Deep Convolutional Networks with Recurrence for Eye-Tracking

LINUS HÄRENSTAM-NIELSEN
Deep Convolutional Networks with Recurrence for Eye-Tracking

LINUS HÄRENSTAM-NIELSEN
Abstract

This thesis explores the use of temporally recurrent connections in convolutional neural networks for eye-tracking. We specifically investigate the impact of replacing the convolutional layers in a regular CNN with convolutional LSTMs and replacing the fully connected feature layers with regular RNNs and LSTMs. This requires us to transition from a static single-frame input model to a time-dependent multiple-frame input model. Doing so naturally introduces extra complexity to the eye-tracking pipeline, so we highlight the advantages and disadvantages. Our results show that adding LSTM-cells to the convolutional layers and RNN-cells to the feature layers can increase eye-tracking performance, but also that LSTM-recurrence in the feature layers can be detrimental to performance.
Sammanfattning

Denna uppsats utforskar användandet av minnesceller i faltningsbase-
rade neuralnätverk för ögonföljning. Vi undersöker specifikt inverkan
av att byta ut faltningslager med faltningsbaserade LSTMer och att
byta ut de fullt sammankopplade feature-lagren med vanliga RNNer
och LSTMer. Vi beskriver hur man bör gå från en statisk modell som
tar en bild i taget som input till en tidsberoende modell som tar flera
bilder som input. Vi understryker även fördelar och nackdelar med en
sådan övergång. Vi visar att LSTM-celler i faltningslagren och RNN-
celler i featurelagren kan förbättra eye-trackingprestandan, men även
att LSTM-celler i featurelagren kan försämra prestandan.
Acknowledgements

I would first like to thank Tobii AB for the opportunity to do my masters thesis at their offices in Stockholm. I would also like to thank my supervisors Oscar Danielsson at Tobii and Atsuto Maki at KTH for all their advice and support throughout the project, and for their valuable feedback on the report. Finally, a big thanks to my family for always being there throughout the process, this would not have been possible without them!
# Contents

1 Introduction  
   1.1 Motivation .............................................. 1  
   1.2 Problem Statement .................................... 3  
   1.3 Thesis Overview .................................... 3  

2 Background and Related Works  
   2.1 Eye-tracking ........................................... 4  
      2.1.1 Eye-tracking using geometrical models ........... 4  
      2.1.2 Eye-tracking using neural networks ............. 5  
   2.2 Deep artificial neural networks ..................... 6  
      2.2.1 Fully connected layers .......................... 7  
      2.2.2 Convolutional layers ............................ 7  
      2.2.3 Pooling layers .................................. 8  
   2.3 Recurrent neural networks ............................ 8  
   2.4 Sequence-based CNNs ................................. 9  
      2.4.1 3D CNNs ......................................... 10  
      2.4.2 Feature layer RNNs .............................. 10  
      2.4.3 Convolutional RNNs .............................. 11  
   2.5 Ethical and societal Considerations .................. 13  
      2.5.1 Model bias ...................................... 13  
      2.5.2 Privacy ......................................... 14  

3 Methods  
   3.1 Networks architectures ............................... 15  
      3.1.1 Baseline architecture ........................... 16  
      3.1.2 Recurrence-based architectures .................. 18  
   3.2 Dataset structure .................................... 18  
   3.3 Loss function and performance measures ............. 19  

4 Experiments  
   4.1 Experimental setup .................................... 20  
   4.2 Eye-tracking dataset .................................. 21  
      4.2.1 Raw data ........................................ 21
4.2.2 Data quality and limitations ............................ 22
4.2.3 Building the dataset of sequences .................. 22
4.3 Training procedure ........................................ 23
  4.3.1 Implementation details .............................. 24
4.4 Results ..................................................... 25
4.5 Evaluation ................................................... 25
  4.5.1 Performance comparison .............................. 26
  4.5.2 Generalization to very long sequences .......... 27
  4.5.3 What features do the networks learn? ........... 29

5 Conclusions .................................................. 31
  5.1 Summary .................................................. 31
  5.2 Key findings .............................................. 31
  5.3 Future work .............................................. 32
Chapter 1

Introduction

1.1 Motivation

As computer-based devices are increasingly playing a key role in people’s everyday lives there is a growing demand for technologies that allow us to interact with them in more flexible ways. To meet this demand, the devices must be better not only at providing us with information and computational power, but also at understanding what we want them to do. Several technologies have emerged in the recent decades that improve human-device interactivity: Touch screens allow us to click without a mouse, speech recognition allows us to give commands to devices without using our hands, and face recognition allows devices to differentiate between users.

Along this trend, eye tracking is a technology that enables devices to understand where we are looking while we are using them. It is already used for a wide variety of tasks today: it makes it easier movement-impaired people interact with their devices, increases interactability in videogames, gives marketers and user interface designers to gain direct feedback about what draws attention from their customers and helps researchers understand the human visual system.

Eye-tracking is a complicated problem to solve however; from simple low-resolution gray-scale images an eye tracking algorithm must be able to account for not only the eye position and orientation, but also the geometry and perspective governing how this translates to a gaze point on a screen. At the same time, the algorithm must remain robust to a variety of context-depending factors such as lighting, facial appearance and physical setup. Until recently, the go-to method for eye tracking has been to use a combination of traditional machine learning methods and geometrical models to estimate the gaze direction and intersect it with the screen to find the point the subject is looking at. To the best of our knowledge, this is how all major commercial
eye-tracking solutions are structured today.

However, much like many other areas of science and technology eye-tracking could potentially benefit a lot from the recent emergence of efficient hardware and algorithms for training and deploying convolutional neural networks (CNNs). Provided a sufficiently informative dataset a CNN can learn to account for phenomena and relationships that are traditionally very difficult to model, without the need for specialized knowledge about eyes, optics or geometry. A popular approach from the literature is to use a CNN to encode the input images as feature vectors. The vectors are then passed through a series of fully-connected layers that produces an \(x, y\)-coordinate pair representing the location of the gaze as output. Ground truth data for the training is usually generated by recording subjects as they follow a stimuli point on a computer screen with their eyes.

While the CNN-based approaches produce promising results one limitation common with all previous models is that they are making estimates based on single frames in isolation. This means that all temporal information in the input is discarded, and the network cannot make predictions based on the motion of the eyes. There is therefore a full dimension of information that is missed by the current state-of-the-art eye-tracking networks. This thesis investigates one approach to making up for this shortcoming. We argue that extending existing architectures with memory cells that are updated recurrently with each new input is a natural way of taking the temporal information into account and we show that such recurrent connections can improve eye-tracking performance in some cases. We also highlight some

Qualitatively, there are several arguments as to why including temporal information in eye-tracking networks should be beneficial: temporal features can provide a richer description of the data-stream, which in turn allows the network to achieve better performance. Training on temporal data could provide a regularizing effect on the output of the networks, as they would prefer output sequences that follow natural trajectories. Finally, the networks could learn to average out various disturbances that are present in the input images such as blinking, jittering or illumination flicker, without sacrificing responsiveness.

This thesis is especially inspired by the recent work [18], which was also performed at Tobii Stockholm. A slightly modified version of one of the networks presented in their paper will be used as a baseline for our experiments. To establish whether the addition of memory has a positive impact on the eye-tracking performance we compare the performance of the memory-based networks to the performance of the baseline on the same image sequences and loss function. We evaluate a few selected combinations of fully connected RNNs/LSTMs and
convolutional LSTMs.

### 1.2 Problem Statement

The aim of this work is to work towards a CNN architecture for eye-tracking that can take temporal information into account. We have therefore formulated the following research question:

- *can recurrent connections be used to improve the performance of CNNs in the domain of eye-tracking?*

There are of course also other methods for taking temporal information into account (such as the 3DCNN mentioned in section 2.4.1) but we have limited the scope to recurrent networks as they seemed the most promising.

### 1.3 Thesis Overview

Chapter 2 introduces the eye-tracking problem and summarizes previous approaches to solving it. It also gives a brief introduction to neural networks and how they can be extended with recurrent connections to take advantage of temporal information. Chapter 3 details our approach to adding temporal information to eye-tracking networks and introduces our main experiments along with the relevant network architectures, dataset and training procedure. The results of the experiments are presented and discussed in in Chapter 4. Chapter 5 gives some concluding remarks and suggests possible directions for future work.
Chapter 2

Background and Related Works

In this section we introduce the eye-tracking problem along with previous approaches to solving it. We then introduce some key concepts from the literature on recurrent neural networks and how they can be combined with convolutional neural networks to process video streams.

2.1 Eye-tracking

The task of an eye-tracking system is to estimate where on a screen a subject is looking based on images containing their eyes. This task is typically split into two sub-problems: eye-detection and a gaze estimation [10]. The eye detector detects the eyes and crops out a surrounding region referred to as the region of interest (ROI). The gaze estimator then processes the ROI and outputs an estimate of the gaze location.

There are two key advantages to decoupling eye-detection and gaze-estimation this way. Firstly, the cropping step ensures that the eyes are roughly centered in the ROI, which greatly reduces the number of possible configurations the gaze estimator needs to handle. Secondly, a significant amount of space can be saved since the ROIs often can be made much smaller than the original images.

In this thesis, we assume a pre-existing eye-detector and focus on the problem of gaze estimation.

2.1.1 Eye-tracking using geometrical models

The historically most common approach to eye-tracking is to use geometrical models. Typically one or several light sources, known as illuminators, are used to produce reflections, or glints, on the cornea and a camera is used to capture images of the eye. By considering geometric constraints imposed by the glints and pupil center a system of equations can be attained and solved for the gaze location. For competitive
performance a calibration procedure is often used to estimate user-dependent parameters such as eye-to-eye distance and pupil size. For further details, an introduction to the mathematics of geometry-based eye-tracking can be found in [9]. There are also methods that do not require illuminators, for instance by estimating the geometry of the eye from the pupil contour [16].

2.1.2 Eye-tracking using neural networks

There have been a few attempts at using neural networks for eye tracking in the recent years. The general approach has been to encode images containing the user’s eyes into a feature-space using a CNN as shown in Figure 2.1. The feature vectors are passed through a series of fully connected layers that finally outputs an \(x, y\)-coordinate pair representing the estimated gaze-location on the screen relative to the camera position.

Ground-truth data can be generated by having a set of subjects follow a stimuli-point on a screen with their eyes while recording their face with a camera. At each time-step \(t\) an image \(I_t\) containing the eyes of the subject is recorded along with the corresponding location of the stimuli-point \(x_t\). Assuming that the subjects are always looking at the stimuli-point during the recordings this yields a collection of image-to-gaze correspondences \(I_t \leftrightarrow x_t\) that can be used to train a neural network.

To the best of our knowledge the first attempt at using neural networks for gaze estimation was done in [14]. They introduce an open-source dataset called GazeCapture that was gathered through the crowdsourcing platform Amazon Mechanical Turk. For the gaze-estimation network they use three ROIs as input: one from each eye and a low-res version of the full face. The ROIs are combined through three parallel CNNs followed by a common set of fully connected layers to produce the \(x, y\)-output. They show that the fully trained network can be shrunk using distillation and made to work in real time on a modern smart phone while preserving performance. Another architecture is
explored in [29] where they use a single ROI containing the full face as input. Instead of cropping out ROIs containing facial features, they take a full face image as input and introduce a spatial weight layer that can explicitly filters out important regions. They evaluate their algorithms on the MPIIGaze [28] and EYEDIAP [5] datasets.

The work described in [18] was also carried out at Tobii Stockholm and they use a single ROI containing both eyes as input. They investigate the use of subject-specific calibration by tuning the final fully connected layers of the network to specialize on a single subject. They also investigate approaching gaze-estimation as a classification problem by replacing the $x, y$-coordinate output with a probability distribution over the screen plane. For training, [18] and [14] uses the L2-distance between the predicted gaze location and the ground-truth stimuli location as the loss function, while [29] uses L1-distance.

### 2.2 Deep artificial neural networks

Deep artificial neural networks are heavily parametrized models that have achieved state-of-the-art results on many previously hard-to-tackle problems. A neural network implements a function $f(x|\theta)$ by composing several differentiable and non-linear functions referred to as layers:

$$f(x|\theta) = f_L(f_{L-1}(\ldots f_0(x|\theta_0)|\theta_{L-1})|\theta_L)$$

where layer $f_i$ is parametrized by a set of parameters $\theta_i$. If there is more than one layer the network is referred to as deep. The configuration of layers together with their parametrization is referred to as the architecture of the network.

Given a dataset $S$ composed of datapoints $s$ these parameters can then trained to minimize a loss function

$$L(S) = \frac{1}{|S|} \sum_{s \in S} L(s)$$

using stochastic gradient descent. Due to the layered structure of the network the gradients can be calculated efficiently using backpropagation. By picking an appropriate configuration of architecture, dataset and loss function neural networks can be used to solve a wide variety of otherwise very difficult problems.

The following parts of this section will highlight some key concepts concepts and layer-types that are useful for understanding the rest of the thesis. For a more in-depth introduction we recommend [8].
2.2.1 Fully connected layers

The simplest layer used to build neural networks is the fully connected layer. A fully connected layer takes a vector \( x \in \mathbb{R}^n \) as input and implements an affine transformation followed by a non-linear activation function to produce a feature vector \( f \in \mathbb{R}^m \):

\[
 f = \sigma(Wx + b). \tag{2.3}
\]

Here, \( W \in \mathbb{R}^{m \times n} \) is a matrix whose elements are referred to as the weights of the layer and \( b \in \mathbb{R}^m \) is a vector referred to as the bias of the layer. The name fully connected refers to the fact that there is a single connection between every pair of input and output nodes \( x_i, f_j \) through the corresponding weight \( W_{ij} \) in \( W \). \( \sigma \) is an element-wise non-linear function referred to as the activation function. The typical choice of \( \sigma \) is the Rectified Linear Unit (ReLU) function [8]:

\[
 \text{ReLU}(x) = \max\{x, 0\} \tag{2.4}
\]

as it promotes sparse gradients. But other activation functions such as \( \tanh(x) \) or the sigmoid function:

\[
 \text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{2.5}
\]

are sometimes used instead.

2.2.2 Convolutional layers

When applying neural networks to images it is very computationally inefficient to learn an independent connection for every single pixel in the input image. To circumvent this issue it is common to use convolutional layers that takes a set of feature maps \( X \in \mathbb{R}^{w \times h \times C} \) with width \( w \), height \( h \) and channels \( C \) as input, note that an image is simply a feature map with either 3 or 1 channel (corresponding to RGB or grayscale). The feature maps are passed through a set of convolution kernels \( \mathcal{W} \in \mathbb{R}^{n \times n \times D \times C} \), also referred to as filters, to produce an output feature map with \( D \) channels:

\[
 Y = \sigma(\mathcal{W} \ast X + B) \in \mathbb{R}^{h \times w \times D} \tag{2.6}
\]

where \( B \in \mathbb{R}^D \) represents a constant bias term for each channel that is applied pixel-wise. Pixel \( i, j \) of channel \( d \) in the output is more explicitly given by:

\[
 Y_{ij}^d = \sigma(\sum_{c \in C} \sum_{k,l=-n}^{n} W_{kl}^{dc} X_{i+k,j+l}^c + B_d) \tag{2.7}
\]
When performing the computations in equation 2.7 the input $X$ is padded with zeros for the terms where the index is out of bounds to ensure that the width and height remains unchanged.

An important property of convolutional layers is that each element of a channel in the output only depends on a small surrounding region in the channels of the input, called the receptive field. The size of the receptive field is determined by $n$ and it grows linearly with depth if several convolutional layers are stacked. The result is that shallow filters can learn to detect very localized features such as edges, while deeper features can learn to detect more complex structures such as eyes or pupils as the receptive field expands.

### 2.2.3 Pooling layers

For a convolutional neural network to learn useful representations of an image it is important to have a large number of filters, which in turn requires a large amount of memory in order to store the feature-map channels. To mitigate this issue it is common practice to downsize the filter maps after every few convolution layers. This downsizing is typically done using a pooling layer that reduces each $p \times p$ grid in the input to a single number, typically by taking the maximum value in the grid (max pooling) or the mean (average pooling). Downsizing can also be achieved without pooling by, for example, adding a stride to the convolution layers.

### 2.3 Recurrent neural networks

A recurrent neural network (RNN) is a neural network with feedback connections that allow the network to store information between inputs. In practice, the feedback is typically achieved by implementing RNN cells that each store a state vector $h_t \in \mathbb{R}^k$ that gets updated with each new input $x_t \in \mathbb{R}^d$ from the data sequence. The update rule for the standard RNN cell is

$$h_{t+1} = \sigma(Ux_t + Vh_t + b)$$

(2.8)

where $U \in \mathbb{R}^{k \times d}$ and $V \in \mathbb{R}^{k \times k}$ are matrices and $\sigma$ is an element-wise activation function. The initial state $h_0$ is typically set to zero.

This way, the output of the layer at time $t$ depends not only on the input at time $t$ but also indirectly on all past inputs through the state vector. What information is kept from the input and what is kept from the state is determined by the matrices $U$ and $V$ respectively. Several
RNN cells can be stacked together in order to allow the network to learn more complex temporal structures.

However, it has been noted that the simple update rule in equation 2.8 has problems with poorly behaved gradients in practice [20] and several modifications have been proposed to alleviate this issue. The most common approach is to replace the standard cell with a Long-Short-Term Memory (LSTM) cell [11]:

\[
\begin{align*}
    i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci} \odot c_{t-1} + b_i) \\
    f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf} \odot c_{t-1} + b_f) \\
    o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co} \odot c_{t-1} + b_o) \\
    c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
    h_t &= o_t \odot \tanh(c_t)
\end{align*}
\] (2.9)

where \(i_t, f_t, o_t\) and \(c_t\) are referred to as the input gate, forget gate, output gate and hidden state at time \(t\), respectively. "\(\odot\)" refers to element wise multiplication between vectors. The terms involving \(W_{ci}, W_{cf}\) and \(W_{co}\) are referred to as peephole connections [6], and they are a common addition to the original model that allow the gates direct access to the hidden state. The main advantage of the LSTM is that its hidden state acts as an internal memory. Reading and writing to memory is made explicit by the output and input gates, and the deletion of memory is made explicit through the forget gate. This process would have to be learned from scratch by a regular RNN cell.

LSTMs are heavily used in the field of natural language processing. Texts can be naturally described as a sequence of letters or words if an appropriate coding method is chosen. A landmark example is the Neural Machine Translation System [26] from Google.

### 2.4 Sequence-based CNNs

Research in human perception indicates that short-term temporal features play an important role in our vision [7] so it is reasonable to expect that CNNs can benefit from them as well.

To allow a CNN to learn such features, its architecture must be extended to in a way that allows it to encode temporal information. However, there are many ways of adding memory to a CNN and there is no clear consensus in the video processing community as to which approach is the most fitting. To survey the area, and to introduce the layers that we will use for eye-tracking, this section introduces what we deem to be the three most popular approaches: 3D CNNs, feature layer RNNs and convolutional RNNs.
2.4.1 3D CNNs

One way to add memory and allow a neural network to learn temporal features is through 3D CNNs. A 3D CNN layer is characterized by the fact that it convolves in the time dimension in addition to the two spatial dimensions. To achieve this, the layer takes a stack of \( K \) successive video frames as input and uses 3 dimensional convolution kernels of size \( n \times n \times k \) where \( k < K \) is the time-extent of the kernel, as visualized in Figure 2.2.

Many variants of the 3D CNN have been proposed in the recent years and [2] provides a good introduction to the area. They employ two parallel 3D CNNs (one for RGB frames and one for optical flow frames) in order to achieve state of the art results in video action classification. A more light-weight architecture named pseudo-3D CNNs is introduced in [22] and shown to achieve state of the art results in video action classification. The pseudo-3D CNN can also be conveniently initialized using the weights from a standard 2D CNN.

The main limitation of the 3D CNN architecture is that it only allows the network to recall events from a maximum of \( K \) time steps back. They are also ill-suited on-line or memory limited tasks since a cache of \( K \) frames must be stored in memory at all time.

2.4.2 Feature layer RNNs

Another approach to adding memory is to add RNN/LSTM cells to the feature layers of the convolutional network. This allows the network to process data online without a cache but the temporal information is processed much later in the network, which potentially makes it more
Difficult for it to pick up on subtle changes between frames.

Feature-layer RNNs were used in [23] for unsupervised learning of video representations, they added an LSTM to the bottleneck of an autoencoder. A similar setup is used in [2] for action prediction from videos, they demonstrate that using LSTMs on the feature layers of a CNN outperforms a single-stream 3D CNN by a large margin but two-stream 3D CNNs attains the best performance among the compared models.

2.4.3 Convolutional RNNs

A more fitting approach to real time video processing is to incorporate recurrence directly in the convolutional layers with a so called convolutional RNN (ConvRNN) cell.

A ConvRNN cell is a modified version of the regular RNN cell in equation 2.8 for the case when the input is a sequence of feature maps $X_t \in \mathbb{R}^{h \times w \times C}$, where $h \times w$ are the height and width and $C$ is the number of input channels. The main idea of the ConvRNN is to replace the state vector $h_t$ with a $D$-channel state map $H_t \in \mathbb{R}^{h \times w \times D}$. Spatial correlations in the frames can then be exploited by replacing the matrix multiplications with convolutions against filter kernels $U \in \mathbb{R}^{n \times n \times D \times C}$ and $V \in \mathbb{R}^{n \times n \times D \times D}$. The update rule for a ConvRNN cell would then be

$$H_{t+1} = \sigma(U \ast X_t + V \ast H_t + B)$$

where $\ast$ denotes convolution as in equation 2.7. This way each region of $H_t$ encodes local spatio-temporal features that can detect for
instance simple motions at shallow layers, and larger moving objects or gestures in deeper layers. A comparison between a ConvRNN cell and a regular RNN cell is shown in Figure 2.3.

We note that the convRNN cell is more sensitive to the choice of initial state map $H_0$ when compared to the choice of initial state vector in a fully connected RNN. This is because the convolutions cannot compensate for spatial variations outside their receptive field, so the choice of spatial structure in the initial state is significant. To alleviate this issue, we let $H_0$ be a trainable parameter initialized to 0 in order to let the network learn a suitable initialization. The learned initial state map can then also serve as a prior to the network by storing information that is common to all sequences.

Similar modifications can also be made to the LSTM cell in equation 2.9 to get a ConvLSTM cell:

\[
\begin{align*}
I_t &= \sigma(\mathcal{V}_{xi} \ast X_t + \mathcal{V}_{hi} \ast H_{t-1} + \mathcal{V}_{ci} \circ C_{t-1} + B_i) \\
F_t &= \sigma(\mathcal{V}_{xf} \ast X_t + \mathcal{V}_{hf} \ast H_{t-1} + \mathcal{V}_{cf} \circ C_{t-1} + B_f) \\
O_t &= \sigma(\mathcal{V}_{xo} \ast X_t + \mathcal{V}_{ho} \ast H_{t-1} + \mathcal{V}_{co} \circ C_{t-1} + B_o) \\
C_t &= F_t \circ C_{t-1} + I_t \circ \tanh(\mathcal{V}_{xc} \ast X_t + \mathcal{V}_{hc} \ast H_{t-1} + B_c) \\
H_t &= O_t \circ \tanh(C_t)
\end{align*}
\]

where $I_t$, $F_t$, $H_t$, $O_t$ and $C_t$ are referred to as the input gate map, forget gate map, hidden state map and output gate map at time $t$ respectively. "o" refers to pixel-wise multiplication in the width, height and channel dimensions. Note that the gates this way not only dictate what information to retrieve, but also where in the image it should be retrieved from.

The convRNN structure was first introduced in [27] where they used a convLSTM for video prediction on weather data and on the moving MNIST dataset. They employed an encoder-decoder model with convLSTMs in both parts and show that it significantly outperforms feature-layer LSTM in their case. Since the original paper, convLSTMs have also been applied to a variety of other video related problems such as the classification of first person interaction [24], anomaly detection in videos [19], video autoencoding [21] and predicting physical interactions between objects and a robotic arm [4, 17]. All these problems would be very difficult to solve with single-frame methods, so giving the networks a tool to efficiently encode temporal information is essential.

Regular convolutional RNNs (without any gates or hidden states) have been applied to video super resolution in [12], where they use a model similar to [27]. NVIDIA also recently used them for video reconstruction in Monte Carlo rendering [3]. They used an encoder-
decoder structure with convRNNs in the encoder layers to enforce temporal consistency between the reconstructed frames.

2.5 Ethical and societal Considerations

2.5.1 Model bias

One ethical concern when building data-driven models that are meant to interact with humans is population biases in the dataset. The issue of biased datasets is similar to that of unbalanced classes [15] in that it is always preferable for a model to trade an increase in performance across a majority group for an equal decrease in performance across a minority group. It is therefore important to consider the statistics of the dataset set when building a data-driven model intended for real-world deployment.

These issues are especially prevalent in eye-tracking since the models have to rely directly on facial features, which are affected by various factors such as ethnicity, age, gender, glasses and facial scars. While it might not be realistic to achieve perfectly equal performance across the entire population there are steps that can be taken to control the discrepancies:

- Gather more data from underrepresented groups
- Sub-sample the data gathered from overrepresented groups
- Add importance weights during training to prioritize underrepresented groups

When taking such measures, one must also decide on what performance to consider as fair; should the model be allowed to prioritize performance according to population statistics or target audience for example? There are also costs to removing biases from a model. Balancing the performance can require significant trial-and-error development time and overengineering a model to remove certain biases could introduce unintended consequences. It might for instance be inherently more difficult to track the eyes of people with glasses, in which case it could be impossible to achieve equal performance without artificially degrading the performance for users without glasses.

Since the networks trained for this thesis are not intended for consumer use we have not taken any concrete measures to track or mitigate this kind of model bias. While we do not expect the relative impact of memory cells to vary significantly across the population we acknowledge that the absolute performance could vary significantly between different people.
2.5.2 Privacy

Privacy issues in databases have gained a great deal of attention recently due to the introduction of the GDPR in the EU [25]. Since eye-tracking models rely on pictures of faces these regulations are of course highly relevant. Care should be taken that all data is stored safely and that anything that is personally identifiable can be erased at the request of the individual.
Chapter 3

Methods

Similar to previous approaches [18, 14, 29] we treat eye-tracking as an end-to-end CNN-based regression problem. To answer the research question posed in section 1.2 we will extend on the previous work by adding recurrent connections in order to exploit the temporal information in the video stream from the eye-tracking camera. This requires us to update the previous single-frame models to temporal models that can process multiple consecutive frames as input. Doing this naturally introduces an extra dimension of information that the network can use to make better predictions, but it also means that we have to introduce additional complexity and structure to the dataset and training pipeline. To summarize, we believe that there are three main components that are required to obtain a temporal model:

1. A network architecture with memory that allows the network to encode temporal information.
2. A sequence-based loss that appropriately reflects how information in the input sequence should be prioritized.
3. A sequence-based dataset that exposes the network to realistic sequences and reflects the use case of the network.

In this chapter, we detail our approach to each of these components in the case of eye-tracking, and introduce our main experiments.

3.1 Networks architectures

The task of a CNN-based regression network $f(I|\theta)$ with parameters $\theta$ is to take in a sequence of images $I_0, \ldots, I_{T-1} \in \mathbb{R}^{h \times w \times C}$ containing task-relevant information and produce a sequence of estimates
\hat{x}_0, \ldots, \hat{x}_{T-1} \in \mathbb{R}^d \text{ that are as close as possible to the corresponding ground truth output sequence } x_0, \ldots, x_{T-1} \in \mathbb{R}^d.

A static single-frame network would solve this problem by taking an image \( \mathcal{I}_t \) as input at each time-step \( t \) and produce a corresponding gaze-estimate \( \hat{x}_t = f(\mathcal{I}_t|\theta) \) that depends only on \( \mathcal{I}_t \). In our case we want to make use of temporal information, meaning that we have

\[
\hat{x}_t = f(\mathcal{I}_0, \ldots, \mathcal{I}_t|\theta). \tag{3.1}
\]

A network with this property is said to have memory. We particularly add memory layer-wise by using recurrence as explained in sections 2.4.2 and 2.4.3. Thus, the output of the network depends implicitly on previous inputs through the states \( h_t = h_t(\mathcal{I}_t, h_{t-1}) \):

\[
\hat{x}_t = f(\mathcal{I}_t, h_{t-1}|\theta). \tag{3.2}
\]

In the remainder of this section we present the particular architectures that we have used to tackle the eye-tracking problem where we a two-dimensional output. The models can be used for any video-based regression problem by modifying the number of output neurons \( d \).

### 3.1.1 Baseline architecture

A network based on the architecture in [18] evaluated on single frames is used as a baseline for all our experiments. The full architecture is visualized in Figure 3.1 and details are listed in the first row of Tables 3.1 and 3.2. The network is composed of two main components: the backbone CNN network and the fully connected feature layers. We use layer normalization [1] for all layers and during training we use 50% dropout [8] in the early feature layers.

The binary face grid that was used in [18] is omitted in all our networks (including the baseline) in order to simplify the architectures. This is not expected to have a significant impact on the network’s ability to learn since there is very little variance in the location of the subjects in the field of view of the camera over the dataset.
### Table 3.1: Comparison between the convolutional layers in the baseline network and the modified convLSTM variant.

<table>
<thead>
<tr>
<th>Layer</th>
<th>input</th>
<th>conv1</th>
<th>conv2</th>
<th>conv3</th>
<th>conv4</th>
<th>conv5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>-</td>
<td>5x5 conv 4x4 max pool</td>
<td>(3x3 conv)x2 2x2 max pool layer norm</td>
<td>(3x3 conv)x4 2x2 max pool layer norm</td>
<td>(3x3 conv)x4 2x2 max pool layer norm</td>
<td></td>
</tr>
<tr>
<td>ConvLSTM</td>
<td>-</td>
<td>5x5 conv 4x4 max pool</td>
<td>3x3 convLSTM 2x2 max pool layer norm</td>
<td>(3x3 convLSTM)x2 2x2 max pool layer norm</td>
<td>(3x3 conv)x4 2x2 max pool layer norm</td>
<td></td>
</tr>
<tr>
<td>output shape</td>
<td>75x290x1</td>
<td>17x71x64</td>
<td>8x35x128</td>
<td>4x17x256</td>
<td>2x8x512</td>
<td>1x4x512</td>
</tr>
</tbody>
</table>

### Table 3.2: Comparison between the fully connected layers in the baseline network and the modified RNN and LSTM variants. FC indicates a fully connected layer.

<table>
<thead>
<tr>
<th>Layer</th>
<th>input</th>
<th>FC1</th>
<th>FC2</th>
<th>FC3</th>
<th>FC4</th>
<th>FC5</th>
<th>FC6</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>FC (Baseline)</td>
<td>-</td>
<td>FC Layer norm 50% dropout</td>
<td>FC Layer norm 50% dropout</td>
<td>FC Layer norm</td>
<td>FC Layer norm</td>
<td>FC Layer norm</td>
<td>FC Layer norm</td>
<td>FC</td>
</tr>
<tr>
<td>RNN(LSTM)</td>
<td>-</td>
<td>FC Layer norm 50% dropout</td>
<td>FC Layer norm 50% dropout</td>
<td>FC Layer norm RNN(LSTM) Layer norm</td>
<td>FC RNN(LSTM) Layer norm</td>
<td>FC RNN(LSTM) Layer norm</td>
<td>FC Layer norm</td>
<td></td>
</tr>
<tr>
<td>Number of nodes</td>
<td>2048</td>
<td>4096</td>
<td>2048</td>
<td>1024</td>
<td>512</td>
<td>256</td>
<td>128</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 3.1: Comparison between the convolutional layers in the baseline network and the modified convLSTM variant.

Table 3.2: Comparison between the fully connected layers in the baseline network and the modified RNN and LSTM variants. FC indicates a fully connected layer.
3.1.2 Recurrence-based architectures

We build a sequence-based network by adding recurrence to a few selected layers in the baseline network. There are three main classes of network we can attain by doing this: recurrence in the convolutional layers, recurrence in the fully connected layers and recurrence in both components in the network. We list the specific architectures that we have used for each component in the bottom rows of tables 3.1 and 3.2.

3.2 Dataset structure

We structure our dataset $S$ such that each datapoint is a length-$T$ sequence of image-to-vector correspondences:

$$ S = \left\{ (T_i^t, x_i^t), \ldots, (T_{i-1}^t, x_{i-1}^t) \mid i \in 0, \ldots, N - 1 \right\}. \hspace{1cm} (3.3) $$

Where $T_i^t \in \mathbb{R}^{h \times w \times C}$ is frame $t$ in sequence $i$ and $x_i^t \in \mathbb{R}^d$ is the corresponding desired output of the neural network.
3.3 Loss function and performance measures

We train the networks by minimizing the $L^2$-distance between the estimates $\hat{x}_t^i$ and the ground truth $x_t^i$, averaged over the training set. The first $k$ frames in each sequence are ignored when calculating the loss in order to reduce the impact of state-initialization. Letting $d_t^i = \hat{x}_t^i - x_t^i$ be the estimation residuals we explicitly have that the loss on our dataset is

$$\text{loss} = \frac{1}{N} \sum_{i=0}^{N-1} \frac{1}{T-k} \sum_{t=k}^{T-1} \|d_t^i\|_2.$$  \hspace{1cm} (3.4)

where $N$ is the number of sequences in the dataset. Note that we have

$$\hat{x}_t^i = \hat{x}_t^i(I_0, \ldots, I_t)$$ \hspace{1cm} (3.5)

so since errors are averaged across the length of each sequence the network is forced to make predictions both based on a small amount of frames and on longer sequences (up to length $T$). For evaluating the networks we also define the following performance metrics:

$$\text{precision} = \frac{1}{N} \sum_{i=0}^{N-1} \frac{1}{T-k} \sum_{t=k}^{T-1} \|d_t^i - \bar{d}_t^i\|_2$$

$$\text{accuracy} = \frac{1}{N} \sum_{i=0}^{N-1} \|\bar{d}_t^i\|_2.$$ \hspace{1cm} (3.6)

where

$$\bar{d}_t^i = \frac{1}{T-k} \sum_{t=k}^{T-1} d_t^i$$ \hspace{1cm} (3.7)

is the average residual for sequence $i$. This way, the precision is the variance of the estimates and the accuracy is the average distance between the mean estimate and the ground truth.

To investigate how the addition of memory impacts performance over time we also define a frame-wise loss:

$$\text{loss}(t) = \frac{1}{N} \sum_{i=0}^{N-1} \|d_t^i\|_2.$$ \hspace{1cm} (3.8)

This way, loss$(t)$ is the average loss over the test set when the network has seen exactly $t$ frames. A network that can make effective use of its memory should have a loss$(t)$-curve that decreases with $t$, as that indicates that it makes better predictions the more information it has access to.
Chapter 4

Experiments

In this chapter we describe the experiments that were performed in order to evaluate the effectiveness of recurrent neural networks for eye-tracking. We begin by detailing the specific network architectures, and then report and evaluate the performance of each of the network architectures presented in section 3.1.2.

4.1 Experimental setup

In order to test the effectiveness of recurrent connections in eye-tracking, we modify the baseline architecture with a few selected combinations of RNN-based components described in section 2.4 using the architectures introduced in section 3.1. We refer to each network with a two-component name: one for how we modify the backbone-CNN and one for how we modify the feature layers. Using this convention, the baseline network with no modifications will be referred to as CNN-FC, where FC stands for "Fully Connected". We perform four experiments:

- CNN-RNN (baseline): with regular convolutions and RNN recurrence in the feature layers
- convLSTM-FC: with LSTM recurrence in the convolutional layers and regular feature layers
- convLSTM-RNN: with LSTM recurrence in the convolutional layers and RNN-recurrence in the feature layers
- convLSTM-LSTM: with LSTM recurrence both the convolutional layers and the feature layers

The experiments are summarized in Table 4.1 and the architectures detailed in the second row of Tables 3.1 and 3.2. The baseline architecture is used for the unmodified part of the networks where appropriate.
<table>
<thead>
<tr>
<th>Network name</th>
<th>CNN layers</th>
<th>FC layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN-FC (baseline)</td>
<td>5xConv</td>
<td>6xFc</td>
</tr>
<tr>
<td>CNN-RNN</td>
<td>5xConv</td>
<td>3xFc, 3xRNN</td>
</tr>
<tr>
<td>ConvLSTM-FC</td>
<td>2xConvLSTM, 3xConv</td>
<td>6xFc</td>
</tr>
<tr>
<td>ConvLSTM-RNN</td>
<td>2xConvLSTM, 3xConv</td>
<td>3xFc, 3xRNN</td>
</tr>
<tr>
<td>ConvLSTM-LSTM</td>
<td>2xConvLSTM, 3xConv</td>
<td>3xFc, 3xLSTM</td>
</tr>
</tbody>
</table>

Table 4.1: Summary of network architectures. RNN refers to the standard RNN cell.

### 4.2 Eye-tracking dataset

To use the methods described in the previous chapter for eye-tracking we need a dataset of ground truth image-to-gaze correspondences. Since we want to evaluate sequence-based models we structure our dataset according to equation 3.3. In our case $x_i^t$ is the location of the stimuli point that the subject was told to look at at time $t$ in sequence $i$ and $I_i^t$ is an image of the eyes of the subject taken at the same time.

#### 4.2.1 Raw data

We build our dataset of sequences from a collection of raw eye-tracking data that has been provided by Tobii. We have access to 789 recordings, each with a unique participant. During each recording the corresponding participant was asked to follow a stimuli point on a 16” computer screen with their eyes. The stimuli points move between roughly 12 random fixed locations and remains at each of them for a few seconds. The locations are picked randomly for each participant with a uniform probability over the computer screen. When transitioning between two fixed locations the stimuli-point moves along a straight line at a constant speed. The transition time between fixed locations is maintained constant meaning that the speed at which the stimuli-point moves varies between transitions and is determined by the distance it has to travel.

The recordings are gathered at 30 frames per second with a commercial eye tracking camera positioned at the bottom of the screen. In each frame the eyes of the subject are detected using a machine learning algorithm and a rectangular ROI containing both eyes is extracted, two sample images are shown in Figure 4.1. The participants were told to sit as still as possible at a fixed distance from the screen and to position their heads roughly in a fixed region within the field of view of the camera. The images are gray-scale and we downscale them to have a height of 75 pixels and a width of 290 pixels, meaning that we
have $\mathcal{I}_t \in \mathbb{R}^{75 \times 290 \times 1}$ for all $i$ and $t$.

### 4.2.2 Data quality and limitations

As always when building a framework for training a neural network, it is important to avoid situations where the structure or statistics of the dataset misrepresents the use-case of the network. In our case we want the network to learn to accurately track the gaze of any user, even if they are not in the training set.

The raw recordings contain both periods of fixation, when the stimuli points are stationary, and from periods of linear smooth pursuit as the stimuli point moves between locations. We could therefore expect a network trained on this dataset to perform well when the user following a slowly moving object with their eyes. However, the dataset does not contain any saccades so we should expect a degradation of performance when the user shifting their gaze quickly, such as when searching for a hidden object.

Another limitation of the dataset is that the participants were told to sit still during the recordings. In a realistic setting, users will occasionally move their head while using the computer and they will not maintain a fixed distance from the screen. We should therefore expect the performance of our networks to degrade if the user moves around while using it or if they sit too close of far away from the camera. This kind of issues can be mitigated by having more relaxed conditions for the participants at the data collection stage, for instance by having them sit naturally at their office desk.

### 4.2.3 Building the dataset of sequences

Each full recording contains a lot of redundancy, since successive frames tend to be very similar. While redundancy in the final dataset is not necessarily detrimental to the training process, larger training sets naturally leads to longer training times and heavier requirements on storage capacity. It is therefore necessary to sub-sample the raw full recordings to produce a more manageable dataset for training, validation and testing.

In the previous works with eye-tracking at Tobii [18] they solved this problem by picking out a small number of frames from each stationary stimuli point location in each recording. Similarly, we create our dataset of sequences by randomly selecting 30 (possibly overlapping) sequences of 20 frames from each recording. The result is roughly 23000 sequences. Among the resulting sequences, about 53% contain some frames where the stimuli point is moving. Furthermore,
among the sequences with movement 15% are entirely moving, 42% are transitions from movement to stationarity and 43% are transitions from stationarity to movement.

The full dataset is divided into a training set, a validation set and a test set with a 80/10/10 split. The split is done by participant to ensure that test and validation is performed on people that the network did not see during training.

4.3 Training procedure

For each epoch of training we split the training set into batches and feed the batches through the network in order to compute gradients due to the loss in equation 3.4. This section highlights some measures that we found useful when making the sequence-based training procedure work.

Reducing the sequence length for training

One of the most significant problems that come with working with models that have video sequences as input is that the memory requirements are much larger than the single-frame counterpart. This is because the storage requirement goes from 1 image per datapoint to $T$ (in our case 20) images per datapoint. In order to keep the batches within the memory limitation of our GPU without compromising the number of datapoints per batch we had to restrict the training to sequences of a shorter length $T_{\text{train}} < 20$. This is achieved using the following procedure: each time a sequence is selected for a batch at training time we instead extract a random subsequence of $T_{\text{train}}$ consecutive frames from the full sequence. The subsequence is then fed to the network in place of the full sequence. The choice of subsequence from a given sequence is allowed to vary between epochs. We have used $T_{\text{train}} = 6$ for all sequence-based models in this thesis, meaning that the longest sequences the networks see during training are 6 frames long. The full-length sequences are however still used for testing and validation.
Controlling for architectural changes

Replacing a regular convolution layer with a convLSTM-cell fundamentally changes the way input/output of the layer is processed due to the gated structure, even in the case of single-frame input. To control for this factor we train two variants of each sequence-based network:

1. A sequence-based variant that is trained on sequences of length up to $T_{\text{train}}$ with $k = 2$

2. A single-frame variant that is trained on single frames without any temporal context

This way, if a network type performs equally well (or better) when trained without memory we can suspect that the memory cells were ineffective (or even confused the network). Note that the network architectures are the same in both cases; the RNN/LSTM connections are still there in the variant without memory but since there is no temporal information in the input during training the network cannot learn to use them to store information.

Pre-training on single frames

During early experiments we noticed that if the networks with memory cells are trained on long sequences from the start they often converge to bad local optima and tend to generalize poorly. To circumvent this issue, all networks with memory are pre-trained on single frames (corresponding to $T = 1$ and $k = 0$) with a larger batch size until the validation loss begins to plateau. $T$ is then steadily increased from 3 to $T_{\text{train}}$ with $k = 2$ by incrementing $T$ by 1 every 5 epochs. This way, the networks are first forced to learn to make good single-frame predictions before they get access to multiple frames.

4.3.1 Implementation details

The experiments were carried out at Tobii Stockholm. Training and evaluation has been performed on a Nvidia GTX1080 GPU and all networks have been implemented using Python and Tensorflow v1.4. We use a batch-size of 64 frames for training the single-frame models and when pre-training the sequence-based networks, and a batch size of 25 sequences for training the sequence-based networks. In all cases, training is allowed to continue until the validation loss stops decreasing and the model with the lowest validation loss is selected as the final version. Due to the reasons noted in Section 2.4.3, we let the initial states of all convLSTM cells be trainable parameters in order to let the
CHAPTER 4. EXPERIMENTS

Table 4.2: Results from the comparative network experiments. For each model we list the performance of both the sequence-based and the single-frame variant.

<table>
<thead>
<tr>
<th>network</th>
<th>trained on</th>
<th>loss</th>
<th>accuracy</th>
<th>precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN-FC (baseline)</td>
<td>sequences</td>
<td>45.81</td>
<td>42.98</td>
<td>14.10</td>
</tr>
<tr>
<td></td>
<td>single frames</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNN-RNN</td>
<td>sequences</td>
<td>47.95</td>
<td>44.40</td>
<td>11.12</td>
</tr>
<tr>
<td></td>
<td>single frames</td>
<td>44.93</td>
<td>41.98</td>
<td>14.31</td>
</tr>
<tr>
<td>ConvLSTM-FC</td>
<td>sequences</td>
<td>43.41</td>
<td>41.17</td>
<td>12.43</td>
</tr>
<tr>
<td></td>
<td>single frames</td>
<td>45.61</td>
<td>42.49</td>
<td>14.76</td>
</tr>
<tr>
<td>ConvLSTM-RNN</td>
<td>sequences</td>
<td>42.97</td>
<td>40.63</td>
<td>12.42</td>
</tr>
<tr>
<td></td>
<td>single frames</td>
<td>45.44</td>
<td>42.67</td>
<td>13.87</td>
</tr>
<tr>
<td>ConvLSTM-LSTM</td>
<td>sequences</td>
<td>47.33</td>
<td>45.17</td>
<td>11.80</td>
</tr>
<tr>
<td></td>
<td>single frames</td>
<td>49.23</td>
<td>46.21</td>
<td>14.78</td>
</tr>
</tbody>
</table>

network learn its own state initialization. The loss function was minimized using the Adam algorithm [13] with learning rate $\alpha = 0.0001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$.

4.4 Results

The loss, accuracy and precision, according to equations 3.4 and 3.6, for all the networks is shown in Table 4.2. For each network we show the performance for both the sequence-based and the single-frame variant. We also show the loss($t$)-plot, according to equation 3.8, in Figure 4.2.

For Table 4.2, we have used $T = T_{\text{test}} = 20$. So in order to have a good score in each of the categories the networks must perform well for sequences that are about three times longer than the longest sequences seen during training (recall that we used $T_{\text{train}} = 6$). We also use $k = 2$ to ignore the first two frames when calculating the performance as was done during training. The baseline network and the memoryless variants of the RNN-based architectures were evaluated on single frame inputs, corresponding to $T = 1$ and $k = 0$.

4.5 Evaluation

This section presents our interpretations of the results presented in the previous section. In addition to the loss we primarily consider the following two indicators of performance:
Figure 4.2: Performance comparison per timestep. Only the convLSTM-RNN model is able to effectively use its memory to improve performance over time. The dashed lines indicate the time-interval seen during training.

1. ability to generalize beyond the training time cut-off at $t = 6$

2. ability to exploit temporal information and reduce loss over time

The first criteria is necessary for the network to work in practice; eye-tracking systems are typically used over extended periods of time (several minutes at least). The second criteria is necessary to gage the usefulness of the memory itself; if the loss does not decrease with time there is no use in feeding the network sequences in the first place.

### 4.5.1 Performance comparison

The first observation we can make from the averaged performance measures in Table 4.2 is that the ConvLSTM-RNN and ConvLSTM-FC architectures with memory both outperform the baseline by in terms of average loss, accuracy and precision over the 20 frame test sequences. The networks manage to reduce the baseline loss by 6.2% and 5.2% respectively, meaning that the ConvLSTM-RNN slightly outperforms the ConvLSTM-FC.

To gain some insight into the performance difference between the ConvLSTM-FC and the ConvLSTM-RNN we consider the loss($t$)-plot in Figure 4.2. The loss of the ConvLSTM-FC is stable over time, meaning that it is able to generalize to longer sequences without losing performance but it cannot exploit the additional temporal information to further decrease the loss over time. The ConvLSTM-RNN however
has a loss that is decreasing over the training interval between $t = 2$ and $t = 6$. This means that it is able to both generalize and exploit the temporal information in the sequences. However, the exploitation of information does not generalize: the loss of the ConvLSTM-RNN does not continue to decrease after the training cut-off.

When trained without memory on single frames both these networks have shown a slight performance improvement over the baseline (0.8% and 0.4% respectively). This indicates that there is a slight architectural advantage to using the input/output gate structure of the ConvLSTM cell but training on sequences is necessary to gain a significant increase in performance.

The CNN-RNN and the ConvLSTM-LSTM do not outperform the baseline but we can still draw some potentially useful information from their results. Referring again to Figure 4.2, we note that both these models have an increasing loss after the training time cut-off. This indicates that they are both unable to generalize and unable to exploit the temporal information of the sequences. For the CNN-RNN, one possible interpretation is that the regular CNN backbone is not able to provide a sufficiently rich description of the input sequence to the feature layers, leading to the RNN layers being ineffective. For the ConvLSTM-LSTM, the addition of LSTM-recurrence in the feature layers instead of RNN recurrence seems to confuse the network, this might be caused by the sequences used during training being too short, meaning that the more powerful LSTM model can overfit.

### 4.5.2 Generalization to very long sequences

From the evaluation based on the test set in the previous section, we know that the ConvLSTM-FC and the ConvLSTM-RNN networks are able to generalize to sequences roughly three times as long as what they were shown during training. However, it is important to note that this does not guarantee that performance is maintained over very long sequences. During a 20-frame sequence, the gaze-type only has time to switch at most once (from moving to stationary or from stationary to moving) and there is no guarantee that the network will be able to handle more complex behaviours. In longer sequences, the gaze could for example go from being stationary to moving and then back to stationary and so on. To investigate how our networks behave when exposed to these long-term phenomena we evaluate the sequence-based and the single-frame versions of the best performing model (ConvLSTM-RNN) on 550 consecutive frames.

A gaze-plot for the two variants is shown in Figure 4.3. Qualitatively, the gaze estimates remain stable over time and do not di-
Figure 4.3: Gaze plot of the ConvLSTM-RNN network evaluated on 550 consecutive frames with and without memory. The variant with memory does not diverge but overall the memory-less variant has a lower loss.
CHAPTER 4. EXPERIMENTS

Figure 4.4: Sample channels from the hidden state map from the first convLSTM cell in the convLSTM-RNN network, $C^2_6$.

Figure 4.5: Sample of learned initial states $H_0$ from the first convLSTM cell in the fully trained ConvLSTM-FC network.

verge. However, the sequence-based ConvLSTM-RNN achieves an average loss of 24.62 while the single-frame ConvLSTM-RNN achieves an average loss of 20.92. This indicates that the information stored in the memory cells are not able remain useful over the very long sequence, so performance degrades. It would therefore be necessary to find a way to preserve the performance of our models over longer time-periods before they could be used in practice.

4.5.3 What features do the networks learn?

Figure 4.4 shows a sample of 64 channels from the state maps $H_t$ and hidden state maps $C_t$ generated by the first convLSTM cell in the convLSTM-RNN network after it has seen six images of a subject looking at a moving stimulus. As is to be expected many channels trigger almost exclusively on the eyes of the subject. The learned initial states from the first
ConvLSTM-cell in the ConvLSTM-FC network is shown in Figure 4.5. Note that some facial features are visible in the states, so the network seems to adapt the initial state to match its typical input states.
Chapter 5

Conclusions

5.1 Summary

We have shown how a standard regressive CNN can be extended with memory in order to allow it to exploit temporal information in its input. Five RNN-based approaches to adding memory to a CNN have been introduced and evaluated on an eye-tracking dataset provided by Tobii. We trained our networks on sequences of length 6 where the first two frames are ignored when computing the loss, and evaluation was done on sequences of length 20 from subjects not seen during training. For comparison, we have also trained and evaluated all our networks on single frames.

By moving from having single frames as input to using sequences as input we give the networks access to additional information but we also unavoidably introduce a number of new problems that have to be tackled. We have therefore highlighted what we believe to be the key aspects to making this transition work and how we approach them in the case of eye-tracking. In addition, we have also described some key tricks that we found useful to making the training process work.

5.2 Key findings

We have found that the best performing approach to adding memory to a CNN for eye tracking under our conditions is to have LSTM recurrence in the convolutional layers and standard RNN recurrence in the feature layers. This was also the only architecture that was able to use its memory to decrease loss over time during the training interval and remain stable after the training-time cutoff.

If the recurrence is removed from the feature layers of the network it still outperforms the baseline but seems to lose its ability to improve
performance over time. On the other hand, if the convolutional recurrence is removed the network is unable to outperform the baseline and performance degrades over time outside the training-time interval. This indicates that convolutional recurrence on its own is insufficient to make use of the temporal information, but they can provide a richer feature vector to the fully connected layers that the RNN structure can exploit. In summary our results suggest that it is necessary to have recurrence both in the convolutional layers and in the feature layers for the approach to work.

We have also observed that not all networks with memory are able to outperform the baseline architecture. Adding LSTM recurrence to the feature layers makes the network unable to generalize and overall it performs worse than the baseline. However, this issue could arguably be caused by the sequence length used during training time (6 frames) being too short.

5.3 Future work

There are several aspects of this work that could be expanded on to gain more insight into how to effectively use memory with CNNs. An important step would be to develop a framework that allow the networks to experience longer sequences during training. Increasing the sequence length during training would expose the networks to a wider variety of temporal phenomena that the memory cells could learn to compensate for and potentially gain further advantages over the baseline.

There are also limits imposed by our dataset; images of people looking at moving dots on a flat background do not necessarily reflect natural user behaviour. It would be interesting to find an approach where gaze data could be gathered from users as they are using their computers normally. For instance, one could use an expensive but very accurate eye-tracker, such as the Tobii Pro X3-120, to gather ground-truth data and train a CNN with memory to fit the results. While we could not expect the CNN to perform better than the original eye-tracker the memory could have a regularizing effect that reduces estimation jitter. A neural network might also prove more flexible than a conventional eye-tracker in some cases; it could for instance be cheaper or require a less complicated camera setup.

We found the use of ConvLSTM-cells to be necessary for improving eye-performance but there are many variations of this network structure that could be investigated. Potential approaches include similar modifications such as ConvRNN and ConvGRU-cells. Residual con-
nections in conjunction with the convolutional recurrence or 3D CNNs could also be investigated.

Finally, simply adding more data could be beneficial. Overall our dataset contains roughly 23000 datapoints while [18] used roughly 550000, which is over 20 times as many. This means that we have much less variation in terms of participants and sparser coverage of stimuli-point locations in our dataset. The reason for the discrepancy is that each of our datapoints in comparison requires six times as much memory during training and 20 as much memory for storage, which limited what could be done in the scope of this work. However, these factors could if found necessary be compensated for by simply spending more time on training and investing in more storage capacity.


