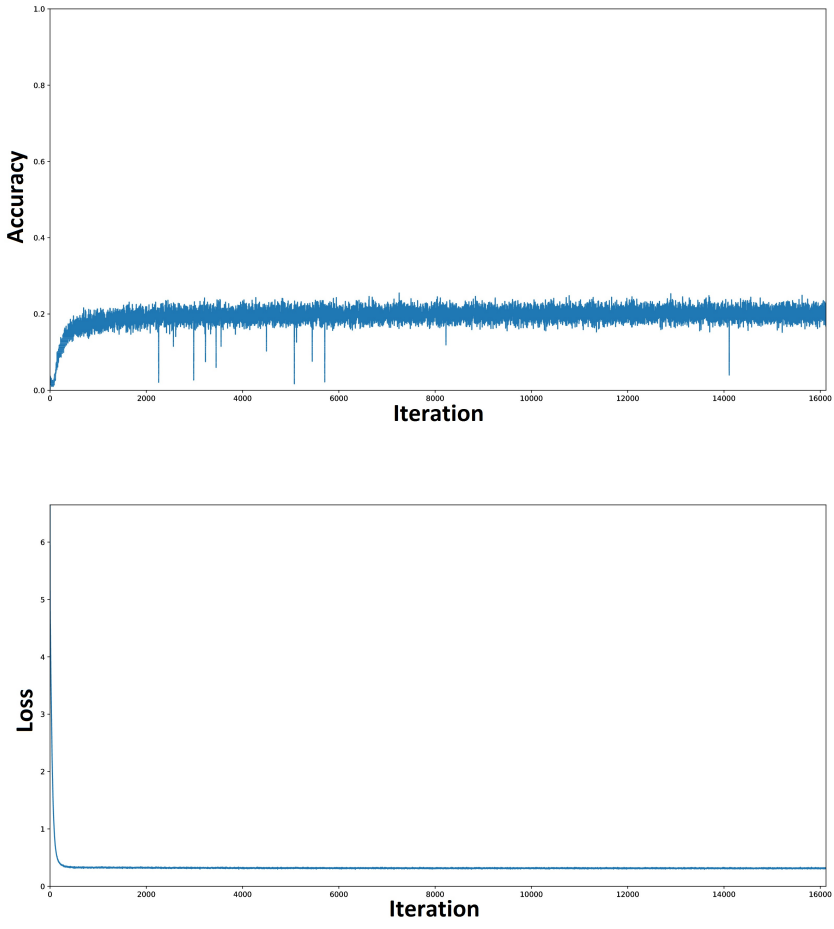


Figure B.4: (top) Multi-label single time instant CNN iteration accuracy when training with cross-entropy loss on data from subject 1 (bottom) Corresponding iteration loss

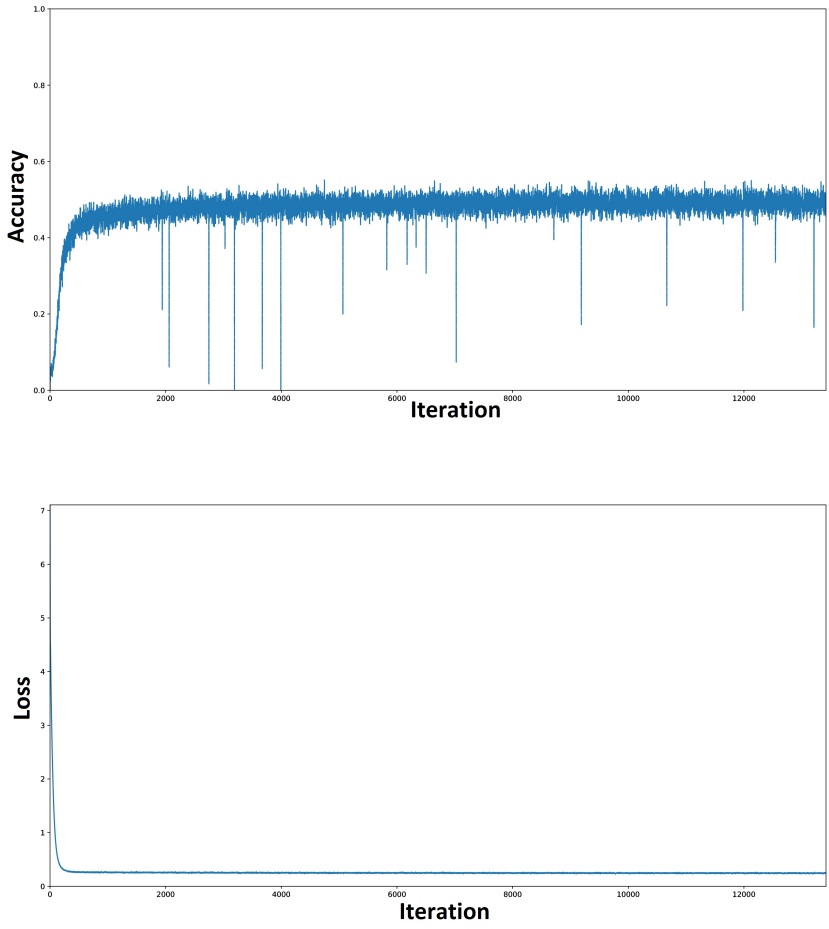


Figure B.5: (top) Multi-label single time instant CNN iteration accuracy when training with cross-entropy loss on data from subject 2 (bottom) Corresponding iteration loss

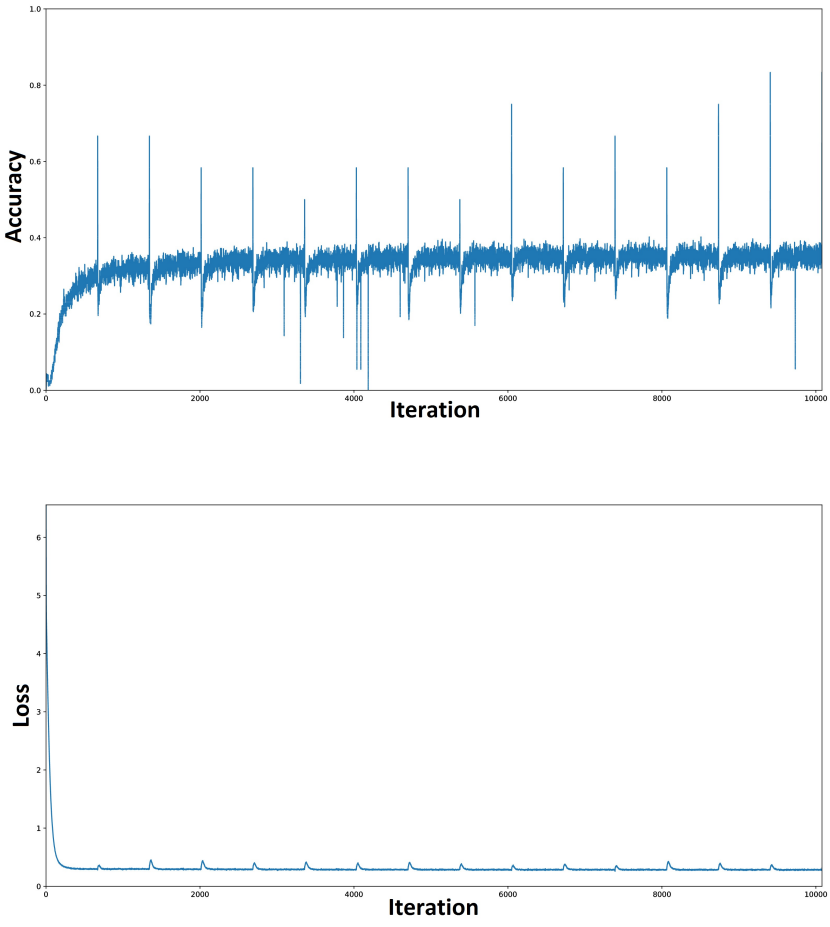


Figure B.6: (top) Multi-label single time instant CNN iteration accuracy when training with cross-entropy loss on data from subject 3 (bottom) Corresponding iteration loss

BP-MLL Loss

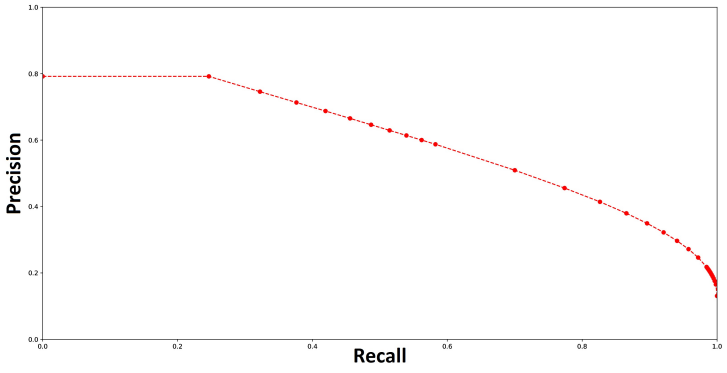


Figure B.7: Precision-recall curve acquired from the multi-label single time instant CNN trained with BP-MLL loss on data for subject 1.

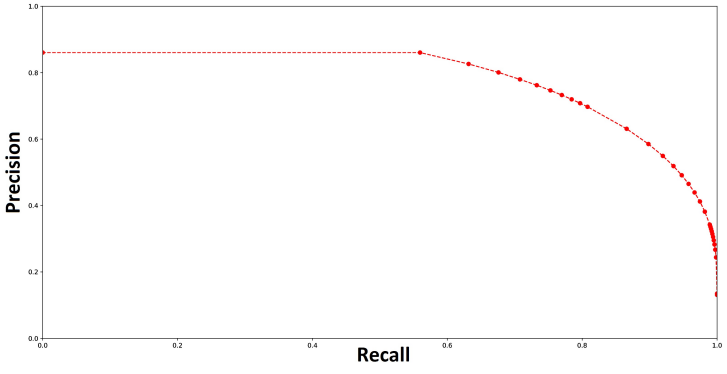


Figure B.8: Precision-recall curve acquired from the multi-label single time instant CNN trained with BP-MLL loss on data for subject 2.

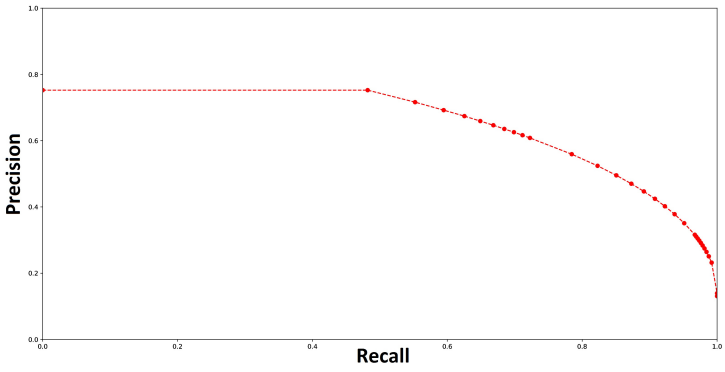


Figure B.9: Precision-recall curve acquired from the multi-label single time instant CNN trained with BP-MLL loss on data for subject 3.

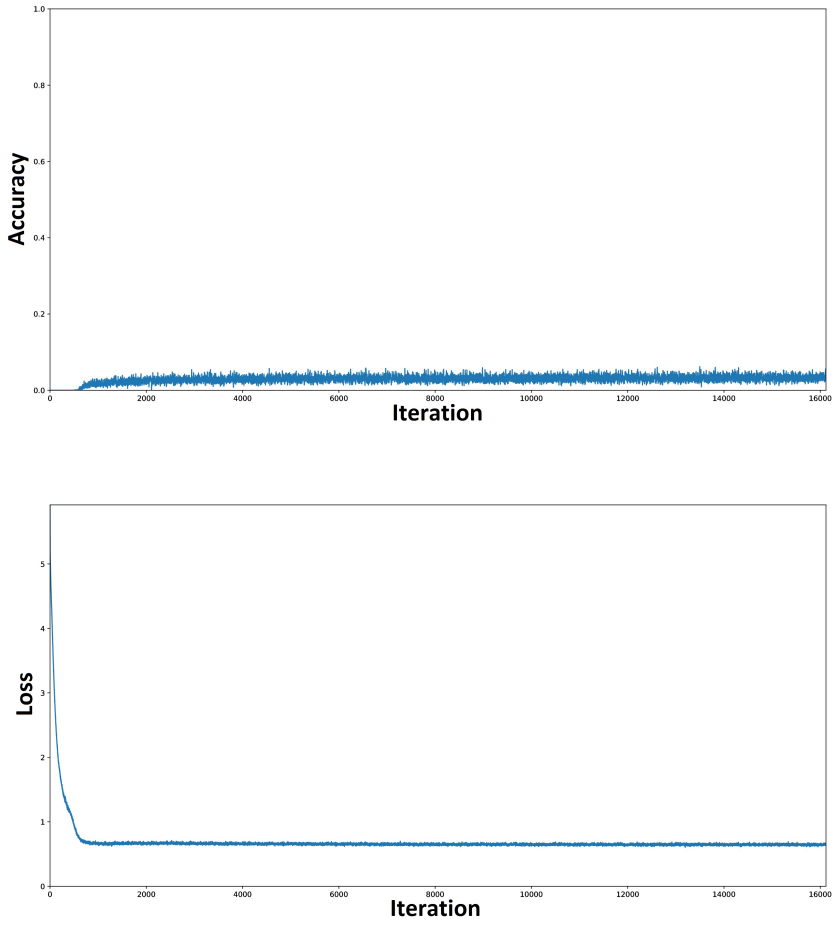
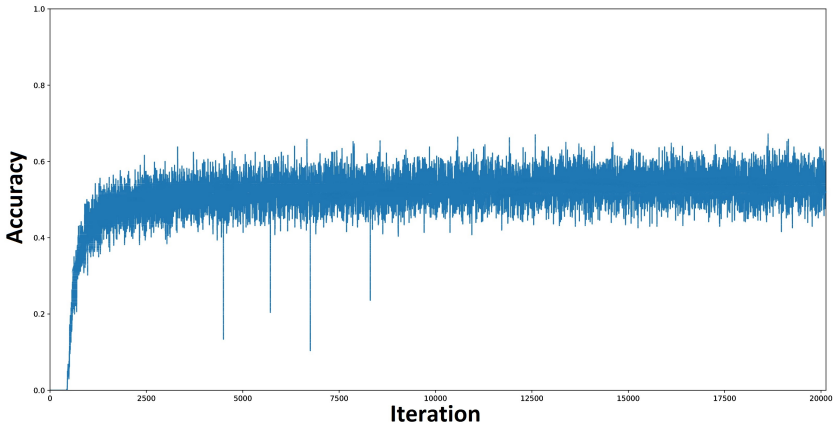


Figure B.10: (top) Multi-label single time instant CNN iteration accuracy when training with BP-MLL loss on data from subject 1 (bottom) Corresponding iteration loss.



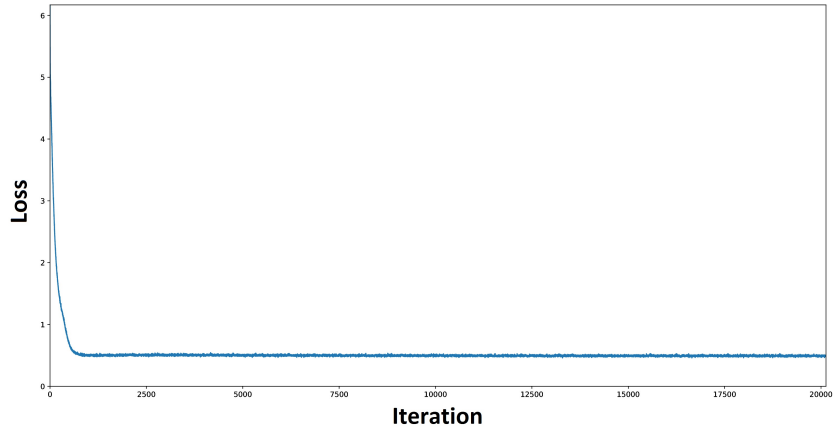


Figure B.11: (top) Multi-label single time instant CNN iteration accuracy when training with BP-MLL loss on data from subject 2. (bottom) Corresponding iteration loss

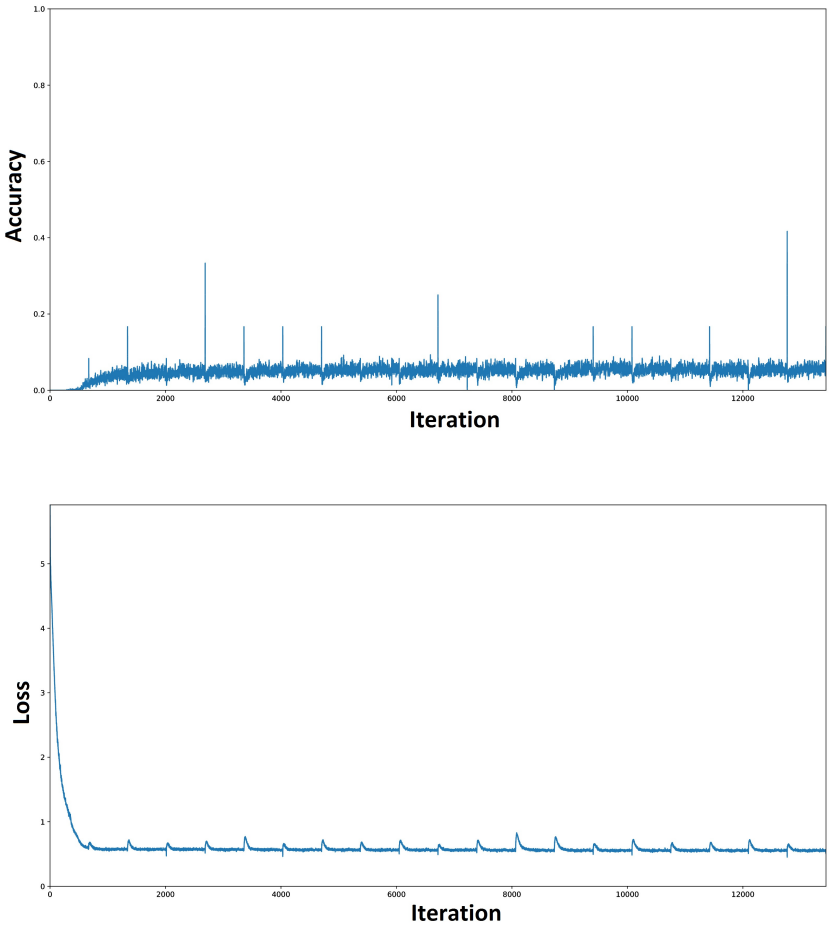


Figure B.12: (top) Multi-label single time instant CNN iteration accuracy when training with BP-MLL loss on data from subject 3 (bottom) Corresponding iteration loss.

Time Domain Features Network

Cross-Entropy Loss

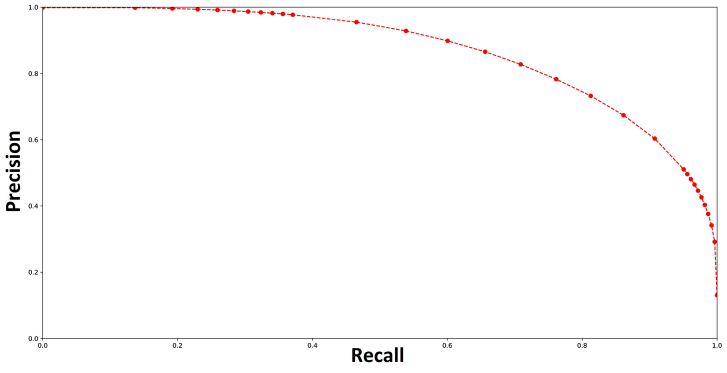


Figure B.13: Precision-recall curve acquired from the multi-label time depth CNN trained with cross-entropy loss on data for subject 1.

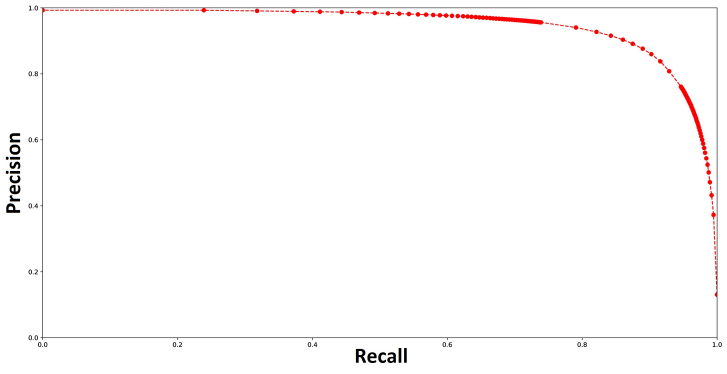


Figure B.14: Precision-recall curve acquired from the multi-label time depth CNN trained with cross-entropy loss on data for subject 2.

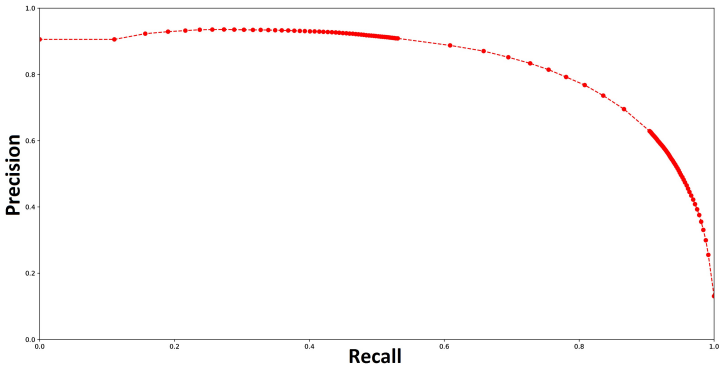


Figure B.15: Precision-recall curve acquired from the multi-label time depth CNN trained with cross-entropy loss on data for subject 3.

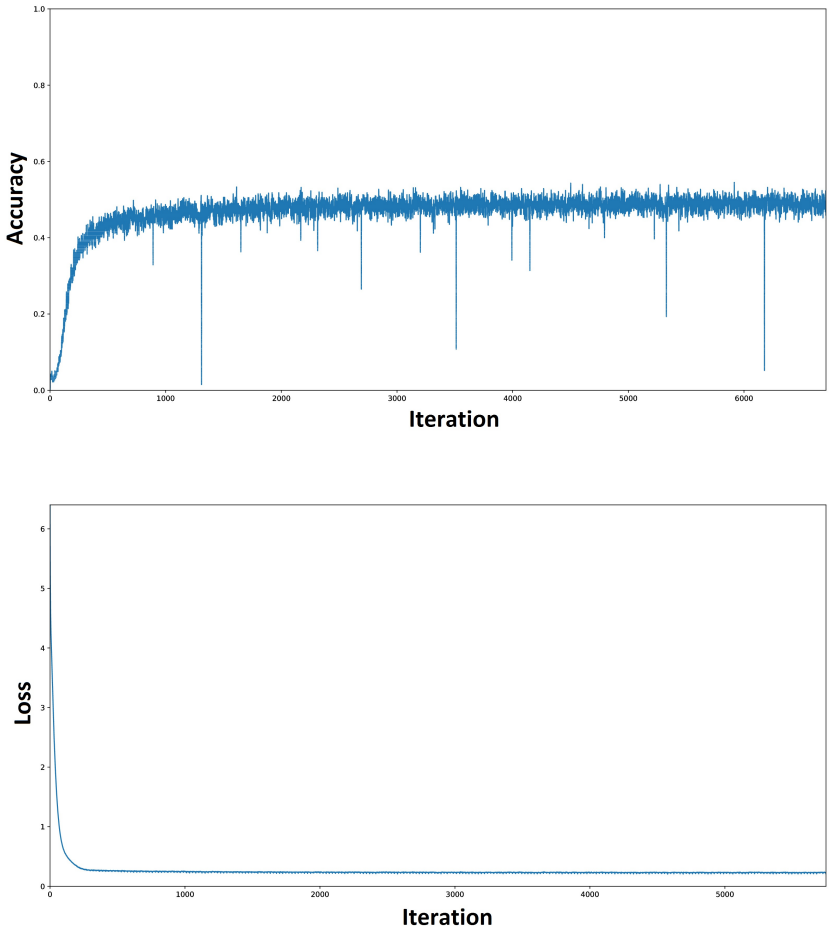


Figure B.16: (top) Multi-label time depth CNN iteration accuracy when training with cross-entropy loss on data from subject 1 (bottom) Corresponding iteration loss.

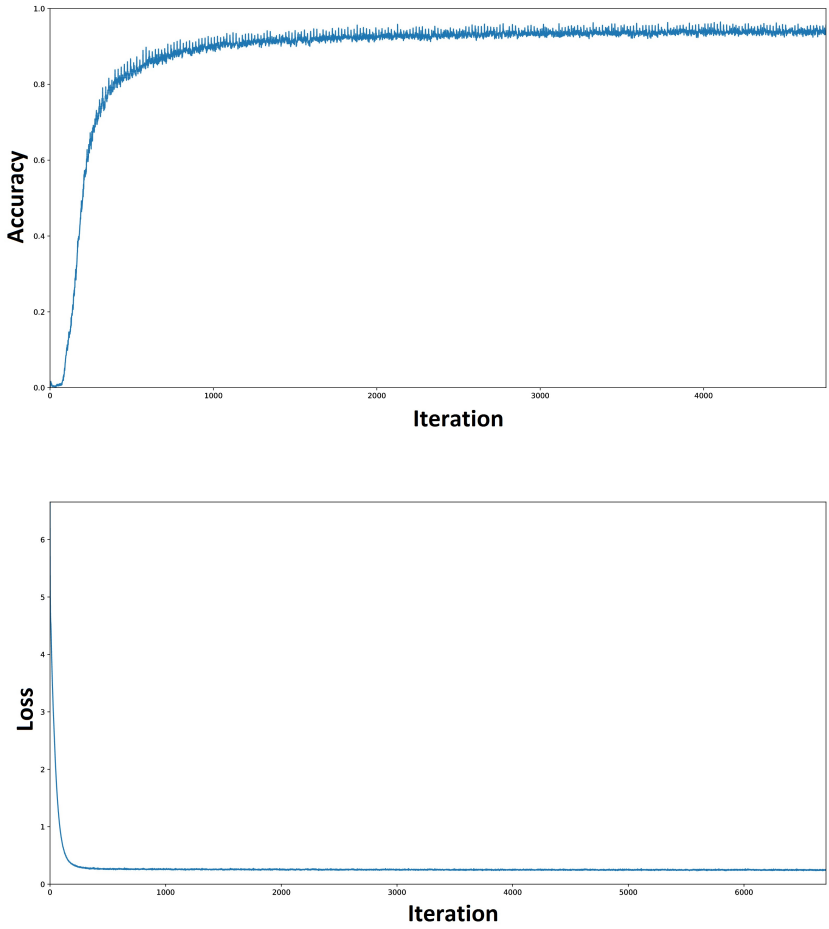


Figure B.17: (top) Multi-label time depth CNN iteration accuracy when training with cross-entropy loss on data from subject 2 (bottom) Corresponding iteration loss.

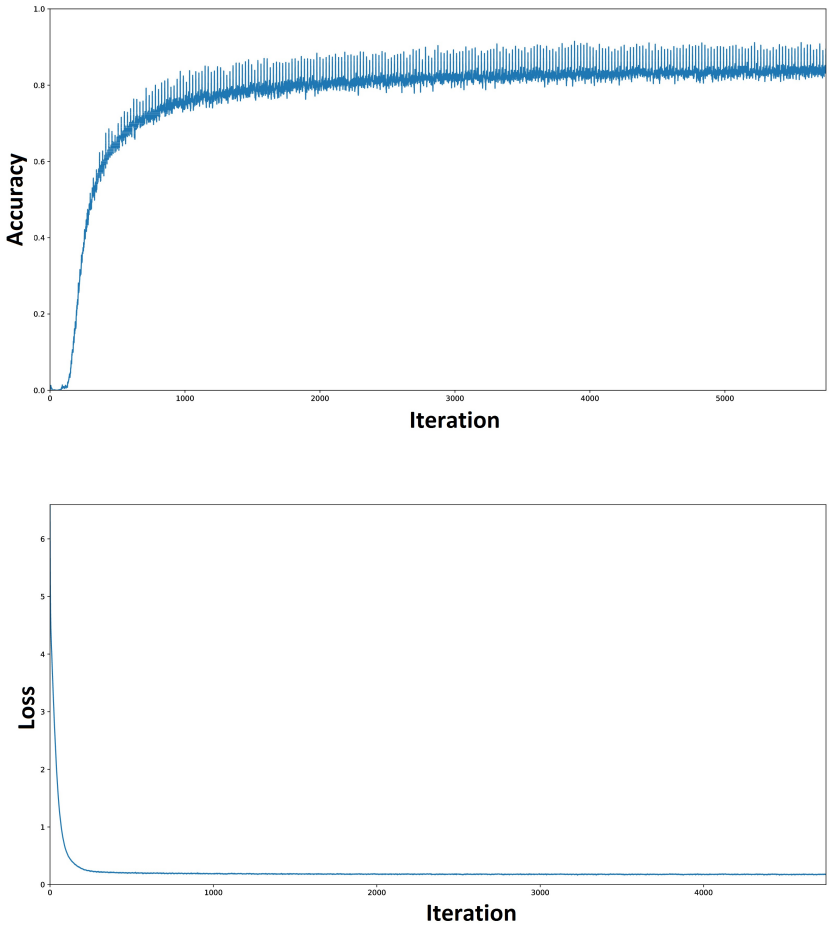


Figure B.18: (top) Multi-label time depth CNN iteration accuracy when training with cross-entropy loss on data from subject 3 (bottom) Corresponding iteration loss.

BP-MLL Loss

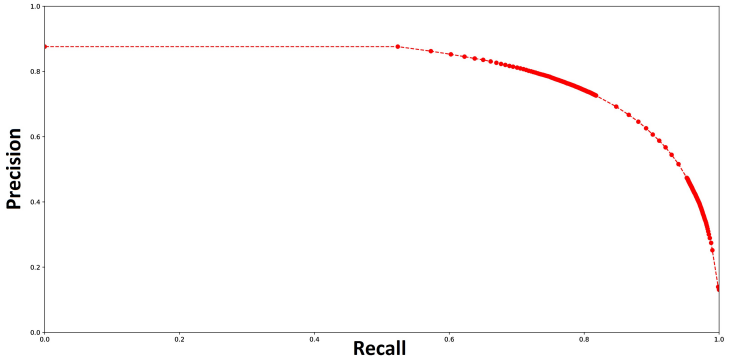


Figure B.19: Precision-recall curve acquired from the multi-label time depth CNN trained with BP-MLL loss on data for subject 1

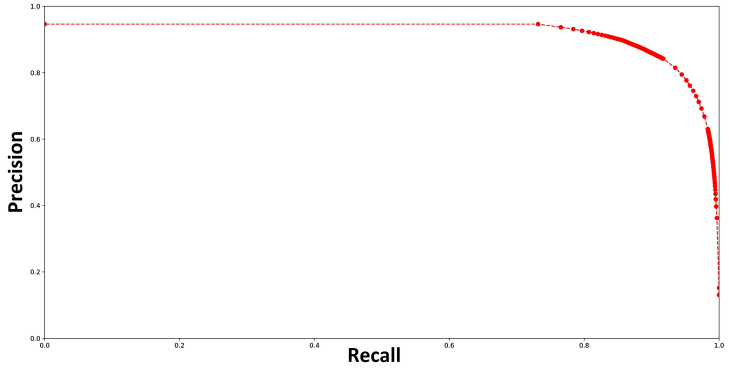


Figure B.20: Precision-recall curve acquired from the multi-label time depth CNN trained with BP-MLL loss on data for subject 2

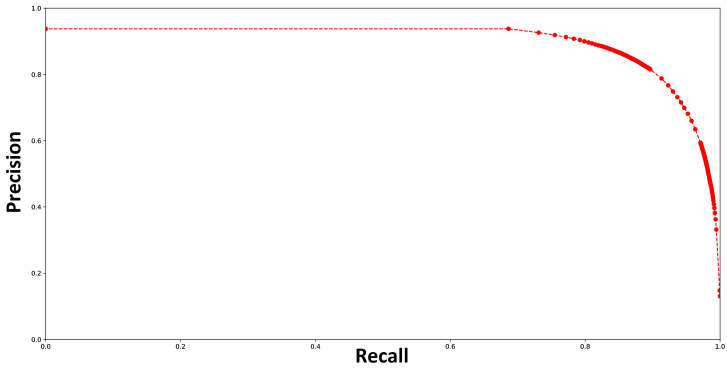
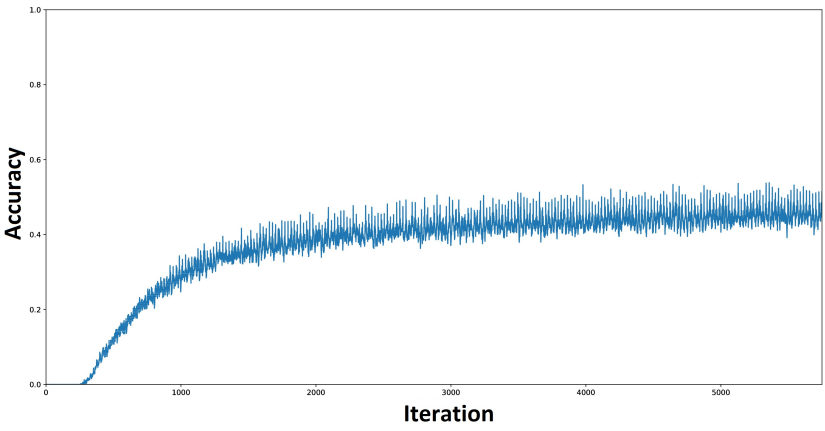


Figure B.21: Precision-recall curve acquired from the multi-label time depth CNN trained with BP-MLL loss on data for subject 3



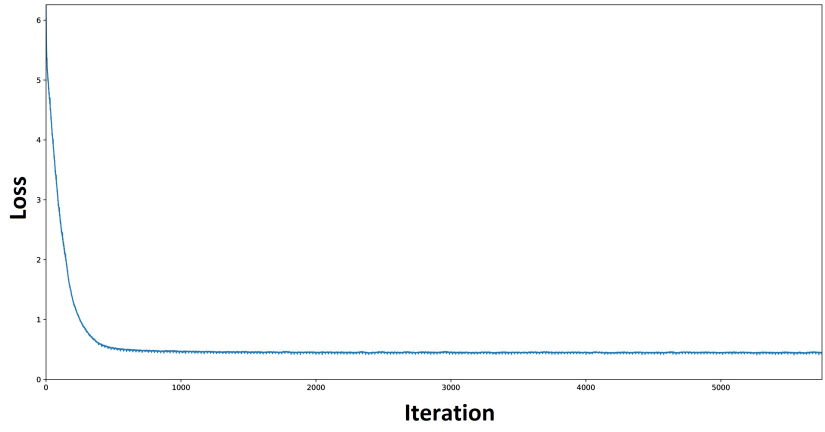
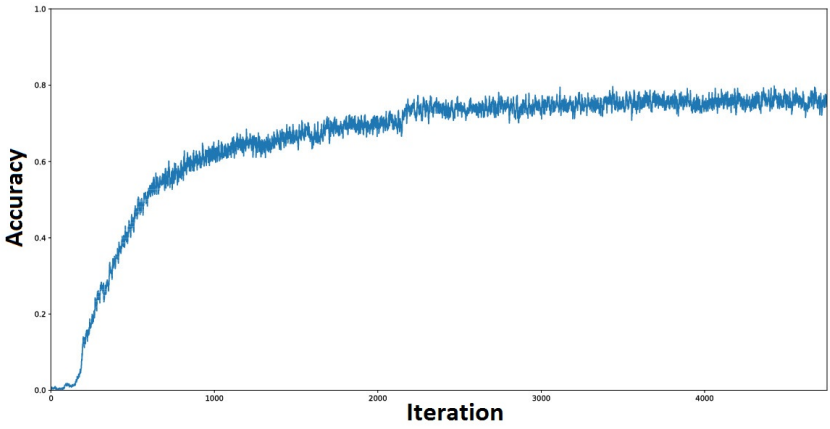


Figure B.22: (top) Multi-label time depth CNN iteration accuracy when training with BP-MLL loss on data from subject 1 (bottom) Corresponding iteration loss



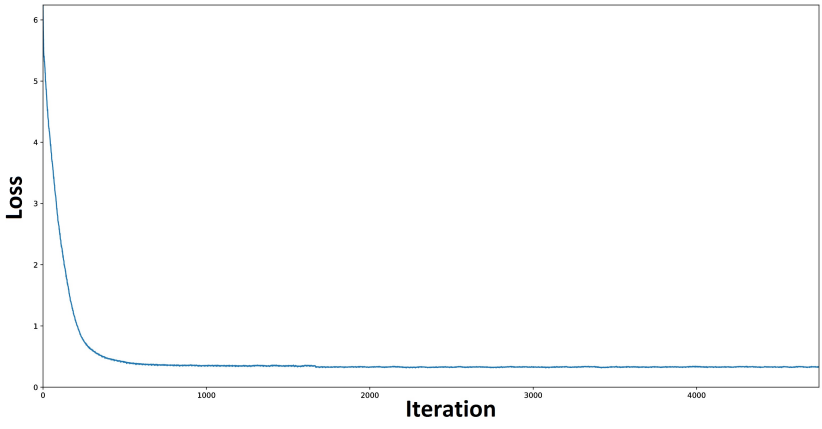
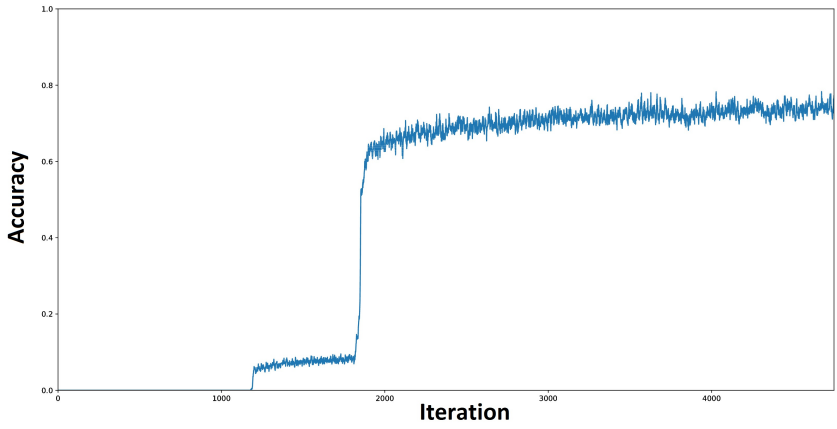


Figure B.23: (top) Multi-label time depth CNN iteration accuracy when training with BP-MLL loss on data from subject 2 (bottom) Corresponding iteration loss



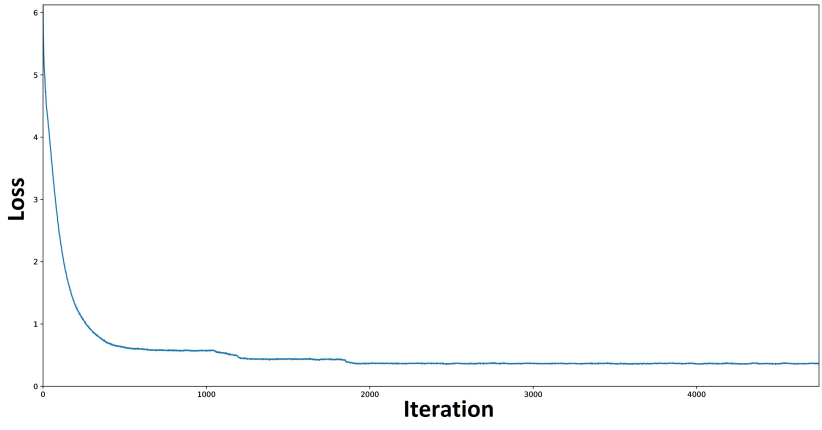


Figure B.24: (top) Multi-label time depth CNN iteration accuracy when training with BP-MLL loss on data from subject 3 (bottom) Corresponding iteration loss