CHALMERS
UNIVERSITY OF TECHNOLOGY

Real time gaze based status recognition

Using Machine Learning to predict user activities and state based on gaze data

Master's thesis in Engineering Mathematics and Systems, Control and Mechatronics

SEBASTIAN AGERHÄLL, FREDRIK JOHANSSON TORNÉUS

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019

Using Machine Learning to predict user activities and state based on gaze data

Is it possible to tell the status of a user based on their gaze patterns?

SEBASTIAN AGERHÄLL, FREDRIK JOHANSSON TORNÉUS

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2019

Real time gaze based status recognition
Using Machine Learning to predict user activities and state based on gaze data

SEBASTIAN AGERHÄLL, FREDRIK JOHANSSON TORNÉUS

Supervisor: Petter Falkman, Department of Electrical Engineering
Examiner: Petter Falkman, Department of Electrical Engineering

Master's Thesis 2019
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A user reading a text on which the user's gaze pattern is drawn.

Typeset in LaTeX
Gothenburg, Sweden 2019

SEBASTIAN AGERHÄLL, FREDRIK JOHANSSON TORNÉUS
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Real time gaze based activity recognition is a way to become more aware of what the user is doing. By using this information it is possible to create user interfaces which can adapt for the user and therefore create a better experience. The purpose of this study is to explore how gaze data and machine learning can be used to predict user status, i.e. activities and states, in real time. The activities chosen for this study were reading, searching in an image and scanning and the state was high or low cognitive load. This was achieved by collecting gaze data with a Tobii 4C eye tracker from 21 participants while they were performing different activities with and without an induced disturbance. The data was cleaned and used for training a Hidden Markov Model (HMM), Long Short Term Memory recurrent neural network (LSTM) and Convolutional LSTM Deep Neural Network (CLDNN). After training, the models were used in a predictor prototype application which predicted user status in real time. For activities the CLDNN performed best with f1-score of 0.62. To smooth the predictions a majority voting scheme was applied. The CLDNN model achieved an f1-score of 0.89 with 20 majority votes on the activity classification. For cognitive load the LSTM model performed best, with f1-score of 0.74 without votes and 0.91 with 20 majority votes. To conclude, it seems possible to predict user activities and states with gaze data. The majority voting increases the prediction performance in terms of f1-score, but since more data is required to make the final predictions, the prediction speed will be slower.

Keywords: real time, gaze, activity recognition, eye tracking, cognitive load, LSTM, CLDNN, HMM.

Acknowledgements

We would like to thank our supervisors at Tobii, Dzenan Dzemidzic and Deepak Akkil, for supporting us by answering questions and helping us find the right people to talk to. Our supervisors at Chalmers, Petter Falkman and Julius Pettersson, for the interest in our work and for the discussions regarding the report. The participants in our study whom have contributed with all data used for training our models by completing the sometimes agonizing test.

Sebastian Agerhäll and Fredrik Johansson Tornéus, Gothenburg, August 2019

Contents

List of Figures

List of Tables

1

Introduction

This chapter will present some background for why this project is relevant and what is possible with the use of eye tracking. Then some previous works which has focused on classifying eye movements with different methods. After this the objective, scope and purpose and aims are presented.

1.1 Background

For any consumer product the user experience is an important part. When interacting with a product we want the interaction to be as smooth and intuitive as possible. Eye tracking can give more understanding about a user's behaviour by analyzing eye movements. This information can be used in order to adapt the user interface to create more intuitive and natural user experiences for each specific user status. An interface that is aware of its user could make for a better user experience. E.g. when the user is focused on its task, do not show distracting notifications. When the user is not focused or seem tired, suggest to take a break. To create one single interface which suits all user statuses might not be possible to achieve.

Depending on the activity of the user the eye behaves differently. For example, while reading a text the eye has a sweeping pattern that follows the direction of the text [7]. This compared to when playing a first person shooter game where the eye has concentrated fixations in the center of the screen [8] makes it possible to distinguish between activities. The usage of eye trackers to analyze and enhance user interaction has previously been done with expensive hardware. Now it is possible to buy affordable eye trackers which can be found even in some consumer computers [9]. That makes this previously niche research topic more available for the masses and eye-tracking is gaining interest as a mainstream human-machine interaction method [10].

1.2 Previous work

Several methods to analyze eye movements have been proposed earlier. [11] investigated eye movement analysis by designing 90 different features based on fixations, saccades and blinks, see Section 2.1 for an explanation of the terminology. The activities analyzed in [11] were: Copying a text, reading a printed paper, taking hand-written notes, watching a video and browsing the web as well as a NULL activity when doing nothing. To find the most important features for the different activities they used minimum redundancy maximum relevance feature selection (mRMR). The data was collected with 8 people which correspond to 8 hours of recorded data. They trained a Support Vector Machine on the data to be able to make predictions. They obtained an average precision of 76.1% and recall of 70.5% over all classes and participants. 62 of the 90 features were based on saccades. [12] found that fixation duration and saccade amplitude are strong indicators of certain activities. They used wordbooks of eye patterns in combination with the mRMR. The wordbooks consist of fixed length sequences of eye movements.

[13] has a similar approach with features. They define high level, medium level and low level features. High level features work on a macro scale considering activity in certain Areas of Interest (AOI), e.g. Time spent in an AOI or transitions between them. Low level features consider eye movements on a micro scale, e.g. duration of individual fixations or direction of saccades. The downside of high level features is that it requires a priori knowledge of the AOI:s. The downside of the low level features is that there is an overlap in eye movements including each low level feature. Analyzing individual low level features may therefore lead to overfitting issues. Therefore [13] introduces medium level features to cope with the trade off between high and low level features. A medium level feature could be the repeated left - right saccades corresponding to reading a line in a western language, i.e. a certain combination of low level features.

The authors of [13] tests their approach of medium level features with a self composed data set with sample size N=24 and predefined eye activities such as `Read`, `Browse` and `Search`. They evaluate three different kinds of classification models, K-Nearest Neighbour (K-NN), Support Vector Machines (SVM) and Random Forest. The Random forest classifier performs best in their case with an overall f1-score of 0.72. They mention several limitations of their work such as, low sample size and restrictive composition of features, e.g. they only considered four possible directions of the saccades.

It is also possible to use methods that do not have predefined features. [14] address the problem of task recognition using gaze interactions as well as mouse- and keyboard interactions, using a Layered Hidden Markov Model (LHMM). The authors used a so called Hierarchical task model to decompose each task in sub tasks or sub states which could more easily be analyzed. Each sub task represents a state in the LHMM. To quantify the gaze activities they counted the occurrence of the gaze in different areas of interest. [14] also touches on the important subject of data collec-

tion. They propose two ways of collecting data. In one case data was collected by letting *Expert Users* solve a task while their eye movements were monitored. In the other case the test subjects had predefined instructions of how to solve a task. These approaches are used to get an as certain label as possible for the actual activity of the user. They try their approach on three different tasks, one is about answering a quiz and the other two are two versions of analyzing a google analytics curve. For the quiz task the LHMM successfully predicted the user activity with 89% accuracy and for the Google analytics it had around 50% accuracy.

[15] uses an improved version of the ordinary Recurrent Neural Network (RNN) called Long Short Term Memory (LSTM) to predict if a user needs navigation help or not in a Virtual Reality (VR) setting. In this case the user's task is to find a fruit by navigating through a maze. The aim was to provide navigation help only when the user was uncertain of which way to go. By letting 22 users perform the task of finding fruits while indicating with a button when they were unsure of where to go, training data was collected. Using this data they trained an LSTM and managed to get above 99% accuracy of deciding whether a user needed navigation aid or not. This method works without predefined features.

The RNN approach has further been investigated by [16], where they aim at classifying if a person is confused or not. Their models are trained on data from a ValueChart user study. ValueChart is an interactive visualization tool for decision tasks. The ValueChart is designed for usability, but with complicated decision tasks confusion might still arise. They compared an LSTM and a Gated Recurrent Unit (GRU), which are extensions to an ordinary RNN, with a Random Forest classifier. The RNN:s were trained on the raw data and the random forest used high level manually crafted features. The LSTM performed the best of the RNN:s with an sensitivity of 0.74 and a specificity of 0.71, which can be compared with the random forest which got an sensitivity of 0.57 and a specificity of 0.91, i.e the LSTM gives fewer false negatives while the RNN gives fewer false positives.

Except for in the case of VR navigation, the previous work has focused on offline classification. In offline classification there is time for extensive preprocessing of the data and the models prediction speed does not have to be taken into account. To be able to make real time adaptive software the predictions have to be made online, which limits the possibilities of preprocessing and puts high demands on the prediction speed of the models. For the real time user interface adaptation the prediction performance should correspond to an f1-score above 0.9 according to the user experience team at Tobii. This work will focus on finding models that can handle the speed accuracy trade-off well.

1.3 Objective

The objective of this thesis is to first define different user activities and states and then create an application that learns to recognize them in real time.

1.4 Scope

There is an enormous amount of activities and states that could be interesting to track. To make this study feasible it will be limited to only focus on the five activities: reading, scanning, searching text, searching images and skimming as well as one internal state, cognitive load, see definition of each activity and state in Section 3.2. The activities and states were chosen to fit interesting use cases at Tobii. Factors that were considered was how valuable for the Tobii tracking of the activity was and how difficult it would be to collect good data related to the activity.

Previous research have considered both using predefined higher level features as well as raw gaze data to predict on. When dealing with real time applications it is beneficial to make predictions on the raw data stream to reduce the time required for preprocessing the data. In particular neural networks are good at learning features from the data [16], which may lead to more optimal features than what could be manually thought of. For these reasons this study will focus on predicting on the raw gaze data as well as some low level features that are fast to compute, see Section 3.4.

The aim is to find models that can recognize activities based only on eye movement data. No information about the context on the screen, i.e. if the user is looking at an area with text, will be available to the models.

To achieve high quality data, the data collection for each participant needs to be done thoroughly and hence it will also be time consuming. Due to time being a limiting factor in this project, the number of test participants will be limited to 21 people.

1.5 Purpose and aims

Our ambition with this study is to make a contribution to the field of human-machine interaction by creating a system that, in real time, can decide user status based on gaze data.

2

Theory

The initial part of this section will describe basic eye physiology and eye tracking terminology. The second part will explain the theory behind the machine learning models that were used.

2.1 Eye Movements

To be able to create eye tracking technology one must understand how the eye works. To be able to perceive objects the eye has to be fixed on the object for a brief moment in order to allow the brain to register it, typically *fixations* of 200ms to 600ms is required. The area with sharp vision, called the *fovea*, is rather small, roughly the size of a thumbnail on arm's length distance. Outside of the fovea the vision accuracy decreases rapidly. This means that the eye has to scan objects by rapid eye movements in order to get an accurate image of the entire object. These quick eye movements are referred to as *saccades* [10].

Another kind of eye movements is when the eyes follows a moving target. This movement is referred to as a pursuit movement. The small size of the fovea and the requirement of scanning is what makes eye tracking possible. The so called gaze vectors, i.e. a vector directed from the eyes to the point where the gaze is focused, can be estimated from the line of sight of the viewer [10].

2.2 Tracking technology

There are several techniques for tracking eye movements where the foremost are video based solutions, IR light reflection solutions and electrooculography (EOG)[10]. In this work we will use trackers which implements the IR method.

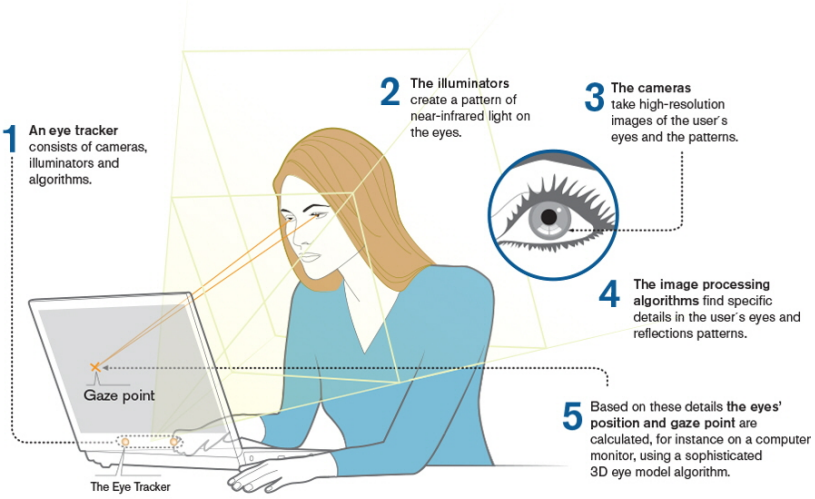


Figure 2.1: A description of how the IR-PCR eye tracking works [1]

2.2.1 Infrared Pupil Corneal Reflection (IR-PCR)

IR-PCR creates a reference point in the eye by sending on- and off axis IR light rays towards the pupil. A camera then captures the reflections of these light rays in the eyes. The on axis light rays gives a bright image of the pupil enabling easier recognition of the pupil. The off axis light ray will provide a dark image of the pupil. The light rays will be reflected on the cornea and on the pupil. The corneal reflection is more stable to eye and head movements than the pupillary reflections and can therefore be used as a reference point to maintain accuracy even when the eyes move. Since IR light is invisible to the eye, the light rays will not disturb the eye of the subject. The gaze vectors are then estimated by the relative motion of the center of the moving pupil and the corneal reflection [10].

The perk of IR-PCR is that it can give high accuracy estimates of the gaze vectors. Video based solutions are usually easier to implement and give good view of the eye movements on a macro scale. Gaze estimation requires high accuracy which IR-PCR can provide. EOG can also provide high accuracy but requires electrodes attached to the test subject and is thus more suitable for lab research than for commercial products [10].

2.3 Machine Learning models

This section will describe the mathematics behind the chosen model architectures and motivations to why they are useful.

2.3.1 Deep Feed Forward Neural Networks

Deep feed forward Neural Networks, or simply Deep Neural Networks (DNN) are methods for approximating a true model f^*, relating some features to some output [17]. The true model f^* is unknown and practically unreachable, but with enough data that accurately describes reality a DNN can be trained to find a good approximation of f^*. The DNN can be represented by the function f which is a multivariate function accepting an input vector $\mathbf{x}$ with p dimensions. p is the number of features, or predictors, in the data. The term deep refers to that the network has several layers. In every layer a set of operations is performed on the input data. The layers between the input layer and the output layer is called hidden layers. Each hidden layer consists of several nodes and these nodes decides how the information can flow through the neural network. For each node in each hidden layer the input from the previous layer is multiplied by some weights, then bias is added and finally everything is "activated" by an activation function to give a single value to the node [18]. An activation function is typically a function that somehow squashes the input to a value between 0 and 1. This value determines how much each node is activated and based on how much each node is activated predictions can be made. Figure 2.2 shows the basic architecture for a DNN with two hidden layers. The fact that every input is connected to every node is referred to as the network being fully connected. The activation function is necessary to introduce non-linearities into the network which also enables the network to learn non-linear relationships [19]. Equation (2.1) shows f for a two layer network.

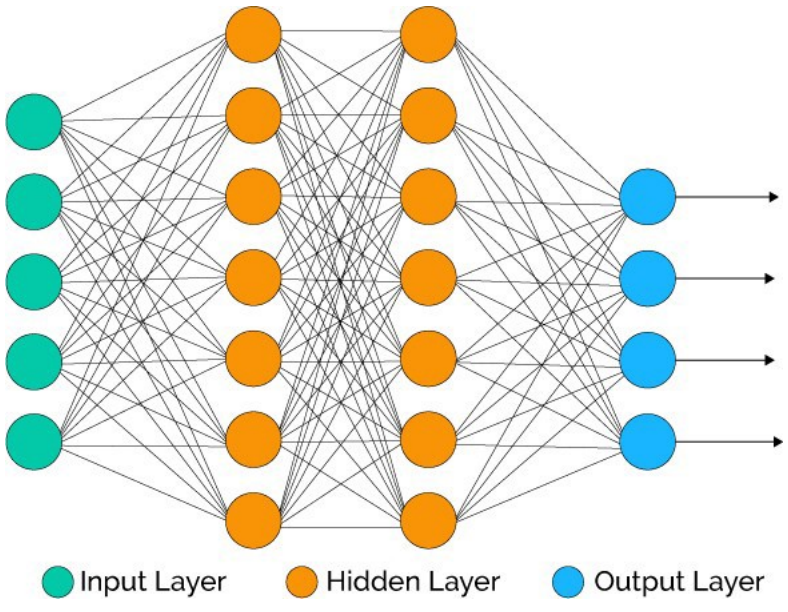


Figure 2.2: Illustration of a two layer DNN [2]

$$f = \sigma(B(\sigma(A\mathbf{x} + b)) + c), \tag{2.1}$$

where σ is some activation function, A and B are weight matrices and b and c are

bias constants. The aim of DNN:s is to find a function f in some family of functions $\mathcal{F}$ that satisfies

$$f = \min_{g \in \mathcal{F}} ||f^* - g|| + \lambda\phi(g). \tag{2.2}$$

The first term in equation (2.2) is an error term that constrains f to be close to f^* and the second term is a regularization term. λ is the regularization parameter and ϕ is the regularization function, eg. the L^2-norm of the weights. The goal of the regularization is to keep the model complexity as low as possible. An overly complex model tend to overfit and can be unnecessary heavy [20]. In practice, with a data set $S = \{\mathbf{x}_i, y_i\}_{i=1}^n$ of size n with observations and corresponding labels, equation (2.2) becomes

$$f = \min_{g \in \mathcal{F}} \mathcal{L}(g; S) = \min_{g \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} l(g(\mathbf{x}_i), y_i)) + \lambda\phi(g), \tag{2.3}$$

where l is some kind of distance function measuring the distance between the true labels y_i and the predicted labels $g(\mathbf{x}_i)$. The first term in equation (2.3) is called the empirical risk [21] .

Solving the optimization problem of (2.3) can be done by adapting layer weights with an algorithm called backpropagation [22].

2.3.1.1 Backpropagation

The idea of backpropagation is to use gradient descent to minimize $\mathcal{L}$. This means using the chain rule for partial derivatives to find the gradient of $\mathcal{L}$ with respect to each weight w_{ij} in each layer, $\frac{\partial \mathcal{L}}{\partial w_{lij}}$, where l represents the layer and i, j is the indices of the weight matrices, meaning the connection from input node i to hidden node j[18]. For a full derivation of $\frac{\partial \mathcal{L}}{\partial w_{lij}}$ see [18]. When the gradient is computed the current weights are updated by taking a step in the negative gradient direction, hence gradient descent. The learning rate ϵ decides the step size and thus the new weights W_{new} are calculated by updating the current weights $W_{current}$.

$$W_{new} = W_{current} - \epsilon \frac{\partial \mathcal{L}}{\partial w_{lij}} \tag{2.4}$$

2.3.2 Convolutional Neural Networks

For high dimensional inputs such as images and speech sequences, the ordinary fully connected DNNs become very heavy since they need huge weight matrices. However if each data point has spatial context, i.e. neighbouring pixels are correlated, a more

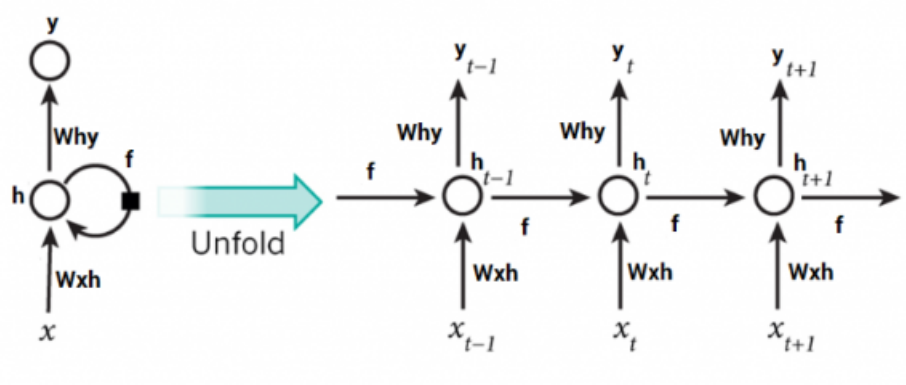


Figure 2.3: Illustration of how data flows in a Recurrent Neural Network [3].

efficient method can be used namely the Convolutional Neural Networks (CNN) [23]. CNNs are still feed forward networks but the difference from the ordinary DNNs is that the weights are convoluted with the input, hence the name. The weights in CNNs are usually referred to as filters. The filters are usually quite small compared to the input and to be able to account for all input the filters are scanned over the input to cover everything. This brings many advantages over DNNs such as weight sharing and local connectivity [24]. Weight sharing means that the weights are shared between the nodes, i.e. each input node does not have a unique connection to each hidden node which means that the number of connections is much less than in a DNN. This results in fewer weights to learn and thus CNNs scale better to large inputs than DNNs. Local connectivity means that only a small (local) fraction of the input is convoluted with the filter to create an activation in the hidden node. because of the above mentioned scanning the entire input is still covered, but the filters can be kept small which also decreases the number of trainable weights and thus also improves the scaling properties.

2.3.3 Sequential data

Sequential data or time series data is data which evolves over time. Finding patterns in time series data includes analyzing the instantaneous features as well as the changes of those features over time, i.e. the temporal dependence. Regular feed forward networks are good at connecting the instantaneous features to a pattern but they are unable to account for the temporal behavior of the data, which in many cases are an important driver of the true pattern [25].

2.3.4 Recurrent Neural Networks

Recurrent Neural Networks (RNN) is a technique that has the ability to account for the temporal, or sequential, behavior in the data and has proven useful when analyzing time series data [26]. The general idea of RNN is that each hidden state

is fed with both new external input as well as the output from the previous hidden state recursively, hence the name. By feeding in the output from the previous state the network can "remember" past events and thus it can account for temporal or sequential behavior in the data. Figure 2.3 shows the general idea of the architecture of an RNN. Mathematically this can be described as

$$h_t = \Phi(W_x x_t + W_h h_{t-1}),$$

$$y_t = \psi(W_y h_t),$$

where h_t is the hidden state at time t, Φ and ψ are activation functions, y_t is the output at time t and W_x, W_h, W_y are weight matrices.

RNNs are trained, i.e. the weight matrices are updated, by an algorithm called Back Propagation Through Time (BPTT)[27], which is an adjustment of the ordinary back propagation algorithm used in feed forward networks. BPTT enables efficient training of RNN:s, however when training RNN:s the gradient tend to either blow up or completely vanish [28]. Exploding gradients lead to very unstable weight matrices while vanishing gradients prevents learning anything new. This makes it hard for RNN:s to account for very long sequences. In 1997 Hochreiter and Schidhuber [28] suggested a new kind of architecture for solving this issue, namely the Long Short Term Memory (LSTM) model.

2.3.5 Long Short Term Memory

The first version of the LSTM network was introduced by Hochreiter and Schidhuber [28] in 1997 and since then several updates and improvements have been done, e.g. adding of the forget gate [29]. The model architecture is comprised of one or several cells where each cell contains different gates with different purposes for processing the data, see Figure 2.4. The idea is to use the gates to modify the so called cell state. The gates control what information in the cell state that should be kept, what information that should be forgotten and what new information to pick up [29]. The ability to learn what to forget and what to keep is what enables the LSTM architecture to handle long sequences better than the classical RNN architecture.

The small rectangles in the LSTM cell in Figure 2.4 represents neural network layers with an activation function. σ means that a sigmoid activation function is used and $tanh$ means that the hyperbolic tangent function is used. The sigmoid functions returns values between 0 and 1 and determine to which degree information should flow through the gate. 0 means that the gate is completely closed and 1 means that it is completely open. The first gate in the LSTM cell is the forget gate which accepts the previous hidden state h_t, i.e. output from previous cell, and the current input x_t as inputs, multiplies them with some weights W_f, adds some bias b_f, activates via sigmoid function and outputs a tensor with values between 0 and 1. Each value in

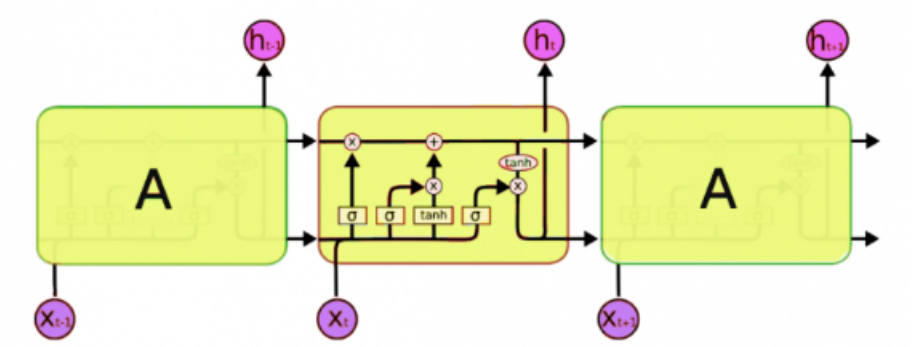


Figure 2.4: An overview of the LSTM architecture. [4]

f_t corresponds to how much of each value in the previous cell state C_{t-1} that should be kept. Mathematically this corresponds to elementwise multiplication of f_t and C_{t-1} [4].

The second gate in the LSTM cell is the input gate. The same procedure applies here, x_t and h_{t-1} are multiplied with weights, bias is added and then sigmoid activated. This time the output i_t represents which elements of the cell state that should be updated. Before this is applied to the cell state the *tanh* gate takes x_t and h_t performs the usual operations and outputs candidate updates $\tilde{C}_t$, with values between -1 and 1, to the cell state. i_t and $\tilde{C}_t$ are multiplied elementwise and added to the "forget adjusted" previous cell state to form the current cell state C_t, i.e. $C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$ [4].

The last gate is the output gate which determines what this cell will output. A sigmoid layer similar to before is applied to x_t and h_t and produces a tensor o_t which decides which parts of the cell state to output. The cell state is sqaushed by a tanh function and then multiplied elementwise with o_t to produce the current output h_t [4].

In conclusion we can write the equations for this procedure in the following way:

$$\begin{aligned} f_t &= \sigma(W_f \cdot [h_t, x_t] + b_f) \\ i_t &= \sigma(W_i \cdot [h_t, x_t] + b_i) \\ \tilde{C}_t &= tanh(W_C \cdot [h_t, x_t] + b_C) \\ C_t &= C_{t-1} \cdot f_t + i_t \cdot \tilde{C}_t \\ o_t &= \sigma(W_o \cdot [h_t, x_t] + b_o) \\ h_t &= o_t \cdot tanh(C_t) \end{aligned} \tag{2.5}$$

2.3.6 Convolutional Long Short Term Memory Deep Neural Networks

All of the above described neural network types have different advantages and disadvantages. DNN:s are good at transforming features to spaces where the features are easily separated, CNN:s are good at handling spatial information and LSTM:s are good at handling temporal or sequential information. A combination of those network structures could potentially utilize the advantages of each structure and form an even higher performing network, as proposed by [30]. The authors of [30] proposes a network architecture called Convolutional LSTM DNN (CLDNN) that combines these three network architectures. This architecture starts with some convolutional layers with the aim of extracting features which should make the sequential modelling of the subsequent LSTM layers easier. After the LSTM layers there are some fully connected ordinary feed forward networks which has the aim of increasing feature separation to make the classification easier.

2.3.7 Hidden Markov Model

A Hidden Markov Model (HMM) [31], [32] is specified with hidden states, Z, a transition probability matrix, A and an initial state probability distribution, π.

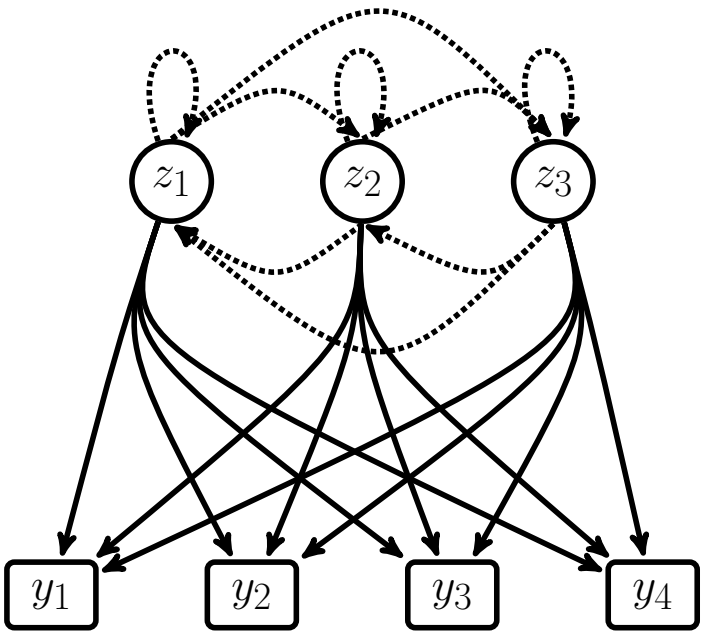


Figure 2.5: An ergodic HMM

There are different types of HMMs, one where all hidden states are reachable from all hidden states in a finite amount of steps is called ergodic. Another type is

called the left to right model, also known as Bakis model. A left to right model is characterized by a state index that either stays the same or increases at every time step, which means that the states advance from left to right [33]. This gives the ability to model signals which change over time, such as speech or eye movements. The 'Hidden' in an HMM refers to the states, which means that it is not known which state that is currently visited. In an HMM the set of all hidden states is called $Z = \{z_1, z_2, ..., z_K\}$. In each time step t the HMM will be in one hidden state. The sequence of visited hidden states is called $X = (x_1, x_2, ..., x_T)$.

The transition probability matrix, A is of size $K * K$ and contains the probabilities of transitions between all hidden states. The general form is $A = \{a_{ij}\}$

where

$$a_{ij} = P(x_{t+1} = z_j | x_t = z_i) \tag{2.6}$$

i.e. the probability of going to hidden state z_j when in hidden state z_i [34]. Note that $\sum_j a_{ij} = 1$. If the model is of type left to right it means that $a_{ij} = 0, j < i$.

The initial hidden state distribution is $\pi = \{\pi_i\}_{i=1}^K$ where

$$\pi_i = P(x_1 = z_i), \quad 1 \leq i \leq K \tag{2.7}$$

i.e. the probability that the initial hidden state x_1 is hidden state z_i [34]. Note that $\sum_{i=1}^K \pi_i = 1$ [35].

Each hidden state z_j has an emission probability distribution, b_j, for the observations generated by the hidden state. An observation sequence is $O = (o_1, o_2, ..., o_T)$. The set of emission probability distributions is called $B = \{b_j(o_t)\}$ [36]. In the case with one multivariate Gaussian for each state we have

$$b_j(o_t) = \mathcal{N}(o_t | \mu_j, \Sigma_j) \tag{2.8}$$

where o_t is an observation at time t, μ_j is the mean vector, and Σ_j is the covariance matrix at hidden state z_j [37].

HMMs has three basic problems [37] [38]:

1. Given the observation sequence O and a model $\lambda = (A, B, \pi)$, find $P(O|\lambda)$.
2. Given the observation sequence O and a model $\lambda = (A, B, \pi)$, find the hidden state sequence X which best explains the observation sequence O.
3. Find $\lambda^* = \arg\max_\lambda P(O|\lambda)$, i.e. find the best parameters for the given observation sequence O.

The goal of training the HMM is to solve problem 3, i.e. find the Maximum-Likelihood Estimate (MLE) of the parameters based on a sequence of observed data, O.

The parameters are the initial state probabilities π, the state transition probabilities A, and the parameters related to the emission distribution, B. In the case of a multivariate Gaussian distribution the parameters of the emission distribution are a mean vector, μ, and covariance matrix, Σ. The set of these parameters for a model can be written as $\lambda = (\pi, A, \mu, \Sigma)$ [37]. In order to find the MLE of λ, the Baum-Welch algorithm is most commonly used [39]. The Baum-Welch algorithm is basically the Expectation Maximization algorithm (EM-algorithm) applied to an HMM. The EM-algorithm is used to find the MLE when some of the data is missing. In the case of an HMM we have observations, O, but are missing data about which hidden state, z, that is currently visited.

The EM-algorithm is divided into two steps. The first step is the E-step which gives an expectation function of the log-likelihood by using the forward-backward algorithm for each one of the current λ parameters [40]:

$$Q(\lambda, \lambda_t) = \sum_{i=1}^{T} \log(P(O, x_i|\lambda)P(O, x_i|\lambda_t)) \tag{2.9}$$

where λ_t are the initial parameter estimates, Q is the space of all hidden state sequences of length T, $O = (o_1, o_2, ..., o_T)$ is an observation sequence and $x \in X$ where $X = (x_1, x_2, ..., x_T)$ is a hidden state sequence [41]. The second step is the the M-step which gives the parameters λ_{t+1} used for maximizing the function found in the E-step[42].

$$\lambda_{t+1} = \arg\max_{\lambda} Q(\lambda, \lambda_t) \tag{2.10}$$

The EM-algorithm begins by randomizing the parameters of λ. These parameters are used in the E-step to create an expectation function, Q, for those parameters. Q is then used in the M-step which creates a new set of λ parameters used to maximize the expectation function. The new set of λ parameters are fed into the E-step, and this loop continues until convergence.

After training, the model parameters λ are set. With the parameters λ and observation sequence O, the Viterbi Algorithm (VA) [43] is used to find the Viterbi path probability, i.e. the probability that the best hidden state sequence X, will match O, as well as the Viterbi path for this X, i.e. which states that are in the sequence X. To find the best X, we want to compute this:

$$\delta_t(i) = \max_{x_1, x_2, ..., x_{t-1}} P(x_1, x_2, ..., x_{t-1}, o_1, o_2, ..., o_t, x_t = z_i|\lambda) \tag{2.11}$$

i.e. the hidden state sequence, $X = (x_1, x_2, ..., x_t)$, ending in hidden state z_i with the highest probability of matching an observable sequence, $O = (o_1, o_2, ..., o_t)$, with the model λ. The algorithm can be divided into a few different steps.

Initialization at time step 1:

$$\delta_1(i) = \pi_i b_i(o_1), \quad 1 \leq i \leq N \tag{2.12}$$

$$\psi_1(j) = 0 \tag{2.13}$$

where δ_1 is the Viterbi path probability until time step 1, and ψ is the back pointer which is later used to find the actual path [44].

Recursion:
For each time step $t = 2, 3, ..., T$ Calculate the Viterbi path probability δ:

$$\delta_t(j) = \max_{1 \leq i \leq N}(\delta_{t-1}(i)a_{ij})b_j(o_t) \tag{2.14}$$

where N is the number of hidden states z. $\delta_{t-1}(i)$ is the Viterbi path probability from the previous step. a_{ij} is the transition probability from previous hidden state z_i to current hidden state z_j. $b_j(o_t)$ is the probability of observing o_t at hidden state z_j. Then calculate the backpointer:

$$\psi_t(j) = \arg\max_{1 \leq i \leq N}(\delta_{t-1}(i)a_{ij}), \tag{2.15}$$

which finds the hidden state index for the state z_i which has the highest probability of transition to state z_j at time t [45].

Termination:
When the recursion is done, calculate the best Viterbi path probability to time step T, called P^*, and the start of the back trace, q_T^* [44]:

$$P^* = \max_{1 \leq i \leq N}(\delta_T(i)) \tag{2.16}$$

$$q_T^* = \arg\max_{1 \leq i \leq N}(\delta_T(i)) \tag{2.17}$$

Back tracing:
The sequence of hidden states with the highest probability of matching the observation sequence is calculated by [46]:

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \quad t = T-1, T-2, ..., 1 \tag{2.18}$$

By looking at P^* acquired from 2.16 we get a measurement of how well the model can match the specified observation sequence. By training one model for each class we can compare which model matches the observation sequence the best and from that tell which class that has been most likely observed.

2.4 Evaluation of machine learning models

The primary measurement of prediction performance in this project is the F1- score. The F1-score is the harmonic mean of the recall and the precision. The recall is the fraction of the actual instances of a class that is correctly predicted as that class. Precision is the fraction of all predictions of a certain class that actually belong to this class . Mathematically the F1-score is calculated as

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}. \tag{2.19}$$

Since the F1 score considers both the recall and the precision it gives a good general estimate of the prediction performance in one single metric [47].

3

Methods

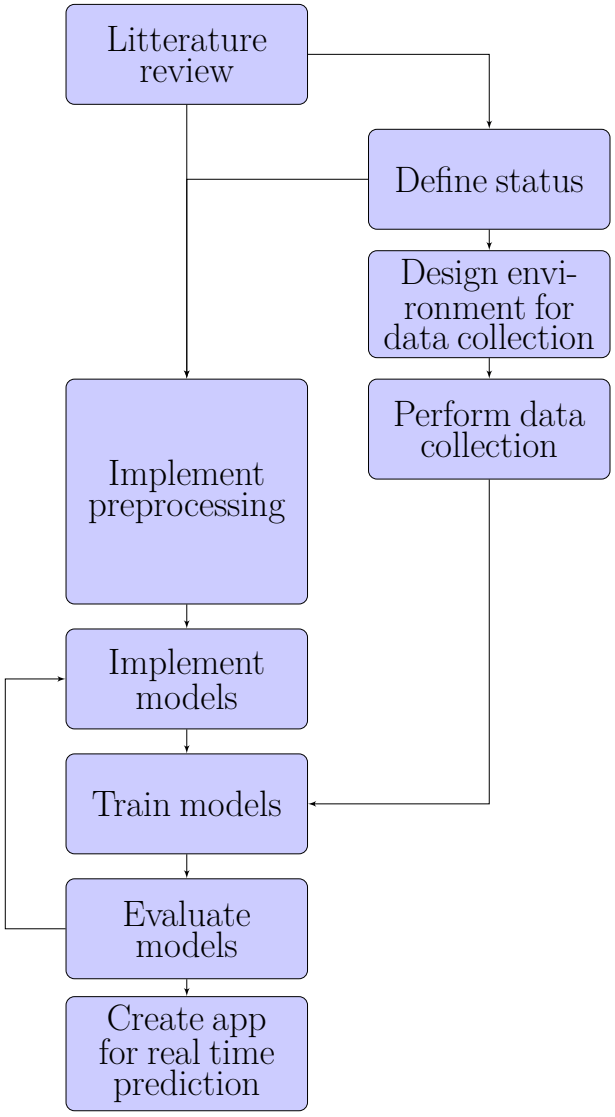
In this chapter a description of the data collection process will be provided, including choice of activities, the design of the tests used for data collection and the actual execution of the data collection. Two data collections were made, so in the first part of the data collection section the first data collection will be described. Then the purpose and execution of the second data collection will be described. After that an overview of the data used will be given as well as a description how the data was cleaned and how the features were transformed before they were fed to the model.After this the model training and model evaluation procedures will be presented. Finally the function of the prediction prototype will be described.

3.1 Method overview

In this section the general work plan will be described. The line of work is visualized in Figure 3.1.

The project was initialized with a literature review to collect more knowledge about relevant machine learning models for time series modelling as well as to find out more about what had been done previously within gaze based activity recognition. When this was done the user statuses that should be considered should be decided. With the user statuses determined the design of the data collection environment could begin. This means figuring out ways to collect data for each specific status. When this environment was in place the data collection could begin. In parallel with the data collection the data preprocessing tools and the software for model training were developed. When the data collection was done and the software tools to process data and train the models were done the model training could begin. Then the models were evaluated and modified iteratively to achieve as good prediction score as possible. Finally the prediction prototype software was developed. Each step will be described in more detail below.

Figure 3.1: Line of work as a flow chart



3.2 Activities and States

Along with representatives from Tobii some use cases of status recognition were defined. User status here refers to either a user state, i.e. cognitively loaded, or an activity, e.g. reading. These use cases were then ranked according to how valuable to Tobii they would be. Activities and states required for each use case was derived. This resulted in a large number of statuses. Considering the time scope of this project the number of statuses had to be limited. With respect to if the status was linked to top ranked use cases as well as how easily it could be measured, five activities and one state was selected.

The activities chosen are

- Reading – Ordinary reading.
- Scanning – Casually and rapidly scanning content in a structured way.
- Searching images – Search for specific objects in images.
- Searching text – Search for a specific piece of information in a text, e.g the answer to a question.
- Skimming – Fast and light reading with the aim of getting a quick overview of the text.

The state chosen is cognitive load, i.e if the user only has the main task in mind or if it has a lot of different things to think about.

3.3 Data collection process

This section will describe the different activities and states as well as how the tests were set up in Pro Lab.

3.3.1 Tests

In order to collect data a collection environment had to be designed and implemented. The software Tobii Pro Lab v. 1.114, as seen in Figure 3.2, was used for test implementation to enable easy data collection. The eye tracker used was a 90 Hz Tobii Tech 4C tracker with a special license allowing it to feed data to Tobii Pro Lab. Pro Lab allows you to create a slide show, here referred to as a test, with different visual stimuli, e.g. images or text, which is shown to the user while its eye movements are recorded. One test was created for each activity, and each test was divided into one part without extra cognitive load, and one part with extra cognitive load. The stimuli was partly images or texts directly related to the activity of the test, and partly text stimuli with instructions on how the user should proceed in the test. The aim was that the written instructions in the Pro Lab slide show should be enough for the participants to understand the task they were asked to perform. This was desirable to try and give all participants the same instructions. Sometimes the written instructions were not enough and then the person supervising the test had to provide extra explanations orally. It was considered more important that the participants understood the tasks than that everyone got the exact same instructions. The tests were carried out in a room shred with a small team with resulted in some noise and movement in the room which varied a bit from time to time.

The process of designing the stimuli for each test consisted of brain storming sessions to come up with ideas of how to get accurate data for the activity. The major part of the design phase was thus to select proper stimuli. Later during the data cleaning, the data recorded from stimuli related to the activity was labeled as data for that

specific activity while the data recorded during instruction stimuli was discarded. Since most activities were everyday activities which the participants had done in some form before, expert users were not used as proposed by [14]. This meant that it was easier to find participants, as any person could participate in our data collection. A challenge instead became to find participants willing to spend around 45 minutes split over two sessions to complete all tests. The data gathered from these tests make up the full data set. A more detailed description of each test can be found below.

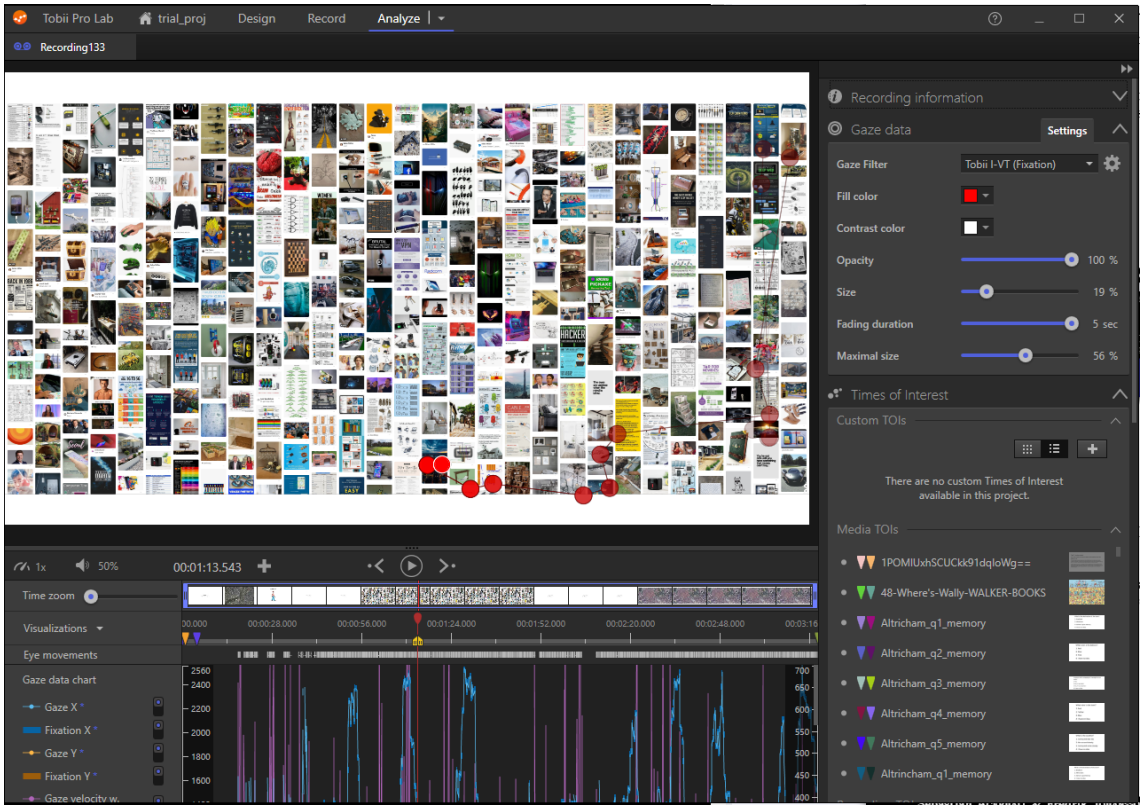


Figure 3.2: A screen shot from Tobii Pro Lab which was used for recording data

3.3.1.1 Cognitive Load

In order to induce cognitive load to the participant it was given a distraction. Cognitive load data was collected during each test along with the activities data. During the second half of the test the participants were subjected to a secondary distracting task, in addition to the primary task e.g. read a text, in order to increase the cognitive load. Two kinds of distractions were used to induce increased cognitive load. The first distraction consisted of listening to a four digit number sequence which were to be recited while performing the primary task. Sometimes the number sequences were exchanged with a simple two number addition which the participant should give the answer to. The participant got a new number sequence or addition task around every four seconds.

The other distraction type was without any extra external stimulus. For this distraction the participant was asked to count out loud from one and upward, or from 100, 200 or 300 and downward. This while performing the primary task. To make this task even more demanding, special rules were introduced during some of the instruction stimuli, e.g. ”skip every occurrence of the number 6” or ”exchange every occurrence of 8 with the word blue”. When the distractions were introduced the participants always did the ordinary counting first and then the test supervisor,i.e. one of the authors of this work, assessed if the difficulty could be increased. The Participant was supposed to be cognitively loaded but still be able to perform the primary task. The secondary task could therefore not be too demanding. In all tests with distractions, the first half of the test was done without distractions and the second half of the test was done with distractions. To ensure that the distraction pattern was not actually just a pattern of the stimuli, the stimuli order was reversed for half of the participants. This ensured that gaze pattern recordings was done for all stimuli both with and without distractions.

3.3.1.2 Reading test

The reading test consisted of six different texts of varying difficulty, font size and font, see figure 3.3 for an example text. Before each text the participant were given some brief context to the text to make it a little easier to grasp. And after each text the participant was asked to briefly summarize the content of the text orally or in writing. When the participant had read the text they clicked once with the mouse to mark that they had finished reading, after this the participant summarized the text before moving on to the next one. The test participant got to read three of the texts without distractions and three with distractions.

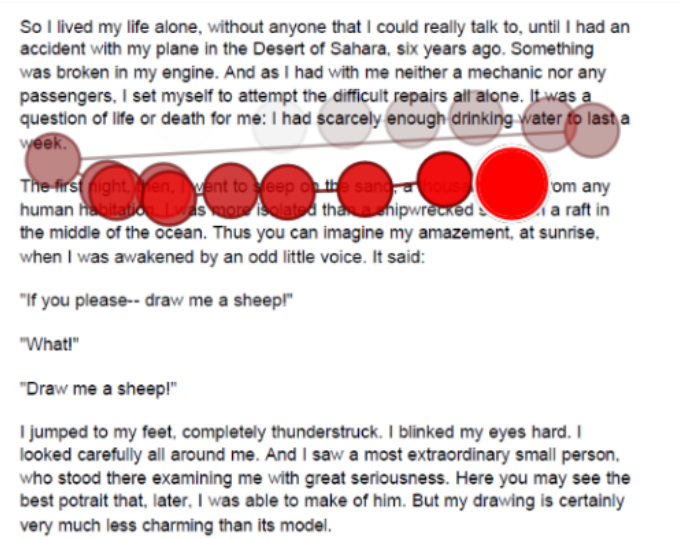


Figure 3.3: One of the texts from the reading test with a trace of the users gaze [5]

3.3.1.3 Scanning test

For this test the participant got to casually scroll through the 'Bestsellers in home' page on Amazon [48] which are located in a structured vertical pattern, see Figure 3.4. The participants were allowed to linger at offers they considered interesting but not to click on any of the offers. When the participant reached the end of the page the participant were to click to get to the next page. This was done for approximately three minutes and without any distractions.

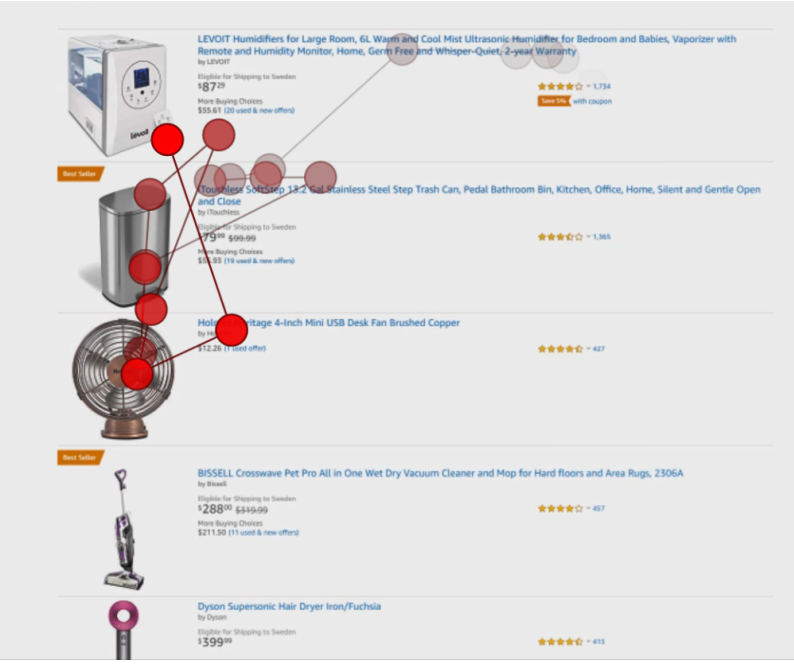


Figure 3.4: The scanning test with a trace of the users gaze

3.3.1.4 Searching images test

In this test the participant is presented with 10 different images. Four images were maps where the task was to find a specific location on the map. Four were so called Finding Waldo images where the objective is to find the character Waldo in a very detailed image with a lot of irrelevant objects. Two images depicted a grid of smaller images where the goal was to find one of the small images matching a description of the images content. The participants did five searching tasks with distractions and five without distractions. Figure 3.5 shows an example searching task with a map.

Figure 3.5: The search image test with a trace of the users gaze [6]

3.3.1.5 Searching text test

The participant was shown a question which were to be answered. The next stimulus was a text in which the answer could be found. The participants objective was to find the answer in the text as quick as possible. When the answer was found the participant clicked with the mouse in order to get to the next question. The participants did three searching tasks with distractions and three without distractions. This test looked similar to what is shown in figure 3.3.

3.3.1.6 Skimming test

This test was very similar to the reading test with the only difference that there were four texts and each text had a time limit before disappearing. The participant was asked to skim the texts rather than reading them carefully. The test was created in order to get data for the less thorough kind of reading done when skimming. Two texts were read without distractions and two were read with distractions. This test looked similar to what is shown in figure 3.3.

3.3.1.7 Memory test

This test was designed only to gather data for the cognitive load state. The aim of the cognitive load classification is to be able to tell if someone is distracted or not regardless of activity that is performed. To make sure that the classifier does not just learn for example cognitively loaded reading, data from an activity that the model had not been trained on was desirable. In this test the participant was asked to look at an image for 45 seconds and memorize it. After 45 seconds the participant

got five questions about the image to check what was remembered. The participant got two such memory tasks without distractions and two tasks with distractions.

3.3.2 Test group

The group of participants taking the tests consisted of 21 people in total. All of the participants worked at Tobii and thus probably had more knowledge and interest in eye tracking than a completely random population sample. This might be a source of bias. However none of the participants were directly involved in this project apart from contributing with data. There was a large spread of ethnic backgrounds in the participant group with the countries Chile, Greece, India, Iran and Sweden represented. The participants were in an age between 25 and 45 years old. Nine of the participants were female and twelve were male.

3.3.3 Second data collection

When the primary data collection was done and the model training had begun, some possible improvements in the data collection process were thought of. To see if these improvements had any effect on the data a new small scale data collection was done. Five participants from the previous data collection got to do the updated version. For this version the test was done in a lab environment to try to reduce the distractions while recording the focused state. Also the scanning activity was changed. In the case of the Amazon website there were labels and a short description for every offer. This had the effect that participants were reading partly during the scanning recordings and thus some data was wrongly labeled as scanning when it actually was reading. In the new version of the test the scanning activity was to scroll an Instagram feed. In the Instagram feed there were only pictures and no text and therefore all reading was eliminated. The models were then trained on data from the second collection for these five participants and compared with the models trained on data from the first collection for the same people.

3.4 Feature description

All eye data from the participants is exported from Tobii Pro Lab. This data contains recordings of all information screens and also invalid data which is data from when the eye tracker did not manage to track the eyes. The data exported from Pro Lab included many features. Some of these features are not available from the standard Tobii tech 4C 90Hz consumer eye tracker. Because of this a special licence was used with the eye tracker to get access to more features. Some of the features exported from Pro Lab was not necessary in this work and were therefore removed. After data processing the following features were left:

- Gaze Point X Left – The screen x-coordinates where the left eye is focused (in pixels)
- Gaze Point Y Left – The screen y-coordinates where the left eye is focused (in pixels)
- Gaze Point X Right – The screen x-coordinates where the right eye is focused (in pixels)
- Gaze Point Y Right – The screen y-coordinates where the right eye is focused (in pixels)
- Eye Position X Left – The left eye positional x-coordinate in relation to the tracker (in mm). The eye position coordinates gives information about were the users head is located in space relative to the tracker.
- Eye Position Y Left – The left eye positional y-coordinate in relation to the tracker (in mm)
- Eye Position Z Left – The left eye positional z-coordinate in relation to the tracker (in mm)
- Eye Position X Right – The right eye positional x-coordinate in relation to the tracker (in mm)
- Eye Position Y Right – The right eye positional y-coordinate in relation to the tracker (in mm)
- Eye Position Z Right – The right eye positional z-coordinate in relation to the tracker (in mm)
- Pupil Diameter Left – Pupil Diameter of the left eye (in mm)
- Pupil Diameter Right – Pupil Diameter of the right eye (in mm)
- Validity – 1 if at least one of the eyes are found by the tracker, zero otherwise.
- Eye Movement type – 1 if the current observation belongs to a fixation, zero for saccades or unclassifiable eye movements.
- Mean Fixation Duration – The mean fixation duration for each activity and each participant
- Var Fixation Duration – The variance of the fixation duration for each activity and each participant
- Sd Fixation Duration – The standard deviation of the fixation duration for each activity and each participant
- Fixation Rate – Number of fixations per time frame.
- Mean Saccade Length – The mean saccade length for each activity and each

participant

- Var Saccade Length – The variance saccade length for each activity and each participant
- Sd Saccade Length – The standard deviation of the saccade length for each activity and each participant

Of these features the mean, var and sd for both fixations and saccades as well as fixation rate were computed from other features. This results in a system delay because of feature computations.

3.5 Data cleaning

The raw data included many data points from uninteresting stimuli, e.g. instructional texts and pause frames. For these stimuli the actual activity of the participant is unknown and due to this, the data points related to these stimuli were removed. When the eye tracker can not see the eyes it results in an invalid data point. Since there could be information in the blinks of the participants, shorter periods of invalid data was kept. To keep data when the participant blinks, invalid data points were filled with the values of the previous data point. This has the effect that instead of not having any data at all during a blink, the gaze is instead stationary. Invalid data for less than 25 consecutive data points, i.e. approximately 275 ms, was included in the data. The number 25 was found empirically by cross validation over an interval from 1 to 50 consecutive data points of invalid data. 275 ms also coincides quite well with the duration of blinks, which according to [49] is between 100 ms and 400 ms. The validity feature was used to mark when the gaze was stationary due to a blink or due to simply being stationary. Zeroes in validity means invalid data, i.e. no eyes found, due to e.g. a blink. Ones in validity means valid data, i.e. data from at least one eye.

Some activities took longer time to record than others which resulted in some activities having more data than others. To make sure that the models would not get biased towards one specific activity the data set was balanced by random down sampling of the more common activities, so that every activity had the same number of observations. The same procedure was applied for the cognitive load classification task so that we trained the models on data that had the same number of distracted observations as non-distracted observations. When testing and evaluating the cognitive load classifiers only data from the memory activity for the test and validation participants was used. This to see if the cognitive load classification generalized to unknown activities.

To match the data output from the eye tracker a one euro filter was implemented in the data cleaning process. The one euro filter is an algorithm which uses a first degree low-pass filter with a cutoff frequency which varies with the update rate. A low update rate gives a low cutoff frequency to reduce jitter and a high update rate

gives a greater cutoff frequency to reduce lag [50].

3.6 Time frame

To facilitate the training of the models the data was split from one long sequence of observations to several small sequences. Hereinafter such a small sequence will be referred to as a sample. The length of each sample i.e. the number of observations in the sample is called the time frame. The time frame is a hyperparameter that was tuned individually for each model. The number of samples in our data thus becomes the number of observations divided by the time frame.

3.7 Feature transformations

The recorded data was in raw data format, i.e. no filtering or smoothing was applied. However to mimic the behavior of the 4C tracker in our training data we applied the same one euro filter as the commercial trackers use. Due to confidentiality the exact properties of the filter can not be disclosed. Moreover the features were normalized within each sample so that the model would be invariant to the size of the stimuli, e.g. the size of the text when reading. Since only the general gaze patterns were interesting and not the actual location of the gaze, only relative changes within each feature was considered. This means that for each point in time the value of each feature was replaced by the difference between the current value and the previous value, see equation (3.1)

$$X_t^{transformed} = X_t - X_{t-1}, \tag{3.1}$$

where X is the feature matrix for each sample. In the sample the number of rows corresponds to the time frame and the number of columns corresponds to the number of features. The data set was also balanced so that each activity in the data set has an equal number of samples to avoid bias towards a certain activity.

3.8 Model selection

This section presents why the models were chosen, how the features were selected and how the hyperparameters of the models were tuned.

3.8.1 Model architecture

The different model architectures (LSTM, CLDNN, HMM) were chosen for this study because they have shown good performance for modelling sequential data [51], [30], [52]. The models have quite different behaviors and different complexity, with HMM being the least complex and CLDNN being the most complex, therefore different results from the models was expected. More complex models are able to learn more complex patterns but they also require more data and are more prone to overfitting [47]. Overfitting means that the models fits very well on the training data but that they have found a very specific pattern for the training data so the models are not able to generalize to new data [47]. LSTM is a commonly used model architecture for deep learning on sequential data [51], therefore it was included in this study. The CLDNN is an interesting extension to the LSTM model granting possible advantages from CNNs to the LSTM. Furthermore HMM was chosen since it is the most commonly used non-deep learning model for sequential data and it can provide more interpretable results than the deep-learning models. Even though model parameters can be learned from data, there are still many parameters that has to be decided manually. The complete model architecture can be found in Section A.1. For the CLDNN model the same core architecture as presented in [30] was used, but with hyperparameters, filter size, number of filter etc, optimized for this task, see Section A.1. For the LSTM network a two layer model was used as recommended by [51]. To perform classification with the HMM, one HMM was created for each status, i.e. one for each activity as well as one for cognitive load. These models were trained on data from its corresponding test. In the prediction stage the classification was done by choosing the status corresponding to the HMM with the maximum likelihood for the data.

3.8.2 Feature selection

The features available in the training data is described in Section 3.4. Out of the available features, a subset was selected to reduce the model complexity. The subset was selected by running cross validation where one feature were excluded at a time. If the Cross Validation (CV) score decreased when a feature was removed the feature was considered to contribute with valuable information. If the CV score increased or was unchanged when the feature was excluded, the feature was considered redundant and was removed from the analysis. With this procedure Gaze Point, Eye Position and Validity were chosen as important features. Since the purpose of this study was not to investigate feature importance, the feature selection was not studied too thoroughly. The features selected in this study were only selected based on their impact on prediction performance for the current data set. To be able to draw final conclusions about feature importance a more thorough research should be pursued.

3.8.3 Hyperparameters

Hyperparameters are parameters that are predefined in the models. These parameters can not be learned but have to be set by the user in an adequate way. Hyperparameters can have a big impact on the prediction performance and is thus an important factor to consider [51]. One way to find the best hyperparameters is simply to run through all combinations of hyperparameters and keep the models which have the best performance. This technique is called grid search[53]. The number of combinations of hyperparameters grows quickly with the number of hyperparameters and thus it quickly becomes unfeasible to evaluate all combinations. One technique for coping with this is Bayesian optimization.

3.8.3.1 Bayesian Optimization

Bayesian optimization [54] makes random selection of hyperparameters from some prior distribution in each iteration. Based on some given loss function the prior distribution is updated after each iteration to focus on a parameter space which seem to minimize the loss. One can say that the algorithm uses the results from previous evaluations to make educated guesses about the next set of hyperparameters for evaluation. This greatly reduces the number of required evaluations compared to grid search. The python package hyperopt [55] was used to implement the Bayesian optimization.

3.9 Training and Evaluation

The implementation, training and evaluation of the models were all done with the Python package Keras [56], except for the HMM model which was implemented from the package Hmmlearn [57]. The full data set was split into one training set and one test set. The training set consisted of 17 participants and the remaining four participants formed the test set. 8 fold cross validation (CV) was used on the training set to fit and evaluate the models on the data. Cross validation means that the data set is divided into different parts, in this case 8 parts. One part is then held out for valdidation while the models are trained on the rest of the data. The trained models are then evaluated on the remaining part and the prediction score is noted. Then another part is chosen as the validation part and the same procedure is repeated until all parts have been validation data. The Cross validation score is the the mean score from all validation parts. Since one participant might have a specific activity pattern the risk of getting overlap between the data sets was considered high if the data sets would have been random splits of the participants' samples. Because of this each fold data set were made with data from separate participants. With 17 participants, the 8 fold CV gives that each fold consist of 2-3 participants. Thus for each fold the models were trained on 14-15 participants and validated on 2-3 participants. After data cleaning and balancing each participant had around

three minutes of data per activity. The reason for it being three minutes is that the scanning activity had the least amount of data which was three minutes, and the other activities data length were cut to match this in order to get a balanced data set. With a sampling rate of 90 observations per second this yielded around 17 thousand data points per activity and participant. All activities except scanning had more than three minutes of data which means that only a sub sample of activity data was used for those activities. The validation set was used to estimate the generalization error during the training phase in order to compare models and to tune hyperparameters. The test set remained untouched until a final model had been chosen. The final model was then evaluated on the test set to give the final performance of the model.

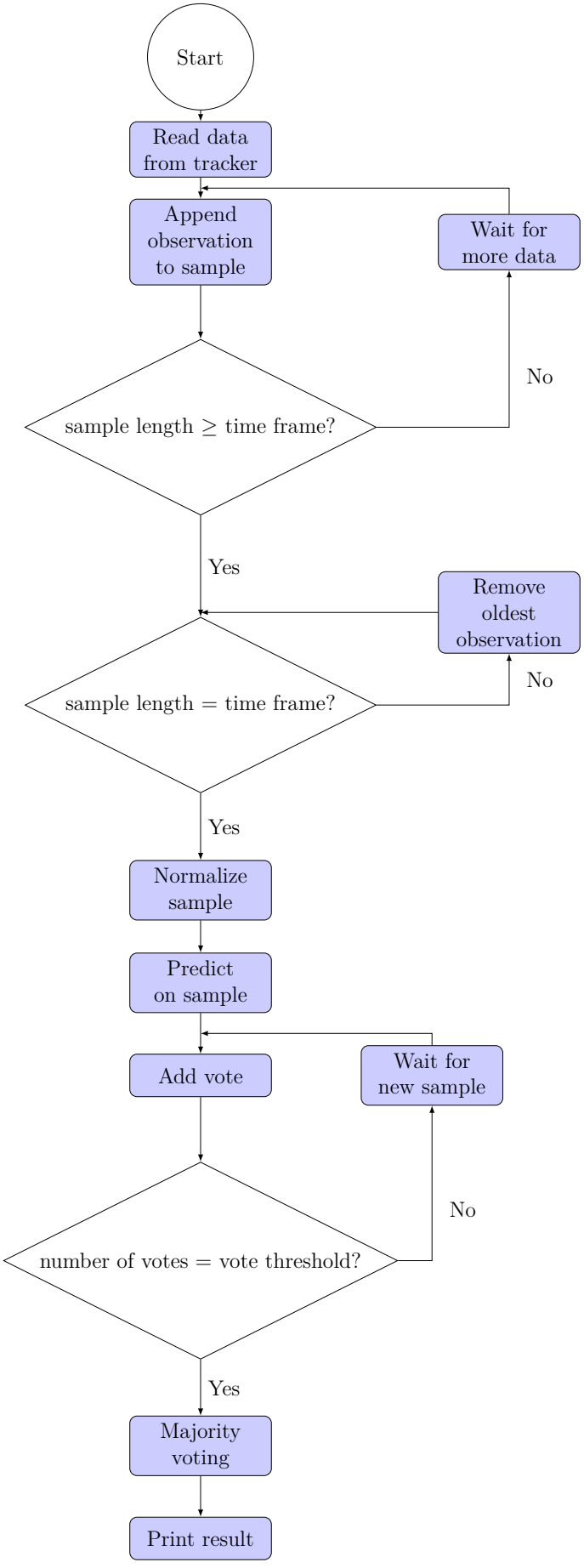
Since the activities are not mutually exclusive with cognitive load we trained and evaluated activity classifiers and cognitive load classifiers separately, but with the same procedure in both cases.

During the evaluation of the models it was decided that the predictions could be smoothed over a longer period of time by an approach here called majority voting. Majority voting means that the final prediction relied on several consecutive predictions. A certain voting threshold was set, i.e the number of predictions to smooth over. When the vote threshold was reached, the final prediction was chosen as the most frequently occurring class among these predictions. This was done because it was assumed that one activity usually lasts for a while. For example, if the model predicts that a user is reading for 15 seconds, then scans for 1 second and then reads again for 15 seconds it is probable that the user was reading the entire time.

3.10 Predictor prototype

The predictor prototype was developed in C#. It uses the Tobii Stream Engine [58] API to read data from the Tobii Tech 4C 90 Hz eye tracker and the library TensorflowSharp [59] to make predictions. The predictor prototype uses a model trained in Keras and saved as a protocolbuffer file to be compatible with Tensorflow. The Figure 3.6 shows the execution order in the prototype. First the prototype reads the data from the tracker via the Tobii Stream Engine API 3.6. With 90Hz in sampling rate the prototype gets one new data point approximately each 0.011s. Each new data point is appended to a list until the list has the length equal to the time frame for the current model, i.e. a full sample is reached. the full sample is then normalized as described in Section 3.7. The normalized sample is then predicted on. To perform the majority voting this prediction is appended to a voting list. If the list has less elements than the vote threshold the previous steps are repeated until enough predictions have been appended so that the number of votes matches the vote threshold. When the voting list contains the same number of votes as the specified vote threshold, the voting is performed, a final prediction is given and the list is cleared.

Figure 3.6: Flow chart for prediction prototype



4

Results

This chapter begins by presenting the f1-scores for activities and cognitive load. Then the results with majority voting is presented along with f1-scores and a comparison between filtered and unfiltered data. After this the results of the second data collection is presented. Finally some results from the predictor prototype is presented.

4.1 Prediction performance

In this section the prediction performance of the different model architectures with tuned parameters is described. The models were evaluated with respect to the f1-score as well as comparison of the confusion matrices. The scores presented here is related to the prediction performance on the previously unseen test set. The confusion matrices with corresponding f1-scores for five activities and cognitive load can be found in Figures 4.1 and 4.3. From the confusion matrices in Figure 4.1 it seems like search text, reading and skimming are easily intermixed by the models. Since the activities skimming and search text seemed to resemble reading it was decided to disregard the skimming and search text data all together and only use the reading data. The confusion matrices with corresponding f1-scores for only one kind of reading can be found in Figure 4.2. The f1-scores can be compared with always guessing reading which is the most common activity in the data set. In the case of three activities where the prediction is always reading the overall average f1-score would be 0.21. The case with five activities the average f1-score would be 0.07.

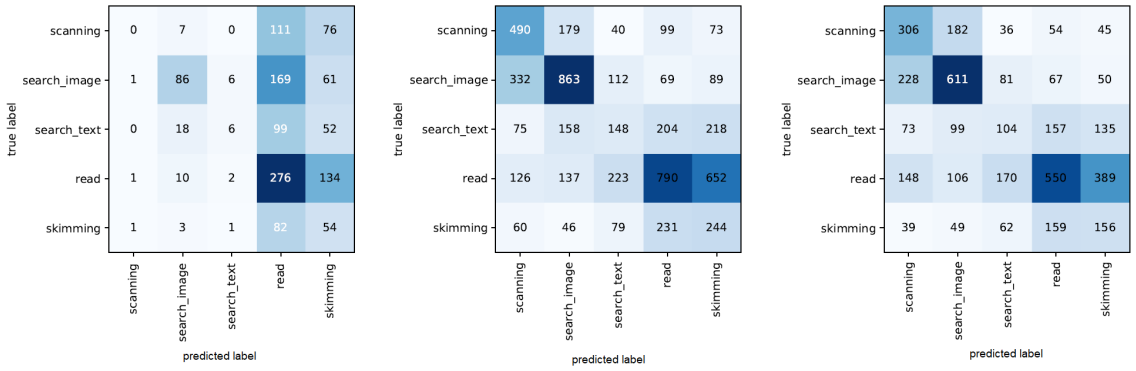


Figure 4.1: HMM f1-score: 0.23 LSTM f1-score: 0.41 CLDNN f1-score: 0.39 Confusion matrices without majority voting for the different model architectures for the activity classification task. All five activities included.

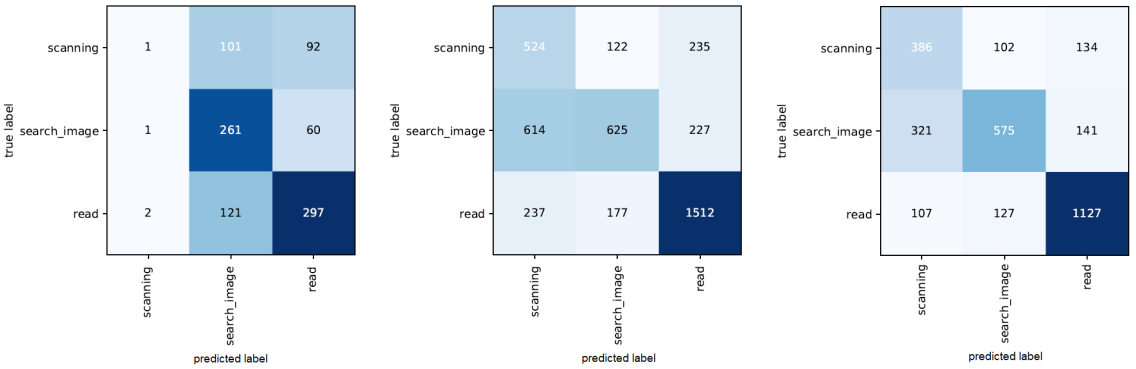


Figure 4.2: HMM f1-score: 0.45 LSTM f1-score: 0.59 CLDNN f1-score: 0.66 Confusion matrices without majority voting for the different model architectures for the activity classification task. Three activities.

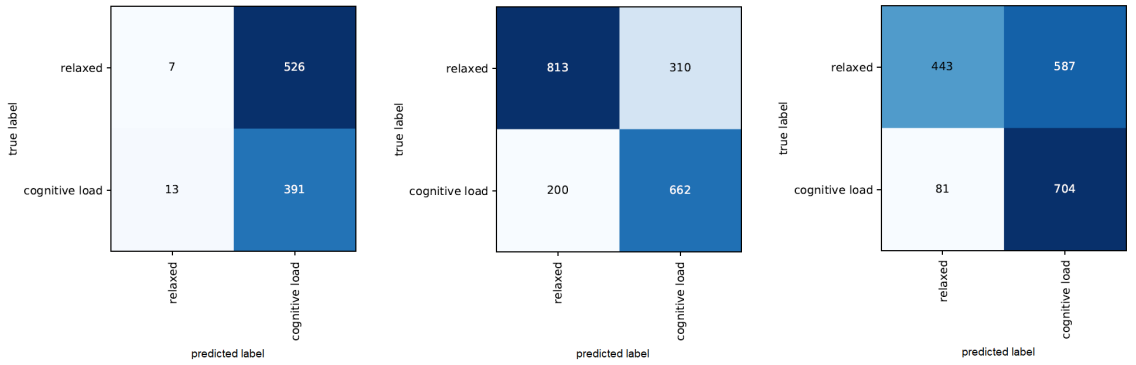
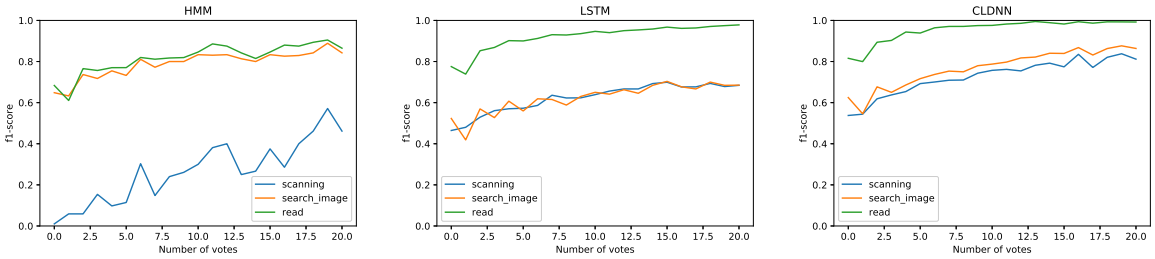


Figure 4.3: HMM f1-score: 0.31 LSTM f1-score: 0.74 CLDNN f1-score: 0.62 Confusion matrices without majority voting for the different model architectures for the cognitive load classification task.

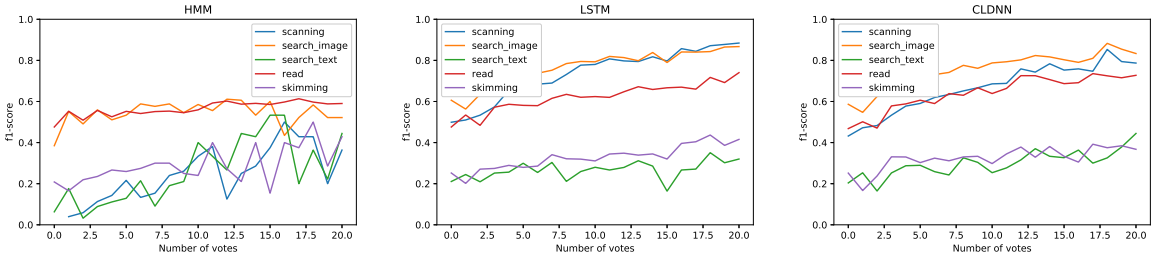
4.1.1 Majority votes

Some of the activities included subpatterns from other activities, e.g. there was some reading in the scanning activity, this resulted in some wrong predictions within activities. In order to smooth the prediction results a majority vote over several predictions was introduced. Figures 4.4, 4.5 and 4.6 shows the change in f1-score depending on the number of predictions used in the majority vote for activity classification and cognitive load classification.



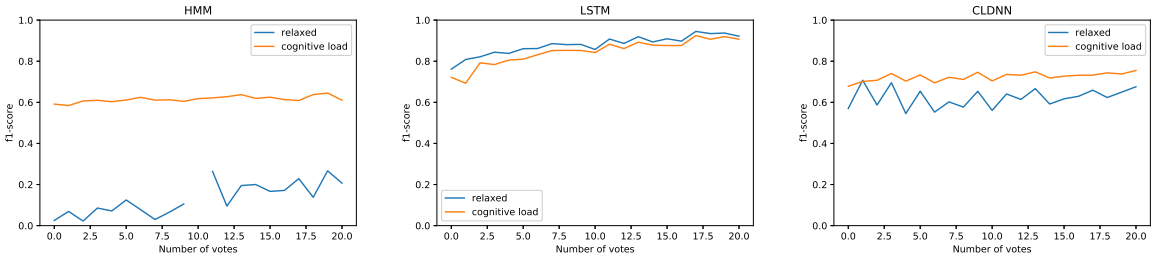
(a) HMM **(b)** LSTM **(c)** CLDNN

Figure 4.4: f1-score vs number of majority votes for three activities.



(a) HMM **(b)** LSTM **(c)** CLDNN

Figure 4.5: f1-score vs number of majority votes for five activities.



(a) HMM **(b)** LSTM **(c)** CLDNN

Figure 4.6: f1-score vs number of majority votes for cognitive load.

The graphs indicate that a large number of votes increases the prediction performance. The introduction of majority voting induces increased latency in the predic-

tion and thus the predictions take more time to compute. The choice of an adequate number of votes has to consider the trade-off between prediction speed and prediction performance.
The confusion matrices for predictions with 20 majority votes can be seen in Figures 4.7, 4.8 and 4.9.

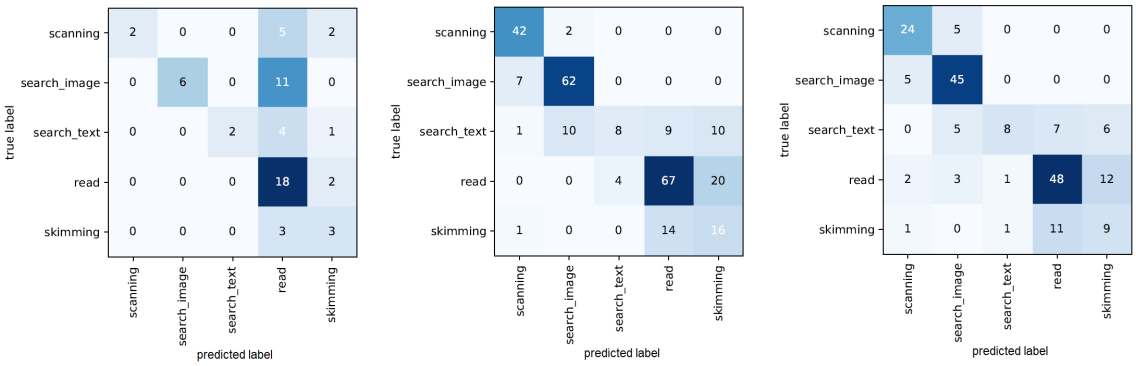


Figure 4.7: HMM f1-score: 0.47 LSTM f1-score: 0.65 CLDNN f1-score: 0.63
Confusion matrices with 20 majority votes for five activities and the average f1-score over those activities.

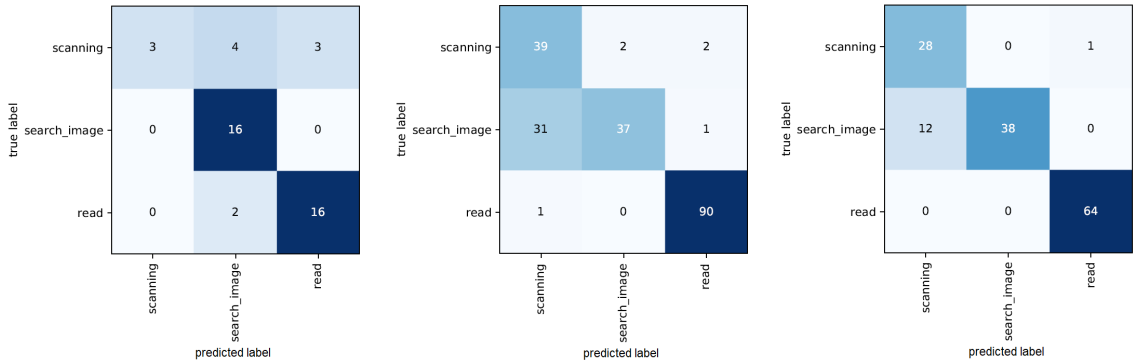


Figure 4.8: HMM f1-score: 0.72 LSTM f1-score: 0.78 CLDNN f1-score: 0.89
Confusion matrices with 20 majority votes for three activities and the average f1-score over those activities.

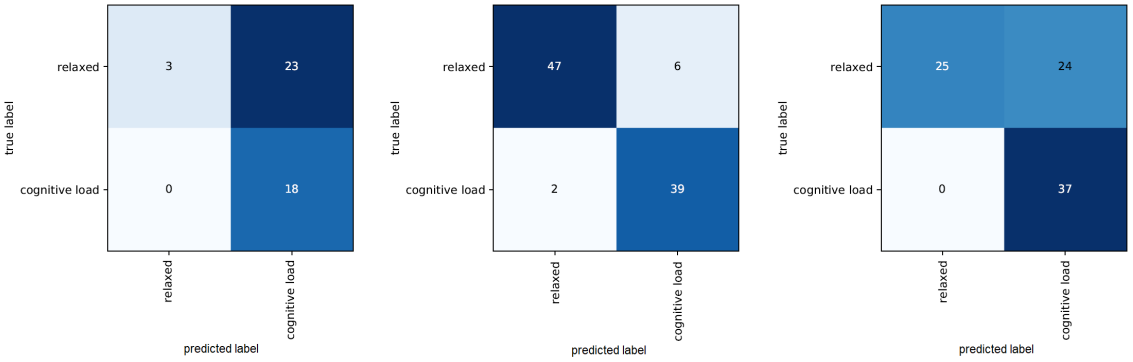


Figure 4.9: HMM f1-score: 0.41 LSTM f1-score: 0.91 CLDNN f1-score: 0.72 Confusion matrices with 20 majority votes for cognitive load and the average f1-score over those states.

These results are obtained when the data is filtered with a one euro filter. This filter is automatically applied to the output of the eye tracker in the stream engine. Since the model should match the data acquired in real time from the tracker the training data was also filtered with an identical one euro filter as found in the stream engine. When comparing models trained on unfiltered data the results have improved some. A comparison of the 8 fold cross validated f1-scores for unfiltered and filtered data without majority voting for three activities can be found in Table 4.1. The 'nan' values for the HMM is because the model never predicted anything as scanning and thus the precision for the scanning activity is zero and then the f1-score can not be computed. This means that the model was completely unable to detect the scanning activity. Table 4.2 shows the corresponding values for cognitive load.

	Unfiltered	Filtered
CLDNN	0.67	0.64
LSTM	0.66	0.65
HMM	nan	nan

Table 4.1: Three activities mean f1-scores for 8 fold cross validation without majority voting on unfiltered data and data filtered with a one euro filter.

	Unfiltered	Filtered
CLDNN	0.52	0.65
LSTM	0.69	0.68
HMM	0.39	0.34

Table 4.2: Cognitive load mean f1-scores for 8 fold cross validation without majority voting on unfiltered data and data filtered with one euro filter.

4.2 Second data collection

Table 4.3 shows a comparison of the f1-scores for the different models trained on data from the original data collection and the models trained on data from the second data collection for the activity classification. The f1-scores for the second data collection are better.

	First data collection	Second data collection
CLDNN	0.51	0.69
LSTM	0.50	0.64
HMM	0.36	0.40

Table 4.3: Three activities mean f1-score for 8 fold cross validation for models trained on data from the second data collection and models trained on data from the first data collection.

Table 4.4 shows the same comparison for the cognitive load classification. Data from the second collection seems to outperform data from the first collection in this case as well.

	First data collection	Second data collection
CLDNN	0.60	0.67
LSTM	0.51	0.66
HMM	0.09	0.57

Table 4.4: Cognitive load mean f1-score for predictions made on the memory activity for models trained on the data from the first data collection and models trained on data from the second data collection.

4.3 Choosen hyperparameters

The result from the bayesian optimization gave the model hyperparameters which gave us the best results. Table 4.5 gives the Hyperparameters used for activity classification and table 4.6 gives the hyperparameters used for cognitive load. The reason why some of the fields are left blank is because all parameters does not apply to all models.

	LSTM	CLDNN	HMM
Hidden units	347	195	
Time frame	98	139	450
Kernel size 1		6	
Kernel size 2		10	
Num filters 1		105	
Num filters 2		223	
Components scanning			4
Components search image			9
Components search text			7
Components read			9
Components skimming			8

Table 4.5: Hyperparameters used for activities classification

	LSTM	CLDNN	HMM
Hidden units	106	311	
Time frame	212	232	450
Kernel size 1		13	
Kernel size 2		13	
Num filters 1		56	
Num filters 2		116	
Components no cognitive load			4
Components cognitive load			9

Table 4.6: Hyperparameters used for state classification

4.4 Predictor prototype

The predictor prototype is able to read the gaze data from the tracker and to perform predictions on the data. The tracker continuously feeds the prototype software with data and then the prototype performs data preprocessing to make the format of the live data correspond to the training data. Since the predictions are done on one sample with the number of observations corresponding to the time frame the prototype saves observations until there is a complete sample. A prediction is then made based on this sample. The fact that the predictor waits until a complete sample has been collected before making the initial prediction induces an initial delay proportional to the sampling frequency and time frame. Afther this a new prediction is made every time a complete new sample is full.
On top of these predictions a majority vote over a set number of predictions is used to smooth the prediction. This induces a lag corresponding to the number of votes multiplied with the time each prediction takes. A system with Nvidia GeForce GTX 1050 GPU and Intel Core i7-7700HQ CPU has a mean prediction time of 26 ms over 1000 predictions. With a sample rate of 90Hz and the activity model requiring a

time frame of around 139 observations and the cognitive load model requiring a time frame of 212 observations this gives a delay of 1.54s and 2.36s, respectively. The additional delay from the majority voting is therefore 1.52s or 2.36s times the number of votes.

5

Discussion

This chapter will present a discussion about the design of the tests, how the models can be used and the filtering of data. Further, possible flaws of the model evaluation process and ethical aspects about this kind of technology is discussed. To finish off there are some thoughts about what work can be done in the future.

5.1 Prediction performance

From the confusion matrices in Section 4.1 it seems that the LSTM model is the best for cognitive load classification while the CLDNN model is the best for activity classification, when only three activities are considered. The HMM model seems to underperform against the deep learning models which makes sense since the HMM model does not have the same ability to learn feature representations as the deep learning models. The HMM model would maybe have worked better with more manually crafted features. Without majority voting the top f1-score for activity classification is 0.66 and for the cognitive load classification the top f1-score is 0.74. The activity classification score can be compared with the results of [11] and [13] who had similar activities and got an average f1-score of 0.73 and 0.72, respectively. The performance of the activity classification models in this study is slightly worse. However [11] and [13] did not predict in real time and therefore they could afford more preprocessing and exploit data over a longer time period.

When it comes to the cognitive load classification it is harder to find previous results to compare with. [16] worked on confusion classification which is somewhat related to cognitive load. They achieved sensitivity of 0.74 and specificity of 0.71 with their RNN approach which is in the same range as our obtained sensitivity and specificity of 0.80 and 0.68 respectively, from LSTM confusion matrix in Figure 4.3. These results strengthens the hypothesis that there is valuable information about a users affective state in their eye movements.

To be able to use these models to adapt user interfaces in real time, in a consumer product, the f1-scores should be above 0.9. Both scores are below the accepted threshold for real time user interface adaptation. The models are however far better than random guessing and could probably be used in settings where the accuracy

requirements are lower than for real time interface adaptation. One possible application could be to give information to commentators of e-sport tournaments. If the models could give a rough estimate on whether a player is cognitively loaded or not it could help the commentators to see if a player is in trouble or not.

5.1.1 Majority voting

When majority voting was applied a clear increase in f1-scores could be observed. For activity classification the best score rose from 0.66 to 0.89 and for cognitive load classification the f1-score rose from 0.74 to 0.91, when comparing 20 votes with no votes. As mentioned in Section 4.1 there is a trade off between prediction speed and f1-score. The majority voting scores outperform the scores that [11] and [13] obtained, while still allowing online predictions.

The optimal balance between speed and f1-score depends on the application. In Figures 4.4 and 4.6, the performance increase seems to saturate above 7-10 votes. Since the prediction speed decreases linearly with increased number of voters, it is usually a good idea to choose number of votes to be close to the inflection point, i.e. 7-10 votes, to get the best speed-accuracy trade off. Since the optimal time frames for the two models were 1.52s and 2.36s respectively, prediction delay with 7-10 votes should be around 10-20 seconds.

5.1.2 Filtered data

Performance has shown to increase without the one euro filter. This might be due to loss of smaller and faster eye movements when the filter is applied. The predictor uses Tobii Stream Engine to get gaze data from the tracker. Stream Engine automatically applies a one euro filter on the data which is the reason why we trained our models with filtered data as well. For this project it would have been beneficial to have access to unfiltered gaze data from the eye tracker. To train models and create a predictor not using filtered data is work for the future.

5.2 Possible improvements

This section presents possible improvements to be made in the design of tests and use of models.

5.2.1 Test design

One of the greatest challenges in this project was to collect correctly labeled data that accurately represents reality. Especially in the case of cognitive load data, it

is hard to know if the data is correctly labeled since it is hard to know a person's actual level of cognitive load. In this case the labeling of cognitive load was based on the assumption that one basic main activity requires low cognitive load, while doing the main task as well as the secondary task requires high cognitive load. It is quite certain that the dual task requires higher cognitive load than the corresponding single task. Reading and counting is harder than just reading. However there might also be a difference in the cognitive load required by the different main activities. For example some participants were not native English speakers and reading in a foreign language might require higher cognitive load than searching for a place on a map. Measurements of brain activity would probably have given a more objective and accurate measures of cognitive load and thus could have enabled more accurate labeling. Equipment for measuring brain activity was not available in this project but could be an interesting thing to consider for future research.

The data collection was carried out in a room where other people worked. This had the effect that sometimes people were talking during tests and there was other distracting things happening in the same room. These surrounding distractions might have had the effect that some recordings labeled as low cognitive load was actually high cognitive load. The scanning activity during the data collection consisted of a list of product pictures with a text for each product. This made people look at the pictures, and also read the text, which probably resulted in reading patterns labeled as scanning. The disturbance from people talking as well as the design of the scanning activity was the main reasons why a second round of data collection with five participants was carried out. The thought with the second round was to get data to be able to compare with the first collection to see if the people talking as well as mixed reading with scanning had any effect on the results. The second data collection gave improvements in the prediction performance. Unfortunately there was not time to do a full scale second data collection, but the improvements with just five participants indicates that it is possible to create a data set with less noisy labels. To think through the creation of the test more and try to maximize the control over variables, such as the sound level in the room, would have been beneficial in the end.

5.2.2 Models

The confusion matrices indicated that some activities had common sub patterns and that the models seemed to learn these sub patterns rather than the main activity, e.g. the scanning activity included some reading patterns from e.g. headlines. this opens up for the possibility to use these models as feature extractors for higher level activity classifiers. There could be activities that comprises of several smaller activities, for example "browsing" could include some reading and some searching. The models developed in this work could work as feature extractors on the raw data which another machine learning model could use as input to predict those higher level activities.

5.3 Possible flaws

Even though the models were evaluated on previously unseen test data the results only show the prediction performance for activities in the same context as in the data collection. In the data collection each activity was done separately for a coherent period of time. In a real case, users might switch tasks more frequently and the users will also perform many activities that the models do not recognize. How the models behave in this context is still to be further investigated. The prototype software that predicts the user activity can perform predictions with an, to Tobii, acceptable delay, but to get a quantitative measure of the real world prediction performance is hard since there is no well defined ground truth. One way to get this measure is to allow users to try the prototype and manually label the activities they are currently doing and compare it with the predicted labels. The problem with this approach is that it is intrusive on the user's behaviour and might therefore prevent the user from performing the activities in a natural fashion.

5.4 Ethics

This study indicates that there is a lot of information about the user just in their gaze patterns. The focus in this study has been on the activities scanning, searching and reading as well as cognitive load, but it is reasonable to assume that the gaze patterns can give information of other activities as well. Further development in this area could enable tracking of users' mood and emotional state as well as their activities. This kind of tracking could be used to find out a users true thoughts and feelings. This could help a user become more aware of itself but it could also become a way to use this information about a user for one's own interest. How could this be used in a place where free speech is not allowed and this enables you to deduce someones opinions based on the response of the eye to specific questions? It could be used to change stimulus to be more pleasing to the user without the user knowing. It could be used by companies to make more appealing commercials. If a banner on a website is the cause of bad feelings it could be changed in order to please the crowd better. When reading terms and conditions for signing an insurance or a contract the company could chose not to sign if the user gives unwanted signals. To conclude, this kind of tracking could be considered a violation of the user's privacy. When this kind of technology arrives it is important that the technology providers take their responsibility to ensure ethical use of their product. Licenses and contracts should be formed to protect the users privacy so that the user can enjoy the perks of aware user interfaces without feeling under surveillance.

5.5 Future work

There is an infinite number of different activities and user states that can occur in front of a laptop. In this study three activities and one state have been tracked and to some extent it seems to be possible to predict these activities and states based on gaze patterns. A natural extension to this study would be to further explore what other activities and states that are possible to track. As previously discussed it could be valuable to try to define low level activities on which models similar to the models in this study can be trained and then use these low level activities to formulate higher level composite activities.

The scope of this project was to only consider eye data, but it is fair to assume that data from other kinds of user interactions such as mouse clicks or key presses could give additional valuable information. Further research could try to include other data sources in order to get more information by the user and thus enable more accurate predictions.

6

Conclusion

This study has explored how eye movement data and machine learning can be used detect if a user is reading, searching for something or scanning page content as well as detecting if a user is cognitively loaded or not. For this task three models were evaluated, HMM, CLDNN and LSTM. For the activity classification task the CLDNN model performed best with an f1-score of 0.62. The LSTM model performed best with an f1-score of 0.74. when predictions were smoothed over a larger time window the prediction performance increased up to around 0.90 f1-score for both activities and cognitive load classification. In terms of prediction speed the prototype performs well and is able to make predictions with a few seconds of delay, which is acceptable for real time user interface adaptation according to Tobii. However in order for Tobii to consider this to be applicable for real time user interface adaptation, accuracy would have to reach well above 0.9. Thus the accuracy of the models are a little bit too low. With a larger amount of carefully annotated data it might still be possible to increase the performance of the models and it is also possible that inclusion of other data sources might further increase the performance

Bibliography

[1] "How eye tracking works," https://www.tobiipro.com/learn-and-support/learn/eye-tracking-essentials/how-do-tobii-eye-trackers-work/, accessed: 2019-08-26.

[2] R. Bhatia, "When not to use neural networks," https://medium.com/datadriveninvestor/when-not-to-use-neural-networks-89fb50622429, 2018.

[3] P. Srivastava, "Essentials of deep learning : Introduction to long short term memory," https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/, 2017.

[4] C. Olah, "Understanding lstm networks," http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015.

[5] A. de SaintExupery, *The Little Prince*. Egmont Books Ltd, 1943, one of the pages from the book.

[6] "Google maps," https://www.google.com/maps, accessed: 2019-08-26.

[7] K. Rayner, "Eye movements in reading and information processing: 20 years of research." *Psychological Bulletin (1998)*, p. 372–422, 1998.

[8] J. A. Eduardo Velloso, Amy Fleming and H. Gellersen., "Gaze-supported gaming: Magic techniques for first person shooters. in proceedings of the 2015 annual symposium on computer-human interaction in play (chi play '15)." *CHI PLAY 2015*, p. 343–347, 2015. [Online]. Available: https://doi.org/10.1145/2793107.2793137

[9] Tobii, "Consumer laptops." [Online]. Available: https://gaming.tobii.com/products/laptops//

[10] P. Majaranta and A. Bulling, *Eye Tracking and Eye-Based Human–Computer Interaction*. London: Springer London, 2014, pp. 39–65. [Online]. Available: https://doi.org/10.1007/978-1-4471-6392-3_3

[11] A. Bulling, J. A. Ward, H. Gellersen, and G. Troster, "Eye movement analysis for activity recognition using electrooculography," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 4, pp. 741–753, April 2011.

[12] R. L. Canosa, "Real-world vision: Selective perception and task," *ACM Trans. Appl. Percept.*, vol. 6, no. 2, pp. 11:1–11:34, Mar. 2009. [Online]. Available: http://doi.acm.org/10.1145/1498700.1498705

[13] N. Srivastava, J. Newn, and E. Velloso, "Combining low and mid-level gaze features for desktop activity recognition," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, pp. 1–27, 12 2018.

[14] E. Aïmeur, F. Mpondo, F. Courtemanche, A. Dufresne, and M. Najjar, "Activity recognition using eye-gaze movements and traditional interactions," *Interacting with Computers*, vol. 23, no. 3, pp. 202–213, 03 2011. [Online]. Available: https://dx.doi.org/10.1016/j.intcom.2011.02.008

[15] R. Alghofaili, Y. Sawahata, H. Huang, H.-C. Wang, T. Shiratori, and L.-F. Yu, "Lost in style: Gaze-driven adaptive aid for vr navigation," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI '19. ACM, 2019.

[16] S. D. Sims, V. Putnam, and C. Conati, "Predicting confusion from eye-tracking data with recurrent neural networks," *CoRR*, vol. abs/1906.11211, 2019. [Online]. Available: http://arxiv.org/abs/1906.11211

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

[18] D. Svozil, V. Kvasnicka, and J. Pospichal, "Introduction to multi-layer feed-forward neural networks," *Chemometrics and Intelligent Laboratory Systems*, vol. 39, no. 1, pp. 43 – 62, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0169743997000610

[19] DeepAI, "Activation function," https://deepai.org/machine-learning-glossary-and-terms/activation-function.

[20] P. Bühlmann and S. van de Geer, *Statistics for High-Dimensional Data*, ser. Springer Series in Statistics. Dordrecht: Springer, 2011. [Online]. Available: https://cds.cern.ch/record/1414271

[21] V. Vapnik, "Principles of risk minimization for learning theory," in *Proceedings of the 4th International Conference on Neural Information Processing Systems*, ser. NIPS'91. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1991, pp. 831–838. [Online]. Available: http://dl.acm.org/citation.cfm?id=2986916.2987018

[22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Neurocomputing: Foundations of research," in *Neurocomputing: Foundations of Research*, J. A. Anderson and E. Rosenfeld, Eds. Cambridge, MA, USA: MIT Press, 1988, ch. Learning Representations by Back-propagating Errors, pp. 696–699. [Online]. Available: http://dl.acm.org/citation.cfm?id=65669.104451

[23] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time-series," in *The Handbook of Brain Theory and Neural Networks*, M. A. Arbib, Ed. MIT Press, 1995.

[24] N. Murray, "Convolutional networks," 2018, presentation made at Deep Learning Indaba, South Africa, 2018.

[25] M. Bodén, "A guide to recurrent neural networks and backpropagation," *Research gate*, 12 2001.

[26] J. Connor, L. E. Atlas, and D. R. Martin, "Recurrent networks and narma modeling," in *Proceedings of the 4th International Conference on Neural Information Processing Systems*, ser. NIPS'91. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1991, pp. 301–308. [Online]. Available: http://dl.acm.org/citation.cfm?id=2986916.2986953

[27] Pearlmutter, "Learning state space trajectories in recurrent neural networks," in *International 1989 Joint Conference on Neural Networks*, 1989, pp. 365–372 vol.2.

[28] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[29] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: continual prediction with lstm," in *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, vol. 2, Sep. 1999, pp. 850–855 vol.2.

[30] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, April 2015, pp. 4580–4584.

[31] L. E. Baum and J. A. Eagon, "An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology," *Bull. Amer. Math. Soc.*, vol. 73, no. 3, pp. 360–363, 05 1967. [Online]. Available: https://projecteuclid.org:443/euclid.bams/1183528841

[32] L. E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state markov chains," *Ann. Math. Statist.*, vol. 37, no. 6, pp. 1554–1563, 12 1966. [Online]. Available: https://doi.org/10.1214/aoms/1177699147

[33] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," in *Proc. IEEE*, 1989, section IV.

[34] ——, "A tutorial on hidden markov models and selected applications in speech recognition," in *Proc. IEEE*, 1989, section II.

[35] jlimahaverford (https://stats.stackexchange.com/users/71119/jlimahaverford), "Understanding elements that characterise hmm," Cross Validated, uRL:https://stats.stackexchange.com/q/175017 (version: 2015-10-01). [Online]. Available: https://stats.stackexchange.com/q/175017

[36] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," in *Proc. IEEE*, 1989, section II.B.

[37] J. Bilmes, "A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models," International Computer Science Institute Berkeley CA, 94704 and Computer Science Division Department of Electrical Engineering and Computer Science U.C. Berkeley, Tech. Rep., 1998, section 4.

[38] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," in *Proc. IEEE*, 1989, section II.C.

[39] Wikipedia contributors, "Baum–welch algorithm — Wikipedia, the free encyclopedia," 2019, [Online; accessed 29-April-2019]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Baum%E2%80%93Welch_algorithm&oldid=892460830

[40] J. Bilmes, "A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models," International Computer Science Institute Berkeley CA, 94704 and Computer Science Division Department of Electrical Engineering and Computer Science U.C. Berkeley, Tech. Rep., 1998, section 4.1.

[41] ——, "A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models," International Computer Science Institute Berkeley CA, 94704 and Computer Science Division Department of Electrical Engineering and Computer Science U.C. Berkeley, Tech. Rep., 1998, section 4.2.

[42] S. Tu, *Derivation of Baum-Welch Algorithm for Hidden Markov Models*, section 3.

[43] A. J. Viterbi, "Viterbi algorithm," *Scholarpedia*, vol. 4, no. 1, p. 6246, 2009, revision #91930.

[44] A. Maas, "lecture 8 hmm, viterbi," 2017, from CS 224S / LINGUIST 285 Spoken Language Processing course. [Online]. Available: https://web.stanford.edu/class/cs224s/lectures/224s.17.lec3.pdf

[45] R. Levy, "lecture 8 hmm, viterbi," p. 23, 2009, from Linguistics/CSE 256: Statistical Natural Language Processing course. [Online]. Available: http://idiom.ucsd.edu/~rlevy/teaching/winter2009/ligncse256/lectures/lecture_8_hmm_pos_tagging.ppt

[46] T. Kanungo, "Hmm tutorial," http://www.kanungo.com/software/hmmtut.pdf, 1999, tapas Kanungo, "UMDHMM: Hidden Markov Model Toolkit," in "Extended Finite State Models of Language," A. Kornai (editor), Cambridge University Press, 1999. http://www.kanungo.com/software/software.html.

[47] T. Hastie, R. Tibshirani, and J. Friedman, *Model Assessment and Selection*. New York, NY: Springer New York, 2009, pp. 219–259. [Online]. Available: https://doi.org/10.1007/978-0-387-84858-7_7

[48] [Online]. Available: https://www.amazon.com/b/ref=br_asw_smr?_encoding=UTF8&node=18505454011&pf_rd_m=ATVPDKIKX0DER&pf_

rd_s=&pf_rd_r=KRNP1BGCHCHHX8V1Q99S&pf_rd_t=36701&pf_rd_p=91247fb7-7595-414d-b9b0-f49b945644d3&pf_rd_i=desktop

[49] H. R. Schiffman, *Sensation and perception : an integrated approach.* New York: John Wiley and Sons Inc., 1976, no. 391.

[50] G. Casiez, N. Roussel, and D. Vogel, "1€ filter: A simple speed-based low-pass filter for noisy input in interactive systems," *Conference on Human Factors in Computing Systems - Proceedings*, 05 2012.

[51] N. Reimers and I. Gurevych, "Optimal hyperparameters for deep lstm-networks for sequence labeling tasks," *CoRR*, vol. abs/1707.06799, 2017. [Online]. Available: http://arxiv.org/abs/1707.06799

[52] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," in *Proc. IEEE*, 1989.

[53] W. Koehrsen, "Intro to model tuning: Grid and random search," https://www.kaggle.com/willkoehrsen/intro-to-model-tuning-grid-and-random-search, 2018.

[54] ——, "Automated model tuning," https://www.kaggle.com/willkoehrsen/automated-model-tuning, 2018.

[55] Y. D. C. D. D. Bergstra, J., "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," *Proc. of the 30th International Conference on Machine Learning (ICML 2013)*, 2013.

[56] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[57] S. e. a. Lebedev, "Hmmlearn," https://hmmlearn.readthedocs.io/en/latest/, 2015.

[58] "Tobii Stream Engine stream engine overview," https://developer.tobii.com/consumer-eye-trackers/stream-engine/, accessed: 2019-08-25.

[59] [Online]. Available: https://github.com/migueldeicaza/TensorFlowSharp

A

Appendix 1

A.1 Deep learning architectures

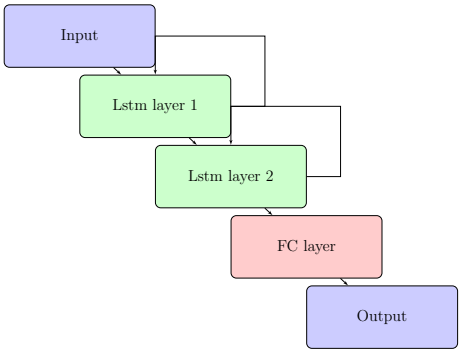


Figure A.1: Illustration of the order of the network layers in the LSTM model.

Listing A.1: LSTM architecture

```
    # Keras declaration of the LSTM model
    # classify_activities is True if we want to classify activities
    # and False if we want to classify cognitive load

    model = Sequential()
    model.add(LSTM(units = 347 if classify_activities else 106,
        recurrent_dropout = (0.25 if classify_activities else 0.0),
        input_shape=input_shape, return_sequences = True))
    model.add(Dropout(0.5))
    model.add(LSTM(units = 347 if classify_activities else 106,
        recurrent_dropout = (0.25 if classify_activities else 0.0)))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes))
    model.add(Activation('softmax'))

    model.compile(loss='categorical_crossentropy', optimizer='adam',
                  metrics=['categorical_accuracy'])
```

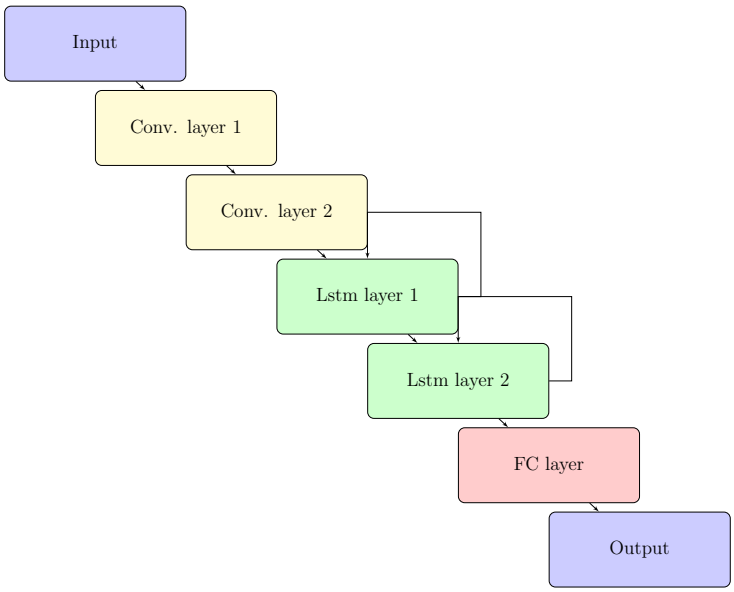


Figure A.2: Illustration of the order of the network layers in the CLDNN model.

Listing A.2: CLDNN architecture

```
1
2       # Keras declaration of the LSTM model
3       # classify_activities is True if we want to classify activities
4       # and False if we want to classify cognitive load
5
6       model = Sequential()
7       # Cnn part
8       model.add(Conv1D(105 if classify_activities else 56,
9           kernel_size = 6 if classify_activities else 13,
10          input_shape=input_shape))
11      model.add(Conv1D(223 if classify_activities else 116,
12          kernel_size = 10 if classify_activities else 13))
13      model.add(Dropout(drop_out))
14      #LSTM part
15      model.add(LSTM(units = 1 if classify_activities else 106,
16          recurrent_dropout = (0.25 if classify_activities else 0.0),
17          input_shape=input_shape, return_sequences = True))
18      model.add(Dropout(0.5))
19      model.add(LSTM(units = 287 if classify_activities else 106,
20          recurrent_dropout = (0.25 if classify_activities else 0.0)))
21      model.add(Dropout(0.5))
22      # DNN part
23      model.add(Dense(num_classes))
24      model.add(Activation('softmax'))
25
26      model.compile(loss='categorical_crossentropy', optimizer='adam',
27                    metrics=['categorical_accuracy'])
```