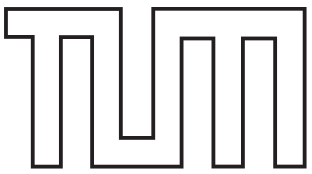


Technische Universität München
Chair of Media Technology
Prof. Dr.-Ing. Eckehard Steinbach

Bachelor Thesis

Encoder-Decoder Sequence to Sequence Model for Head Movement Prediction in 360° Videos

Author:	Xavier Oliva i Jürgens
Matriculation Number:	03681864
Address:	Briennerstr. 39
	80333, München
Advisor:	Dr.-Ing. Tamay Aykut
Begin:	10.06.2019
End:	25.10.2019

With my signature below, I assert that the work in this thesis has been composed by myself independently and no source materials or aids other than those mentioned in the thesis have been used.

München, October 23, 2019 _____

Place, Date Signature

München, October 23, 2019 _____

Place, Date Signature

Abstract

In recent years, the rise of VR, possible thanks to Head Motion Displays getting accessible to the general public, has accelerated research on methods aiming to increase the visual comfort during the virtual experience. To limit bandwidth and hardware requirements, studies have focused on the task of predicting future head movements. Knowing the user's Field of View beforehand can be exploited to exchange only the needed portion of the 360° visual content. In telepresence systems, this information can drastically reduce latency and restrain the appearance of virtual reality sickness.

In this Bachelor's Thesis, we propose a Deep Learning model based on the novel Encoder-Decoder Sequence-to-Sequence model architecture and apply it to the Head Movement Prediction task for investigated latencies $\tau \in [0.1\,\mathrm{s}, 0.2\,\mathrm{s}, ..., 1\,\mathrm{s}]$. Based on the analysis of head motion sequences, we found the viewer motion to be predictable for latencies below 0.6 s. Our proposed model leveraging past sensor-based data outperforms baseline and state-of-the-art Head Movement Prediction techniques, showing its fitness for the given task.

Saliency is a concept used in cognitive neuroscience to describe attention mechanisms making humans focus on the conspicuous parts of their visual image. Using saliency maps, computer-generated topographical representations of saliency, as a content-based feature has the potential to improve the prediction of head motion. We develop a saliency encoder module based on Convolutional Neural Network layers to extract visual features of the saliency map of the user's current Field of View and use it as input to the decoder part of our proposed Seq2Seq model. Nevertheless, the prediction accuracy does not seem to profit from this early-fusion approach, which could serve as a starting point for future work analyzing other methods to integrate saliency in head motion prediction.

Contents

Contents

Chapter 1

Introduction

In recent years, there has been a rise of interest in Virtual Reality (VR). The rise of low-budget smartphone VR headsets like Google Cardboard and Samsung Gear VR and the introduction of 360° videos in popular platforms like YouTube and Facebook has made this technology more accessible to the general public. Prominent use cases of VR include gaming, marketing, simulations in military, sports and medicine, as well as telepresence systems.
Like any new technology, the new opportunities it offers entail new challenges that have to be tackled. According to [EK16], the ideal VR experience given current HMD resolutions with stereoscopic vision requires the visual data to render at 60 frames per second and at a 6K resolution per eye, translating to a bitrate of approximately $20 - 40$ Mbps. Technological advancements in Head Mounted Displays (HMDs) available to general consumers (Oculus Rift, HTC Vive, FOVE VR...) have made them more performant, considerably augmenting the Quality of Service (QoS), yet in many cases these specifications surpass the system capabilities, both from the hardware and the network perspective. These limitations cause stuttering in the image, a drop of the frame rate and a reduction of the resolution, often resulting in unsatisfactory immersive experiences. In some cases, users can develop symptoms of virtual reality sickness, a phenomenon that is very similar to motion sickness.
In VR telepresence systems, where audio and video are shared through a communication network, it is critical to maintain the Motion-to-Photon (M2P) latency under a certain threshold [LYKA14] to achieve a feeling of immersion and to prevent VR sickness. The latency τ is defined by endogenous and exogenous variables like bandwidth, distance and hardware specifications.

In recent years, research on how to minimize the factors responsible for the end-to-end delay has been conducted to maximize the visual comfort during the virtual experience. In VR systems, users are able to freely move their heads within a sphere, making their Field of View (FoV) change through time. The provision of Quality of Service (QoS) in VR systems can largely benefit from predicting future head movements and thus its future FoV.

Exchanging only the portion of the video that will be viewed by the user can considerably reduce bandwidth consumption and the demand of powerful decryption capabilities of the user hardware.

This Bachelor's Thesis proposes a Deep Learning (DL) Head Movement Prediction (HMP) model based on the novel Encoder-Decoder Sequence-to-Sequence (Seq2Seq) model architecture, which has already shown its efficacy on Time Series Forecasting tasks. Using sensor-based information from the past, the network predicts the future head position for a delay $\tau \in [0.1\,\text{s}, 0.2\,\text{s}, ..., 1\,\text{s}]$. Our problem can be phrased as a supervised learning multi-step regression problem.

To improve head motion predictions, other work [NYN18, AEJK17, LZLW19] has not only taken advantage of sensor data, but also of the information about the visual content. Saliency maps, topographical maps representing the conspicuity of each individual pixel of an image, give us an insight into the parts of the image where human attention is usually focused. Humans tend to move their head in such a way that the fixed object is situated in the middle of their FoV, so it is intuitive to assume that saliency maps can help in the task of predicting the future head position. In this work, a saliency encoder module is developed to discuss the feasibility of an early fusion approach to merge the information about saliency maps with sensor-based data as inputs to our models.

The models are trained and tested on a new dataset containing head and gaze data, created in our own lab. 50 subjects watched 23 different 360° videos, totaling 22 h of tracked data.

Chapter 2

Theoretical Framework

In this chapter we lay the theoretical foundations of important concepts we are going to discuss during this Bachelor's thesis.
First of all, we are going to talk about Recurrent Neural Networks (RNNs), a class of artificial neural networks that is especially successful modeling temporal sequences of data. Our proposed network architecture, the Encoder-Decoder Seq2Seq model, is based on a subcategory of RNNs, namely the Long Short-Term Memory (LSTM).
Our network is fed with sensor-based data for the Head Movement Prediction (HMP) task, which is the past head motion history. Nevertheless, related studies suggest adding content-based information. We discuss saliency maps, a visual representation of the pixelwise conspicuity of an image, and describe different approaches to generate them.

2.1 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a class of neural networks where connections between nodes form a directed graph along a temporal sequence. This allows them to show temporal dynamic behaviour. Unlike Feedforward Neural Networks (FFNNs), RNNs have a feedback loop connected to past time steps. Ingesting their own outputs as input allows information to persist. Thus, it is often said that RNNs have *memory*. They have shown to be successful in areas like Natural Language Processing, Speech Recognition and Time Series Prediction, among others. More generally, RNNs are well suited for supervised learning problems where the dataset has a sequential nature.

A naive implementation of a RNN can cause the Vanishing Gradient problem when dealing with long time series. Backpropagation in Feedforward Neural Networks (FFNNs) moves backward from the final error through the outputs, weights and inputs of each hidden layer, assigning those weights responsibility for a portion of the error. This is done by calculating their partial derivatives, which are then used by the optimizer to adjust the weights so that the final error of the prediction is decreased.

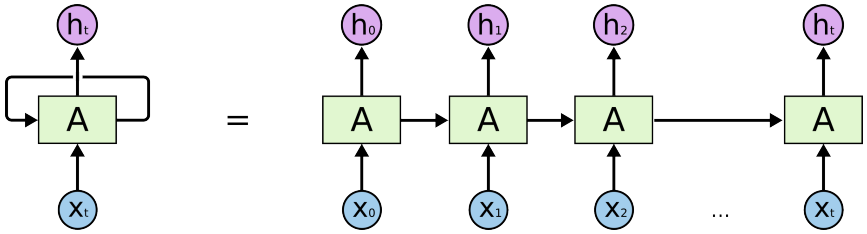


Figure 2.1: Unrolled Recurrent Neural Network (RNN) architecture [Ola15]. A loop allows information to be passed from one step of the network to the next, allowing RNNs to show temporal dynamic behaviour.

For RNNs, a gradient-based technique called Backpropagation Through Time (BPTT) is used for training. The full input sequence is treated as a single training sample, so the total error is the sum of the errors at each time step. The contributions of each time step to the gradient are added. The key difference to traditional FFNNs is the fact that RNNs share the parameters across one layer. The weights, used in every step to create an output, are responsible for the total error, making it necessary to backpropagate gradients.

RNNs are particularly suitable for modeling dynamical systems as they operate on input information as well as a trace of previously acquired information, allowing for direct processing of temporal dependencies.

Nevertheless, naive implementations of RNN have difficulties learning long-term dependencies, that is, interactions between distant time steps. This is due to the fact that the gradient values shrink exponentially fast, eventually vanishing completely after multiple time steps. Gradient contributions from *far away* become zero, and the state at those steps does not contribute to the learning process. Vanishing gradients are not exclusive to RNNs. They also appear in deep Feedforward Neural Network, but given that RNNs tend to deal with long input sequences, the Vanishing Gradient Problem is far more common.

2.1.1 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is an advanced type of RNN, first proposed by Hochreiter et al. in [HS97]. It was developed to solve the problem of vanishing gradients and is able to learn long-term dependencies in the input data. Gated RNNs like LSTMs are based on the idea of creating paths through time that have derivatives that neither vanish nor explode. It consists of 3 gates which compute the amount of past information that is kept or forgotten: the input gate, the forget gate and the output gate.

The first step inside an LSTM unit is to compute the weight by which the past information, saved in the cell state c_{t-1} is multiplied. This operation is done in the forget gate, which yields a vector ranging between $\mathbf{0}$ and $\mathbf{1}$, $\mathbf{0}$ meaning that past information is completely *forgotten*. The forget gate is computed as follows:

$$f_t = \sigma\Big(\boldsymbol{W}^f x_t + \boldsymbol{U}^f h_{t-1} + b^f\Big), \tag{2.1}$$

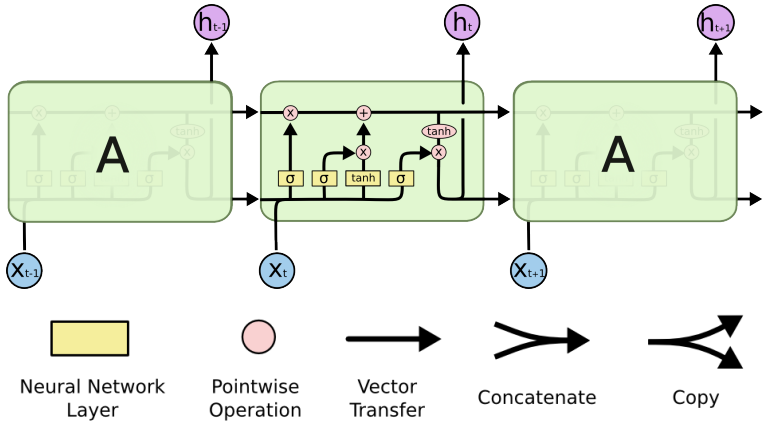


Figure 2.2: Visual representation of the interior of an LSTM cell. [Ola15]. It consists of an input gate, a forget gate and an output gate and is able to learn long-term dependencies.

where $\sigma(\cdot)$ is the sigmoid function.

In the input gate the new information, comprised in the input for the current time step x_t, is used to update the cell state. Thus, the cell state c_t is a weighted addition of the past cell state and the new input, making the LSTM contain information from previous LSTM units in the same layer. The input gate i_t is combined with a tanh layer a_t containing new candidate values for the cell state.

$$i_t = \sigma\Big(\boldsymbol{W}^i x_t + \boldsymbol{U}^i h_{t-1} + b^i\Big) \tag{2.2}$$

$$a_t = \tanh\Big(\boldsymbol{W}^c x_t + \boldsymbol{U}^c h_{t-1} + b^c\Big) \tag{2.3}$$

The cell state is then updated with both the forget and input gate values:

$$c_t = i_t \odot a_t + f_t \odot c_{t-1}, \tag{2.4}$$

where $\odot$ represents the element-wise product (Hadamard product).

In the output gate the LSTM unit generates o_t:

$$o_t = \sigma\Big(\boldsymbol{W}^o x_t + \boldsymbol{U}^o h_{t-1} + b^o\Big). \tag{2.5}$$

o_t is then multiplied by the tanh of the cell state to map the values between $-\mathbf{1}$ and $\mathbf{1}$:

$$h_t = o_t \odot \tanh(c_t) \tag{2.6}$$

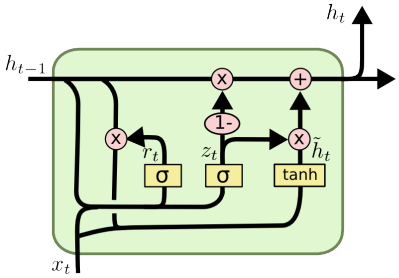
Thus, each unit gets the previous output vector h_{t-1}, the previous cell state c_{t-1} and the input x_t, to generate a new output and cell state (h_t, c_t), which are sent to the next unit,

giving the RNN a dynamic temporal behavior. A LSTM layer consists of m identical LSTM units, with m corresponding to the number of input time steps. A LSTM layer consists of $4(nd + d^2 + d)$ trainable parameters, while n is the size of the input vector and d the size of the output vector. The parameter sharing used in recurrent networks between all units of a same layer relies on the assumption that the conditional probability distribution over the variables at time $t + 1$ given the variables at time t is stationary, meaning that the relationship between the previous and the next time step does not depend on t.

LSTMs solve the problem of Vanishing Gradients by creating the connection between the forget gate activations and the gradients computation. This connection creates a path for information flow through the forget gate for important data from the past that should not be neglected.

2.1.2 Gated Recurrent Units (GRUs)

The LSTM was followed by the Gated Recurrent Unit (GRU), another type of RNN, introduced by Cho et al. [CvMG+14] in 2014. It has the same goal of tracking long-term dependencies effectively while mitigating the vanishing/exploding gradients problem. It has a similar structure to the LSTM, as both share a forget gate, but it has fewer parameters, considering the lack of output gate (see Figure 2.3). The resulting model is simpler than standard LSTM models, but computationally more efficient to train. In certain tasks like polyphonic music modeling and speech signal modeling it has shown to achieve a very similar performance, which has made it increasingly popular in the last years. However, due to the larger amount of parameters to learn, LSTMs can potentially remember longer sequences and learn more complex hypotheses.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 2.3: Visual representation of the interior of a GRU cell [Ola15]. It consists of an input and a forget gate. Its simpler architecture makes it computationally more efficient than LSTMs.

2.2 Saliency Maps

Saliency is a concept used in the fields of cognitive neuroscience and neuropsychology describing the attentional mechanism that makes humans and animals focus on the conspicuous parts of their visual perception. Humans unconsciously gain the ability to assess

salience during their lifetime by using memories or past stimuli, e.g during the process of learning how to read or how to drive. Letters and street signs get more salient, since they gain importance in order to gather data of the environment or to take decisions. The human brain learns how to concentrate on the important features of an image in order to react fast, or as a way to filter the visual data. Saliency can be due to low-level features, such as colors, intensity or contrast between items, but can also be guided by top-down or even anticipatory mechanisms, e.g especially rare objects, or looking ahead of a moving object.

The concept of saliency has been later adapted to the fields of Computer Vision (CV) and image processing as a way to perform feature extraction [IKN98], image segmentation [AEWS08] and object recognition [GLM17] among others.

As stated in [BSI13], in literature, saliency and visual attention are often confused. Visual attention is a broad concept covering many topics (e.g., bottom-up/top-down, overt/covert, spatial/spatio-temporal, and space-based/object-based attention). Visual saliency, on the other hand, mainly refers to bottom-up processes that render certain image regions that are more conspicuous [BSI13].

Saliency maps, topographically arranged maps that represent visual saliency of an image or video, were first proposed by Koch et al. in [KU87]. Neuroscientists Itti et al. developed a first complete implementation and verification of the proposed model in [IKN98]. The purpose of saliency maps is to represent the conspicuity of each pixel of an image, the cognitive model being inspired by psychological and neuro-physiological findings. The proposed dynamical neural network filters the image colors, intensity and orientations separately and combines the normalized feature maps to generate the saliency map in a bottom-up approach, see Figure 2.4.

The intensity of the image I is obtained as

$$I = (r + b + g)/3. \tag{2.7}$$

I is then used to create a Gaussian pyramid $I(\sigma)$, where $\sigma \in [0, 1, ..., 8]$. The color channels r, g and b are normalized by I to decouple hue from intensity in locations where I is larger than 1/10 of its maximal value. Four broadly-tuned color channels are created (for red, green, blue and yellow):

$$R = r - (g + b)/2, \tag{2.8}$$

$$G = g - (r + b)/2, \tag{2.9}$$

$$B = b - (r + g)/2, \tag{2.10}$$

$$Y = (r + g)/2 - |r - g|/2 - b. \tag{2.11}$$

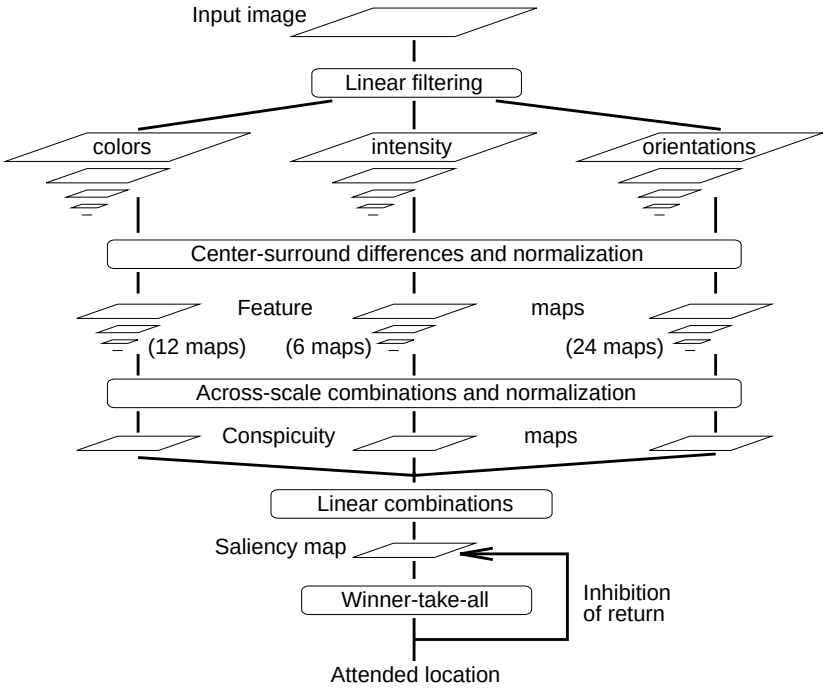


Figure 2.4: Itti et al.'s bottom-up approach [IKN98]. The input image is filtered based on color, intensity and orientations.

Four Gaussian pyramids ($R(\sigma)$, $G(\sigma)$, $B(\sigma)$ and $Y(\sigma)$) are created from these color channels. Itti et al. [IKN98] define a set of six features maps regarding intensity contrast $\mathcal{I}(c, s)$ with $c \in 2, 3, 4$ and $s = c + d$, with $d \in 3, 4$.:

$$\mathcal{I}(c, s) = |I(c) \ominus I(s)|, \tag{2.12}$$

with $\ominus$ being the center-surround difference between the center fine scale c and a surround coarser scale s. The set of color maps is represented using a color double-opponent system [IKN98]. In the center of their receptive fields, neurons are excited by a certain color (e.g. red) and inhibited by another one (e.g. green), while the inverse happens in the surroundings. Chromatic opponency is accounted for in color feature maps, resulting in 12 maps:

$$\mathcal{RG}(c, s) = |(R(c) - G(c)) \ominus (G(s) - R(s))|, \tag{2.13}$$

$$\mathcal{BY}(c, s) = |(B(c) - Y(c)) \ominus (Y(s) - B(s))|. \tag{2.14}$$

Equation 2.13 accounts for red/green and green/red double opponency, equation 2.14 for blue/yellow and yellow/blue. Orientation feature maps are are generated by Gabor pyramids $O(\sigma, \theta)$, where $\theta \in 0°, 45°, 90°, 135°$ represents the preferred orientation. 24 orientation

feature maps are obtained by encoding local orientation contrast between center and surround scales:

$$\mathcal{O}(c, s, \theta) = |O(c, \theta) \ominus O(s, \theta)|. \tag{2.15}$$

In total, 42 feature maps are generated. A normalization operator $\mathcal{N}(.)$ is applied, replicating cortical lateral inhibition mechanisms (neighboring similar features inhibit each other), making the different feature maps comparable. The feature maps are combined into three conspicuity maps:

$$\bar{\mathcal{I}} = \bigoplus_{c=2}^{4} \bigoplus_{s=c+3}^{c=4} \mathcal{N}(\mathcal{I}(c, s)), \tag{2.16}$$

$$\bar{\mathcal{C}} = \bigoplus_{c=2}^{4} \bigoplus_{s=c+3}^{c=4} \Big[\mathcal{N}(\mathcal{RG}(c, s)) + \mathcal{N}(\mathcal{BY}(c, s))\Big] \tag{2.17}$$

$$\bar{\mathcal{O}} = \sum_{\theta \in \{0°, 45°, 90°, 135°\}} \mathcal{N}\Big(\bigoplus_{c=2}^{4} \bigoplus_{s=c+3}^{c=4} \mathcal{N}(\mathcal{O}(c, s, \theta))\Big). \tag{2.18}$$

The saliency map is finally computed as the mean between the three normalized conspicuity maps:

$$S = \frac{1}{3}\Big(\mathcal{N}(\bar{\mathcal{I}}) + \mathcal{N}(\bar{\mathcal{C}}) + \mathcal{N}(\bar{\mathcal{O}})\Big). \tag{2.19}$$

Thus, the most salient pixel of the saliency map represents the most salient location of the image, to which the focus of attention should be directed [IKN98].

This was the first approach to generate saliency maps, taking neurological assumptions in order to replicate human reactions to visual stimuli, and is classified as a cognitive model in Borji et al.'s [BI13] study. After Itti et al.'s work [IKN98] introducing a saliency map implementation, numerous research studies have investigated different approaches to generate saliency maps. We will give a brief synopsis of these different procedures, focusing on the more novel approach based on deep learning algorithms.

In [BI13], Borji et al. review and compare 65 different models, classifying them into 7 model categories based on their proposed approach:

1. **Cognitive models**, which are directly or indirectly inspired by cognitive concepts, as discussed above. Other approaches include Le Meur et al. [LMLCBT06] and Navalpakkam et al. [NI06].
2. **Bayesian models** use prior knowledge (e.g. scene context) and sensory information (e.g. target features) and combine them probabilistically according to Bayes' rule. Instances of such frameworks can be found in [Tor03], [ZTM+08], [ZTC09] and [IB09].
3. **Decision theoretic models** affirm that perceptual systems evolve to produce decisions about the states of the surrounding environment that are optimal in a decision

theoretic sense (e.g. minimum probability of error). The underlying idea is that visual attention is optimally driven by the end task. [GV05][MV09]

4. **Information theoretic models** state that localized saliency computation serves to maximize information sampled from the environment. The most informative parts of a scene are selected and the rest is discarded. ([BT06], [HZ09])

5. **Graphical models** propose a probabilistic framework in which a graph denotes the conditional independence structure between random variables. Approaches like Hidden Markov Models (HMM), Dynamic Bayesian Networks (DBN), and Conditional Random Fields (CRF) are used. ([SAA02], [LYS+10], [LJHX08])

6. In **spectral analysis models**, images are analyzed in the frequency domain, assuming that statistical singularities in the spectrum may be responsible for conspicuous regions in the original image. ([HZ07], [GZ09])

7. In **pattern classification models**, machine learning algorithms are applied to model visual attention, using datasets containing eye-fixations or labeled salient regions. One advantage of this type of approach is that less assumptions have to be made, given that the learning process is data driven. In [JEDT09], Judd et al. propose a supervised learning model (support vector machine) of saliency which combines both bottom-up image-based saliency cues and top-down image semantic dependent cues.

In Figure 2.5 the saliency map classification proposed by [BI13] and the corresponding studies are illustrated.

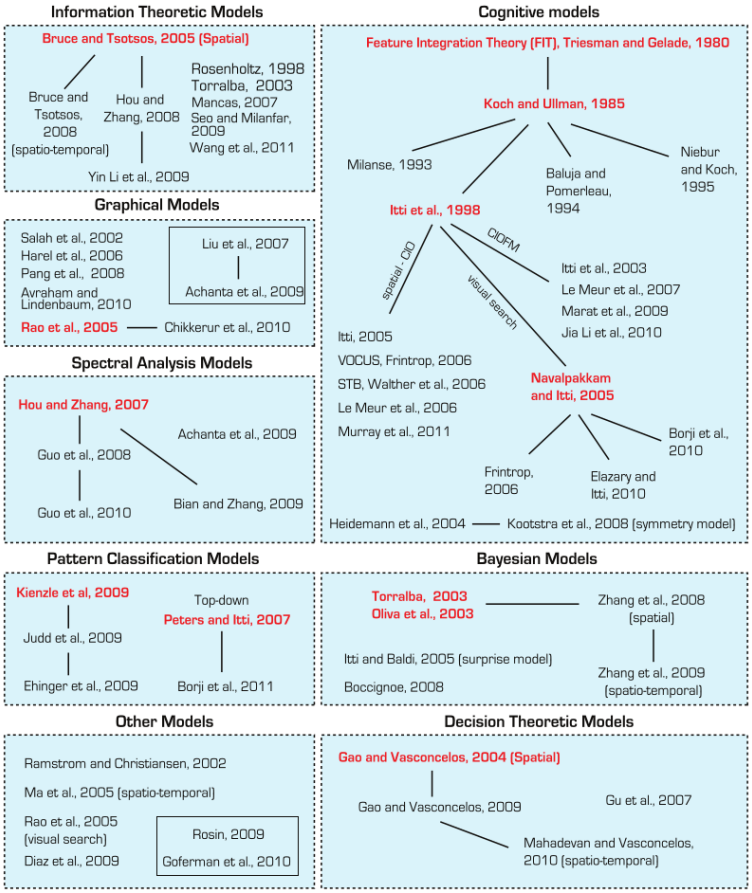


Figure 2.5: Borji et al.'s [BI13] illustration of the proposed saliency map model categories.

2.2.1 Saliency Maps Using Deep Learning

Deep learning has revolutionized the way visual problems in machine learning are addressed and has led to breakthroughs in fields like image classification, image segmentation and object detection. Convolutional Neural Networks (CNNs), a class of deep neural networks, are commonly applied to visual imagery. LeNet-5 [LBB+98], developed by LeCun et al. in 1998, is one of the first CNNs and was used to classify handwritten and machine-printed characters (see Figure 2.6). CNNs are the foundation of modern state-of-the art deep learning-based Computer Vision. CNNs gained general interest in 2012, when the convolutional network called AlexNet [KSH12] won the ImageNet Large Scale Visual Recognition Challenge and impressed with its high performance. These networks are built upon 3 main ideas: local receptive fields, shared weights and spacial subsampling. They are computationally efficient and automatically detect the important features without direct human supervision, making them suitable for saliency map generation.

Deep learning is a class of machine learning algorithms that use multiple layers to progressively extract higher level features from raw input. The approaches described below can be

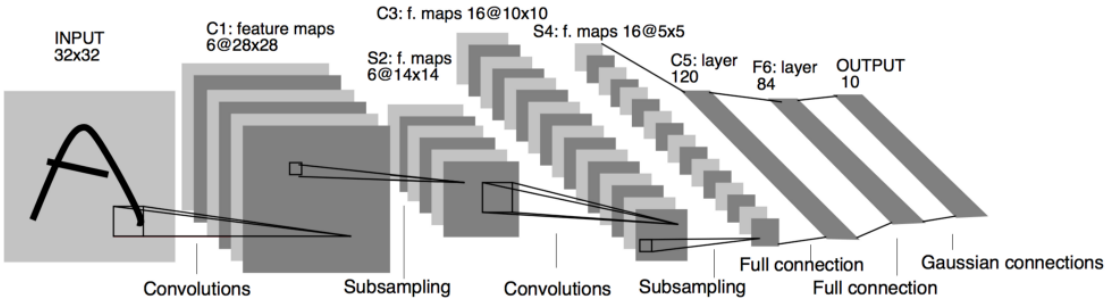


Figure 2.6: Proposed architecture of LeNet-5 [LBB+98], one of the first CNNs. It consists of two sets of convolutional and average pooling layers, followed by a flattening convolutional layer, two fully-connected layers and a final softmax classifier.

classified as pattern classification models, with the specificity of using models with several layers. The MIT Saliency Benchmark [BJB+] shows that Deep Learning (DL) approaches considerably outperform previous models.

In 2015, Huang et al. proposed Saliency in Context (SALICON) [HSBZ15]. Based on Itti et al.'s assumption [IK01] that the perceptual saliency of stimuli critically depends on surrounding context. In order to use semantic information to predict human fixations, Huang et al. use one of three popular Deep Neural Network (DNN) architectures for object recognition: VGG-16 [SZ14], AlexNet [KSH12] and GoogleNet [SLJ+15]. After fine-tuning, VGG-16 [SZ14] is the DNN that performed best in this task. The SALICON model significantly outperformed previous approaches.
Pan et al. introduced SALGAN [PFM+17], a Generative Adversarial Network (GAN) (network consisting of a generator and discriminator which contest with each other). SALGAN consists of two networks: The generator predicts saliency maps from raw pixels of an input image. It consists of an encoder (identical in architecture to VGG-16 [SZ14]) and a decoder, which is structured in the same way as the encoder, but with the ordering of layers reversed and upsampling layers instead of pooling layers. The discriminator takes the output of the generator to discriminate whether a saliency map is a predicted one or ground truth. It consists of six 3×3 kernel convolutions interspersed with three pooling layers, followed by three fully connected layers. SALGAN is the first adversarial-based approach to saliency prediction and achieved state-of-the-art performance.
In 2016, Cornia et al. [CBSC16] proposed ML-NET, a model which combines features extracted at different levels of CNNs. It composed of three main blocks: a feature extraction CNN, a feature encoding network, that weights low and high level feature maps, and a prior learning network.

Kümmerer et al. developed a state-of-the-art model for saliency prediction, DeepGaze II [KWGB17]. The model is based on deep features that are trained on object recognition using the normalized VGG-19 [SZ14] network. The feature maps of a selection of high-level convolutional layers are up-sampled so that spatial resolution is sufficient for precise pre-

(a) Original Image

(b) Saliency map

Figure 2.7: Image and corresponding saliency map using DeepGaze II with centerbias [KWGB17]

diction. These feature maps are combined and used as input for a second network (*readout network*), consisting of four 1×1 convolutional layers. The final output is convolved with a Gaussian blur to regularize predictions and center bias is modeled as a prior distribution.

$$S(x, y) = O(x, y) * G_\sigma, \tag{2.20}$$

$$S'(x, y) = S(x, y) + \log p_{\text{baseline}}(x, y). \tag{2.21}$$

Finally, $S'(x, y)$ is converted into a probability distribution over the image by the means of a softmax:

$$p(x, y) = \frac{\exp(S'(x, y))}{\sum_{x,y} \exp(S'(x, y))}. \tag{2.22}$$

DeepGaze II achieves top performance in both area under the curve (AUC) with center bias and shuffled area under the curve (sAUC) without center bias.

Table 2.1: Comparative results on the MIT Saliency Benchmark Results [BJB+] (MIT300 test set)

Model Name	AUC-Judd	SIM	EMD	AUC-Borji	sAUC	CC	NSS	KL
Baseline: infinite humans	0.92	1	0	0.88	0.81	1	3.29	0
SALICON [HSBZ15]	0.87	0.60	2.62	0.85	**0.74**	**0.74**	**2.12**	**0.54**
SALGAN [PFM+17]	0.86	**0.63**	**2.29**	0.81	0.72	0.73	2.04	1.07
ML-NET [CBSC16]	0.85	0.59	2.63	0.75	0.70	0.67	2.05	1.10
Deep Gaze II [KWGB17]	**0.88**	0.46	3.98	**0.86**	0.72	0.52	1.29	0.96

Chapter 3

Related Work

In recent years, numerous studies have worked on the task of predicting future head motion. Improving the accuracy of such systems can be critical to minimize latency, bandwidth and general hardware demands. All these factors can hinder a pleasant and immersive VR experience. In the following, we will discuss different approaches found in literature using both sensor-based data and visual saliency to predict head movements in VR systems.

3.1 Head Movement Prediction Using Sensor-Based Information

Time series modeling head motion show short-term autocorrelations, which can be used to predict future behaviour of the user's position inside the 360° VR sphere. In this chapter we examine an approach relying solely on past sensor-based information.

In [BWZ+16], Bao et al. propose a motion-based transmission mechanism to address the challenges 360° videos pose from the network's perspective. It was found that viewer motion can be confidently predicted in 100 ms $\sim$ 500 ms. In this study, the proposed system predicts both viewer's head motion and the prediction's deviation. Using a time window of pitch and yaw angles as input, the proposed network predicts the future head position at $t + T_r$. During the experiments, different values are set for $T_r \in \{0.1\,\text{s}, 0.2\,\text{s}, ..., 1\,\text{s}\}$. Three models are compared: a naive approach, Linear Regression (LR) and a Feedforward Neural Network (FFNN) with 3 layers and 5 hidden neurons. Two different models are trained for pitch and yaw. The neural network performs best for the prediction of the pitch angle, while LR shows a similar performance for the yaw angle's prediction. Using the predicted viewpoint and its corresponding predicted deviation, they design a motion-prediction-based transmission algorithm that reduces bandwidth consumption by 45%.

3.2 Head Movement Prediction Using Saliency Maps

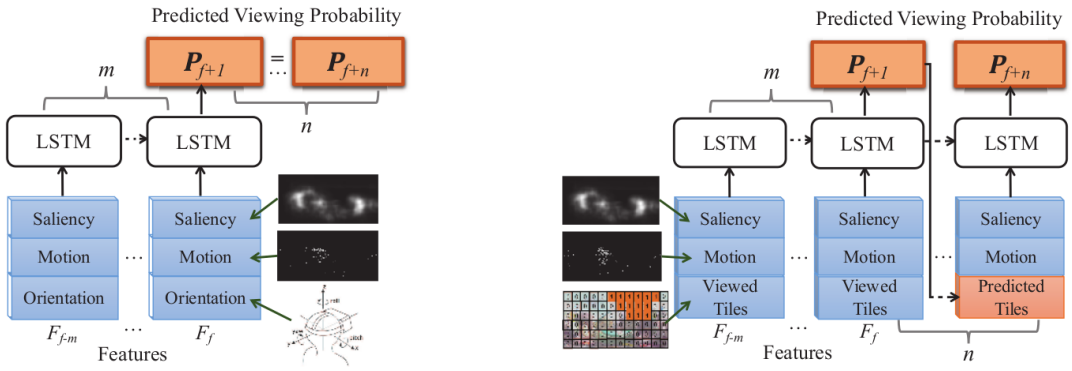
Understanding users' visual attention in HMDs is essential to the head movement prediction task, since users are more likely to move towards regions of interest. Saliency maps give us an understanding of the parts of the image where human attention is focused on. Saliency detection can thus strongly benefit the task of predicting future head motion. By combining saliency-based data with sensor-based information, a more accurate head movement prediction may be achieved.

In 2017, Fan et al. [FLL+17] proposed two fixation prediction networks for 360° videos (see Figure 3.1), leveraging both sensor- and content-related features. The goal is to reduce consumed bandwidth, initial buffering time and running time in 360° video streaming platforms. To the best of our knowledge, this is the first approach using saliency maps as input for view prediction in HMDs. Among the content-related features, Fan et al. use saliency maps and motion-vectors for each frame, as sensor-related features the viewer's orientation and the viewed tiles. They use Cornia et al.'s [CBSC16] ML-NET to generate the saliency maps, and the fixation prediction network is based on a Recurrent Neural Network (RNN) that extracts information from the temporal sequence of video frames. Both networks use Long Short-Term Memory (LSTM) layers to take features of the m past video frames and predict the viewing probability $p_f^t \in [0, 1]$ of tiles for n future video frames.
The orientation-based network takes the view orientation as inputs, and predicts the viewing probability of the next frame. The same prediction is then reused for all the n future frames. The tile-based network, however, takes the viewed tiles as inputs, and predicts the viewing probability of the next n frames. To select the optimal parameters, three metrics are used:

1. **accuracy**: ratio of correctly classified tiles to the union of predicted and viewed tiles
2. **F-score**: harmonic mean of the precision and recall, where the precision and recall are the ratios of correctly predicted tiles to the predicted and viewed tiles, respectively
3. **ranking loss**: number of tile pairs that are incorrectly ordered by probability, normalized to the number of tiles

Both networks are trained to minimize the cross-entropy loss, using Stochastic Gradient Descent (SGD) with a learning rate of 10^{-2}, while $m = 30$ and $n = 30$. All tiles for which $p_f^t \geq \rho$ with $\rho = 0.5$ are considered as predicted tiles.
The optimal model parameters are found to be 1 LSTM layer with 512 neurons and dropout for the orientation-based network and 2 LSTM layers with 1024 neurons without dropout for the tile-based network. The orientation-based network shows to be slightly more precise in fixation prediction. The former achieves 86.35% accuracy, while the latter achieves 84.22%.

More recently, Xu et al. [XDW+18] proposed a gaze prediction network using saliency maps. Given a sequence of 360° VR video frames, gaze prediction aims to regress the future gaze coordinates corresponding to the future T frames. Saliency maps and temporal



(a) Orientation-based network (b) Tile-based network

Figure 3.1: Proposed view prediction networks in Fan et al.'s study [FLL+17]. The orientation-based network shows to be slightly more accurate for view prediction.

saliency, inferred from the optical flow between neighboring frames, are used as input to the network. It consists of 3 modules, as can be seen in Figure 3.2: a trajectory encoder, a saliency encoder and a displacement prediction module. The trajectory encoder module encodes the gaze path history of a user. For each video clip, the gaze points ($\{l_1^p, l_2^p, ..., l_t^p\}$), where $l_t = (x_t, y_t)$ and $x_t \in [0, 360]$, $y_t \in [-90, 90]$ are fed to a RNN based on two stacked LSTM layers, both with 128 neurons.
Xu et al. propose a multi-scale scheme to calculate saliency in the saliency encoder module:

1. Local saliency: the saliency of a local patch centered at the current gaze point, computed by a Gaussian-based saliency approximation for efficiency reasons.
2. FoV saliency: the saliency of the current Field of View.
3. Global saliency: the saliency of the global scene (equirectangular image representing the 360° scene).

The saliency maps of the FoV and the 360° equirectangular image are generated using SALNET [PSGiN+16], while the optical flow between frames is computed by FlowNet 2.0 [IMS+17]. Motivated by the effectiveness of [LH16], Xu et al. [XDW+18] propose to concatenate the RGB images with all the generated saliency maps and feed them into the pretrained Inception-ResNet-V2 network [SIVA17] to extract more accurate saliency features for gaze prediction. The displacement prediction module takes the output of saliency encoder and trajectory encoder modules to estimate the displacement between the gaze point at time $t+1$ and t, δl_{t+1}^p. This module consists of two fully connected layers with 1000 and 2 neurons respectively.

The interval between two neighboring frames is $\frac{1}{5}$s, the history gaze path in the past five frames is used to predict the gaze points in next five frames ($obs = 5$ and $T = 5$). Thus, the past 1 second is exploited to predict the future 1 second. Results show the rise in performance when adding temporal saliency with optical flow and combining the saliency

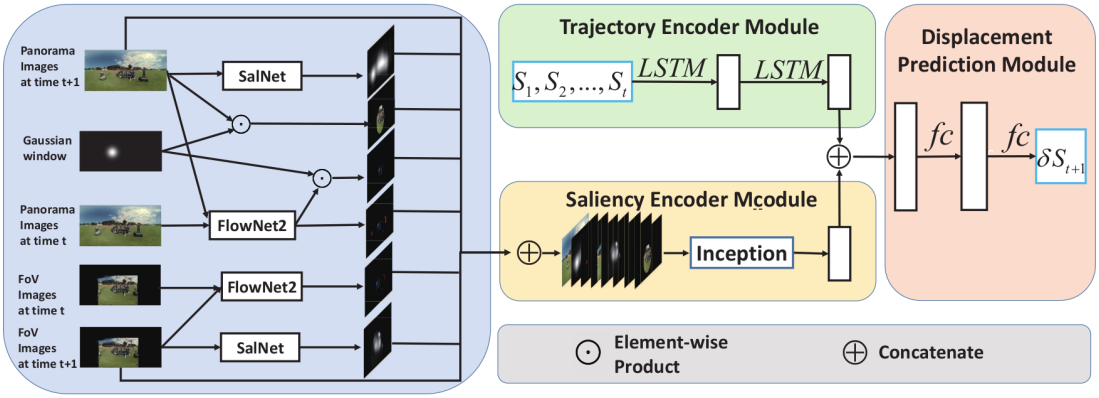


Figure 3.2: Xu et al.'s [XDW+18] proposed network, consisting of a trajectory encoder module and saliency encoder module, whose output is combined to estimate the displacement prediction of the gaze.

map and RGB image in the saliency encoder module.

Given that both described studies do not use the same metrics and that the code is not publicly available, it is not possible to quantitatively compare both methods.

In Aladagli et al.'s work [AEJK17], the relationship between sequences of fixations predicted from graph-based visual saliency maps and actual head movements of individual users is investigated. This is done by applying cross-correlation between the series of predicted attention centres and recorded view centres for varying window widths of samples and delays. Although a similarity between the two sequences is found, no meaningful relationship between them can be identified using a winner-take-all approach on saliency maps, which can be due to the flickering of the saliency map peaks. A frame-based approach to predict sequential fixation using only saliency maps is not robust enough. These findings have motivated other studies to find methods to accurately predict head motion in VR using saliency maps as a content-based feature.

In [NYN18], Nguyen et al. present a different approach to use saliency maps in the context of HMP in VR. Instead of predicting the saliency map of the current FoV, Nguyen et al. build a dataset for panoramic saliency and train a saliency detection model for 360° videos, called PanoSalNet. Thus, the the saliency map is constructed based on the whole equirectangular image and combined with what is called head-orientation map: the tile pointed by current head orientation is given a likelihood of 1 and a Gaussian kernel is applied to make the gradually cover the whole viewport. As seen in Figure 3.3, both the panoramic saliency map and the head-orientation map are fed to a model consisting of multiple stacked LSTM layers to handle more complex data: α layers and β neurons per layer. The model parameters are updated using the Root Mean Square Propagation (RMSprop) method, which has the ability to dynamically adjust the learning rate during training time.

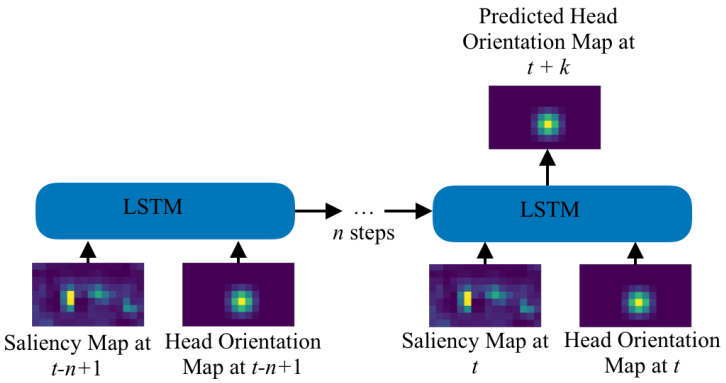


Figure 3.3: LSTM architecture proposed by Nguyen et al. [NYN18]. The saliency map of the equirectangular image and the head orientation map are fed into the model to predict the head orientation map in k time steps. Best performance is achieved using the proposed saliency map prediction for panoramic images, called PanoSalNet.

The past one second of data (n time steps) is used to predict the head orientation in k time steps, ranging from 0.5 to 2.5 seconds. The equirectangular frame is split into 16×9 tiles. The model is evaluated using different saliency map detection methods, among which PanoSalNet [NYN18] and Deep Convnet [PSGiN$^+$16], a saliency model for regular images. The model performs best by combining head orientation and the saliency map generated by PanoSalNet. For a higher delay k, the head prediction accuracy converges, as the input gets less relevant for the output. The model shows varying accuracy based on the video content, achieving better performance on static videos than on dynamic ones.

Chapter 4

Methods

4.1 Approach

The key contribution of this Bachelor's thesis is to propose a Head Movement Prediction (HMP) algorithm based on a novel deep neural network architecture, the Sequence-to-Sequence (Seq2Seq) Model. Although first conceived for the field of Machine Translation, the model can be also applied to other areas like Time Series Forecasting and thus for the Head Movement Prediction (HMP) task. The architecture consists of two modules that work synchronously to generate future head positions. The main advantage of the Seq2Seq model is the flexibility it offers when dealing with many to many (m to n) sequence learning setups.

4.1.1 Encoder-Decoder Sequence-to-Sequence (Seq2Seq) Model

Sequence-to-Sequence Models were introduced for the first time in 2014 by Sutskever et al. [SVL14]. They map a fixed length input to a fixed length output, where the length of the input and output may differ. In [SVL14] this model architecture is successfully used in the field of Machine Translation to translate English sentences to French. Seq2Seq models have since been applied to other fields, such as Text Summarization, Conversational Modeling, Image Captioning and Time Series Forecasting.

Seq2Seq models use an encoder-decoder framework, mapping an arbitrarily long input sequence to an arbitrarily long output sequence. The model consists of 3 different parts: the encoder module, the intermediate hidden state (context vector) and the decoder module (see Figure 4.1).

Introduced by Sutskever et al. [SVL14], the Sequence-to-Sequence (Seq2Seq) model was first applied to the field of Machine Translation. One of the main difficulties of translating sentences (sequences of words) is the fact that the input and output length may differ, which make these types of problems inappropriate for regular LSTM networks. Furthermore, the

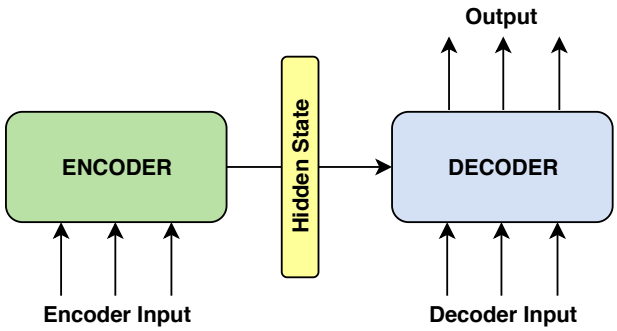


Figure 4.1: Basic Encoder-Decoder Model architecture. The encoder transforms the input into the hidden state, a fixed length representation of the input sequence. This information is then used as initial state for the decoder module, which is responsible for generating the output sequence. Both encoder and decoder consist of RNN layers.

language syntax may be completely different, so translating each word in the same order may be inaccurate. To tackle these problems, Sutskever et al. devised a more flexible model consisting of encoder and decoder. The encoder is responsible for accepting the input words and collecting information for each element, to finally encapsulate the sentence information in its final state, which acts as initial state to the decoder part of the model. The first LSTM cell of the decoder processes the hidden state and creates a first word as output, which is reinjected as input to the next cell. This process continues until a `<STOP>` string is detected in the output. In question-answering problems, the output sequence is a collection of all words from the answer. In Figure 4.2 a basic visual representation of a Seq2Seq model applied to question-answering is shown.

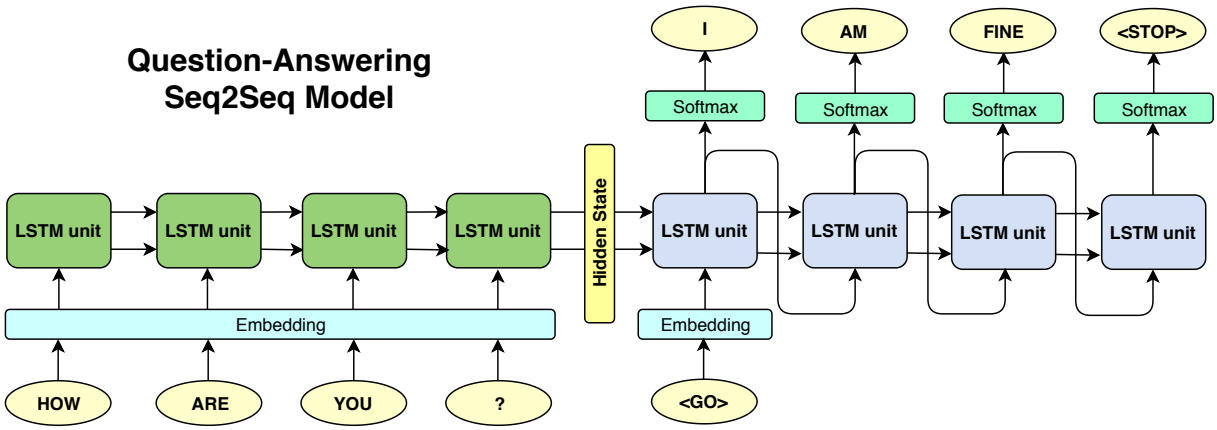


Figure 4.2: Visual representation of a Seq2Seq model applied to the question-answering problem. The output sequence is a collection of all words from the answer. Input and output sequence lengths tend to differ.

In this work, we apply this novel network architecture to the Head Movement Prediction problem. Encoder and decoder are built using a single or a stack of several RNN layers. In

practice, gated RNNs like LSTMs or GRUs are used to achieve a higher performance. The intermediate encoder vector acts as initial hidden state of the decoder module and can be interpreted as the information about the past. The hidden state contains the network's understanding of the past head motion data, with a window size of $T_W = 250$ms. The decoder module takes this information to generate the final prediction for future time steps, with a delay $\tau \in [0.1\,\text{s}, 0.2\,\text{s}, ..., 1\,\text{s}]$.

In Kaggle, a popular online community of data scientists and machine learning practitioners, Google launched a Time Series Forecasting Competition in 2017 to test state-of-the-art methods on the problem of forecasting future web traffic for Wikipedia articles. The winning approach, developed by Suilin [Sui17], was based on an Encoder-Decoder architecture, showing a successful application of this type of network for time series prediction.

There are three ways of setting the decoder input for our problem. First, it can be added externally for every time step using a constant value (no additional information), letting the hidden states propagate the information from the context vector trhough the gated RNN unit cells. Another option is to use past data points from the sequence as external encoder input for each step. Finally, the previous unit's output can be reinjected as input to the next time step's cell. That way, only the decoder input for the first time step has to be added externally, while all the others are generated dynamically.
Especially in sequence prediction problems, it can be useful to use the decoder output of the previous time step y_{t-1} as input for the current time step. In Machine Translation Seq2Seq problems, it is common to use the previous predicted word as input for the next one, since it can give valuable insight into the grammatical or lexical nature of the next word to reconstruct a correct sentence. In the Head Movement Prediction task, previous outputs can also potentially help to predict the head position in the near future, since there is a strong correlation between the values corresponding to successive time steps.

Teacher Forcing (TF) is a strategy for training RNNs models and can be applied to the Encoder-Decoder Seq2Seq model. To allow a quick and efficient training, Teacher Forcing uses the ground truth as decoder input instead of the output generated by the network. Thus, the real values of the head motion position of future time steps are used in place of the outputs of the decoder cell. In the beginning of the training, the network's weights are not optimized yet, generating suboptimal predictions. Reusing these values can result in a slow convergence or model instability during training. Using the ground truth can help speed up the process of training the model, as the RNN unit cells are trained from the beginning with a correct input for each time step and the weights are optimized to correctly map the output. In inference mode, the future values are not available anymore. Thus, the predicted values for each time step and the updated hidden state of the RNN unit are reinjected at subsequent time steps (see Figure 4.3).

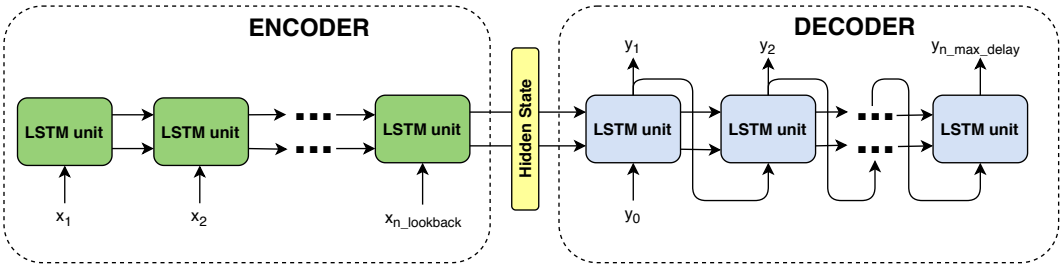


Figure 4.3: Basic Encoder-Decoder Model architecture, reinjecting the decoder's output as input to the next RNN unit. To accelerate training, it can be trained using the Teacher Forcing method, which applies the ground truth as input to the decoder module.

Attention

All information about the past must pass through the hidden state to get to the decoder. It can be considered as the model bottleneck, since the performance of the Seq2Seq model greatly relies on the ability of the encoder to generate a vector summarizing the information it gets as input. On the one hand, this can be controlled by the vector size, as more weights can make the hidden state contain more information. On the other hand, Bahdanau et al. [BCB14] tackled this issue by proposing the concept of attention. Attention is a mechanism by which the decoder does not only rely on the hidden state vector, but has access to all the past states of the encoder. When the model learns attention, it learns where to look in the encoder and put the relevant knowledge into the context vector.

The context vector becomes a weighted sum of all the past encoder states, not only the last one (see Figure 4.4). During training, the attention layer learns by which weights a_{ij} each state of the encoder has to be multiplied to be used as initial state of the decoder module. For the head motion prediction task, the attention layer has the potential to generate an improved initial state, with which the decoder module could make better predictions.

Attention-based models are classified into two broad categories, global and local. The idea of a global attentional model is to consider all the hidden states of the encoder, while in local attention models the attention is placed on a few source positions. In the following, we will use global attention models to get the most out of the past data.

4.1.2 Saliency Encoder Module

To improve head motion predictions, a model may not only leverage sensor data, but, besides that, it seems promising to incorporate the visual content into the prediction mechanism. Saliency maps, topographical maps representing the conspicuity of each individual pixel of an image, give us an insight into the parts of the image where human attention is focused. Humans tend to move their head in such a way that the fixed object is situated in the middle of their FoV, so it is intuitive to assume that saliency maps can help in the task of predicting future head positions.

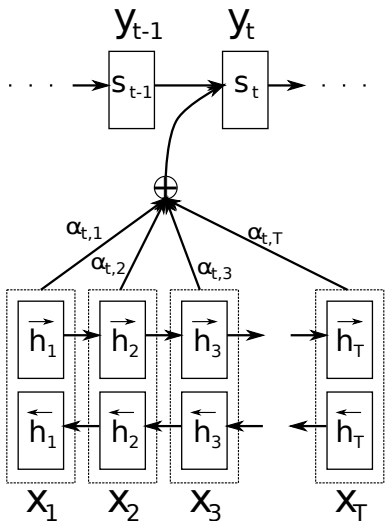


Figure 4.4: The graphical illustration of the proposed model generating y_t given an encoder input $(x_1, x_2, ..., x_T)$ [BCB14]. The attention layer learns the parameters $a_{t,1}, ..., a_{t,T}$ to generate a final context state consisting of a weighted sum of all the encoder hidden states.

In [NYN18], Nguyen et al. use a single stacked LSTM layer with n steps to predict head orientation at time $t + k$. The network is fed with n saliency maps merged with the head orientation map, responsible to contain sensor- and content-related information. The problem with this approach is that in order to input $2 \times n$ images per sample, the resolution of these images has to be low to avoid memory storage issues. Thus, the head orientation history is very vague and the network is not capable to provide an accurate prediction. The prediction consists of a single head orientation map at the same insufficient resolution.

We take a different approach. In order to test the efficacy of using saliency maps directly as input to the Encoder-Decoder Seq2Seq model, we feed the encoder with the head orientation history, while the decoder is fed with the saliency map SM_t generated from the user's perspective viewport image at time t. Using a single image as input to our model makes it more memory efficient than other approaches at the cost of a lack of information about temporal saliency. Nevertheless, the last frame visualized by the user is the most important one for the Head Movement Prediction task. We develop a saliency encoder module based on sequential CNN and Max-Pooling layers responsible for extracting the features from the saliency maps and downsampling them. The resulting vector is the decoder's first input. In subsequent time steps, the values are reinjected from the previous LSTM unit. The saliency encoder module has the potential to create a vector containing useful content-based information, from which the decoder can benefit to make better predictions.

4.2 Experiments

4.2.1 Dataset

The used dataset containing the sensor and content related data was created in our lab. The data gathering setup was performed with a Windows 10 PC and the FOVE HMD headset. Subjects were asked to watch 360° videos and the tracked data was gathered using the FOVE C++ API. The videos were displayed on the integrated SteamVR Media Player. Participants sat on a swivel chair, allowing them to freely move and explore the 360° visual scene.
The dataset consists of 50 subjects and of 23 different videos in equirectangular format (18 monoscopic, 5 stereoscopic). Each session was approximately 45 minutes long. Before each session, the gaze tracking system was calibrated to improve the accuracy of the data. The videos have a resolution of 4K with a frame rate of 30 fps. The maximal video width is 3840 px and the maximal height varies between 1920 px and 2160 px, depending on the video. The length of the videos varies between 45s and 60s. Videos are in different types of environments and circumstances. 19 videos are static, in 4 of them the visual scene is in motion. User age ranges from 19 to 63, with an average of 27 years. 81% of the participants are male and 19% female.

23 videos of approximately one minute were shown with no task and the subjects could freely move and explore the environment. 5 videos were shown twice, the second time they visualized the video they were given a specific task. Users could take a break at any point, especially if they felt motion sickness. At least one pause was done during each experiment, usually in the middle of the session. The total length of the dataset is approximately 22 hours. On the one hand, it consists of the content data, which are the 23 videos the users viewed. While watching the VR videos with the FOVE Eye Tracking VR Headset, head and gaze position data were tracked. Head and gaze orientations are saved using equirectangular pixel coordinates. The given pixel value of the head orientation represents the pixel that is situated in the exact middle of the current FoV of the user. As the gaze data is only correctly tracked when the user's eyes are open, the dataset includes a value in $\{0, 0.5, 1\}$, giving the confidence of the gaze point (0 if eyes are closed, 0.5 if one eye is open and 1 if both are). The roll angle of the head allows to reconstruct the complete head orientation. In the FOVE HMD website, it is claimed that the eye tracking system has a sampling rate of 140 Hz - 180 Hz with an accuracy of 1° and that the head position is updated at a rate of 100 Hz. Taking into account the processing time of the C++ API, the conversion of the coordinates and the hardware capabilities, the data is finally saved with a mean timestep of $\overline{\Delta}_t = 8.75$ ms (mean sampling rate: 114.3 Hz).

4.2.2 Preprocessing

Filtering

To reduce the effect of noise in the signal, the input time series is smoothed, giving us a fair approximation of the noise-filtered, original series. We try applying the median filter, a nonlinear filter used for signal smoothing to the input of our time series to achieve better predictions. The median filter is particularly good for removing impulsive type noise from a signal.

$$y(n) = \text{med}[x(n-k), x(n-k+1), ..., x(n), ..., x(n+k-1), x(n+k)] \tag{4.1}$$

The filter collects a window containing $N = 2k + 1$ samples of the input signal and then performs the median operation on this set of samples. We set the filter window N to 25. The dataset contains some clear outliers, probably caused by glitches in the FOVE HMD gaze and position tracking system. They cause sudden jumps, which we remove by replacing the value with the average between the two neighbouring data points.

Interpolation

The intervals between two samples were slightly random due to the variable processing time of the tracking system. The time step Δ_t between two consecutive data points of the time series has a mean $\mu(\Delta_t) = 8.75\,\text{ms}$, with a standard deviation of $\sigma(\Delta_t) = 2.88\,\text{ms}$. To transform our data to get an evenly time spaced series we use Python's *scipy* module. We interpolate the time series using linear interpolation and apply it to a new time array $t_{new} = [t_0, t_{0+\hat{\Delta}_t}, ..., t_n]$, with a fixed $\hat{\Delta}_t = 10\,\text{ms}$, getting a constant sample rate of 100 Hz.

Coordinate conversion

The data points for head and gaze position are saved in equirectangular pixels. When visualizing a 3D video with a HMD, the user is situated in the middle of a sphere surrounded by the video. The videos of the dataset are formatted in an equirectangular, monoscopic projection. This projection maps the 3D sphere into a 2D surface (see Figure 4.5). In this type of projection all verticals remain vertical, and the horizon becomes a straight line across the middle of the image. The poles are located at the top and bottom edge and are stretched to the entire width of the image. While processing such a video, the video player does the inverse transformation to reconstruct the 360° image. Figure 4.6 shows that the origin of the coordinate system is situated on the top left of the image. The videos have a 4K resolution, the video width of all videos is $width_{eq} = 3840\,\text{px}$ and the height varies, $height_{eq} \in \{1920\,\text{px}, 2048\,\text{px}, 2160\,\text{px}\}$.

We want to convert the equirectangular pixels to a Cartesian unit vector $\vec{v}_t$, which indicates the direction of the user's viewpoint center. The motivation behind this conversion is to avoid jumps at the limits of the interval of allowed values, which could complicate the prediction. In pixels, the width coordinate is $x_{eq,t} \in [0, width_{eq}]$, while the interval limits

Figure 4.5: Equirectangular image extracted from a frame of a 360° video of the dataset, published to YouTube by [Loo16]. In this type of projection all verticals remain vertical, and the horizon becomes a straight line across the middle of the image. The poles are located at the top and bottom edge and are stretched to the entire width of the image.

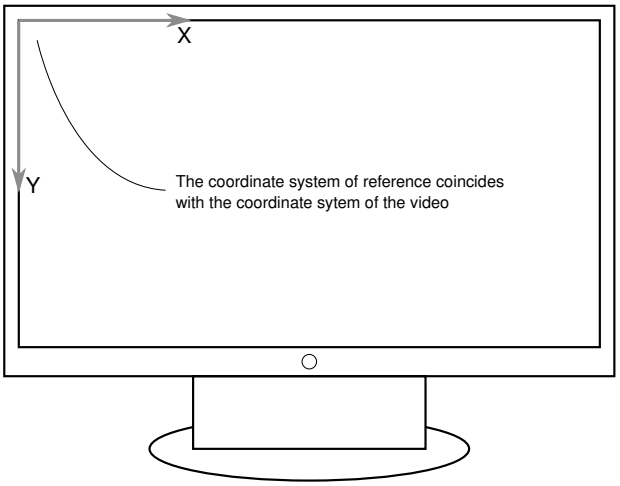


Figure 4.6: Visual representation of equirectangular coordinate system, which coincides with the pixels of the image [AD19]. The origin is situated at the top left and x_eq increases until the right border of the image.

actually describe neighboring pixels. With a unit vector $\vec{v}_t$ we effectively solve this issue by generating data points which are easier to compare. As a first step to calculate the Cartesian coordinates, we convert the equirectangular pixels to Euler angles in radians. The pitch angle represents the horizontal angle in the interval $\phi_t \in [0, 2\pi]$ and yaw the vertical angle, $\theta_t \in [0, \pi]$. The transformation is performed according to the equations below:

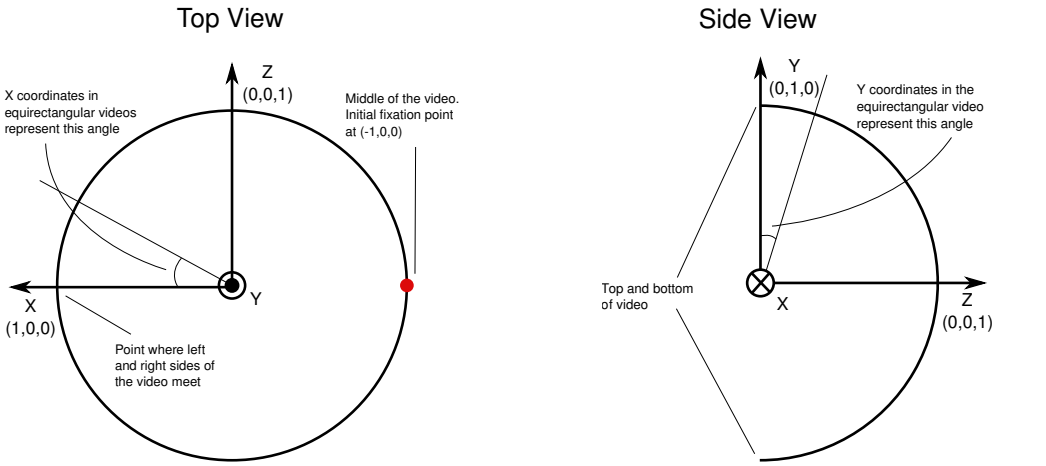
$$\begin{aligned} \phi_t &= x_{eq,t} \cdot 2\pi / width_{eq} \\ \theta_t &= y_{eq,t} \cdot \pi / height_{eq} \end{aligned} \tag{4.2}$$

Once we have the pitch ϕ_t and yaw θ_t angles, we apply the following transformation to generate a Cartesian unit vector $\vec{v}_t = [x_t, y_t, z_t]$:

$$\begin{aligned} x_t &= \sin(\theta_t) \cdot \cos(\phi_t) \\ y_t &= \cos(\theta_t) \\ z_t &= \sin(\theta_t) \cdot \sin(\phi_t) \end{aligned} \tag{4.3}$$

The vector $\vec{v}_t = [x_t, y_t, z_t]$ represents a unit vector from the center of the sphere indicating the direction of the user's viewpoint center. We use the Cartesian coordinate system to model the head position. The values of the three components of the vector are in the interval $I = [-1, 1]$, while $\sqrt{x_t^2 + y_t^2 + z_t^2} = 1$.

Figure 4.7 [AD19] shows how the Cartesian vector $\vec{v}_t$ can be interpreted. The middle of the video, where the user's head is positioned at the start of each video, corresponds to the point $\vec{v}_t = [-1, 0, 0]$ and the top of the image is situated at $\vec{v}_t = [0, 1, 0]$.



(a) Top view of 3D Cartesian coordinate system. The circle represents the intersection of the sphere with the x-z-plane.

(b) Side view of the 3D Cartesian coordinate system. The angle between $\vec{v}_t$ and the y-axis is visualized through a half circle.

Figure 4.7: Cartesian coordinates illustrations, from [AD19]

4.2.3 Time Series Analysis

Our dataset consists of data points recorded while the experiment subject watched 360° videos on a VR headset. Thus, the dataset consists of multiple time series, one for each video. As mentioned above, the head orientation coordinates have been transformed to be a Cartesian unit vector. They can be interpreted as 3 time series taking place in parallel ($\vec{v}_t = [x_t, y_t, z_t]$). A time series is defined as a sequence taken at successive, equally spaced points in time. After having applied the interpolation process mentioned above, the dataset effectively consists of data points with a constant time step $\hat{\Delta}_t = 10\,\text{ms}$. Time

series analysis involves comprehending important aspects about the inherent nature of the series so that meaningful and accurate forecasts can be made.

Stationarity

Stationarity is a very important property of a time series, especially when assessing how well future values can be predicted. Most statistical forecasting methods are based on the assumption that the time series can be rendered approximately stationary using a mathematical transformation. Stationarity is given when the statistical properties (like mean, variance and autocorrelation) of a time series are constant over time. Thus, time series with trends or with seasonality are not stationary, as the trend and seasonality will affect the properties of the time series at different times.

A popular transformation to make a time series stationary is computing the first order difference:

$$\Delta\vec{v}_t = \vec{v}_t - \vec{v}_{t-\Delta t} = [\Delta x_t, \Delta y_t, \Delta z_t] \tag{4.4}$$

Stationarizing a time series through differencing is an important part of the process of fitting an Autoregressive Integrated Moving Average (ARIMA) model, which is often used in economics and statistics to better understand the data or to predict future points of a time series.

A method to quantitatively determine if a given series is stationary is the use of statistical tests called *Unit Root Tests*. The most commonly used implementation is the Augmented Dickey Fuller (ADF) test. It tests the null hypothesis that a unit root is present in a time series sample. A linear stochastic process has a unit root if 1 is a root of the characteristic equation of the function. If a unit root is present, such a process is non-stationary. When conducting the ADF test, a p-value is calculated to support or reject the null hypothesis. p-values are expressed as decimals and can be interpreted as a percentage. We set the alpha-value to 0.05. To reject the null hypothesis, the p-value has to be smaller than our chosen alpha-value.

Table 4.1: Augmented Dickey Fuller (ADF) test results

Time Series	p-Value
x_t	0.079566
y_t	0.019260
z_t	0.090958
Δx_t	0.000666
Δy_t	0.000003
Δz_t	0.000178

We run the ADF test both on the original time series $\vec{v}_t$ and its first difference $\Delta\vec{v}_t$. We see that for the coordinates x_t and y_t the p-value is larger than the alpha-value = 0.05,

indicating that the null hypothesis cannot be completely discarded. For the differenced coordinates, however, the p-value is largely under our alpha-value. This is strong evidence to assume that $\Delta\vec{v}_t$ is indeed stationary.

Autocorrelation

Stationarity is an important assumption for time series forecasting models. However, white noise is also stationary, since it has the same mean and variance at any point, but future values are by definition impossible to predict. That is why we will analyze the autocorrelation of our time series. Autocorrelation is the correlation of a series with its own lags. If a series is significantly autocorrelated, it means that the previous values of the series are similar to future values and may thus be helpful to predict them.

In Figure 4.8, we show the autocorrelation of the three coordinates of the Cartesian vector $\Delta\vec{v}_t$ indicating the direction to the middle of the user FoV for a delay $\tau \in [0.1\,\text{s}, 0.2\,\text{s}, ..., 1\,\text{s}]$. As it can be observed in the plot, the autocorrelation is especially strong for little delays and decreases monotonously. Based on these results in Figure 4.9, we can see that in a short period of time, $\tau < 0.6\,\text{s}$, viewer motion, that is the differenced time series, is predictable. After that the autocorrelation gets very close to zero, which makes the prediction task more difficult.

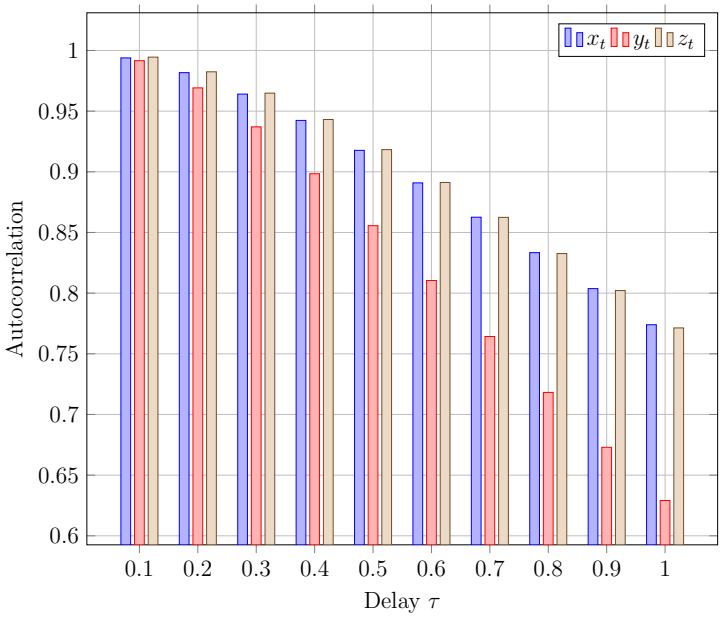


Figure 4.8: Autocorrelation of the time series for the Cartesian coordinates $\vec{v}_t = [x_t, y_t, z_t]$ for delays ranging from 0.1 s to 1 s. The strong autocorrelation is due to the trivial fact that the head position of the user will be similar for short time delays.

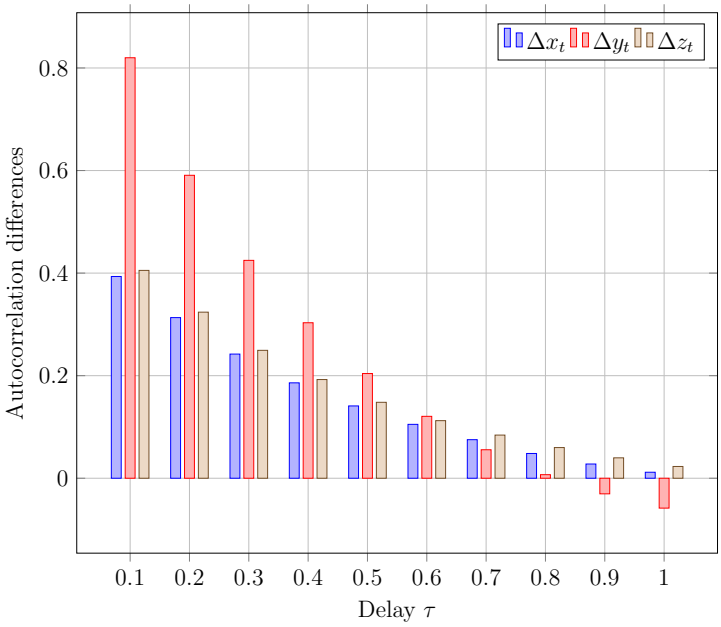


Figure 4.9: Autocorrelation of the time series $\Delta\vec{v}_t = [\Delta x_t, \Delta y_t, \Delta z_t]$ after applying first order differencing. In a short period of time ($\tau < 0.6\,\mathrm{s}$), viewer motion is predictable.

4.2.4 Performance Measurement

Four metrics are used in order to evaluate the accuracy of the predictions. Since our model predicts a head movement position, consisting of 3 continuous values in the interval $x_{head}, y_{head}, z_{head} \in [-1, 1]$, it is a regression problem.
The first metric, used as loss function during training, is the Mean Absolute Error (MAE). It is is calculated as an average of absolute differences between the target values and the predictions:

$$\mathrm{MAE}(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{4.5}$$

The MAE measures the average magnitude of errors in a set of predictions, without considering their directions.

The second metric used is the Root Mean Squared Error (RMSE), the square root of the most simple and common metric for regression evaluation, the Mean Squared Error (MSE). The MSE is the sum of squared distances between target and predicted values:

$$\mathrm{RMSE}(y, \hat{y}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} = \sqrt{\mathrm{MSE}} \tag{4.6}$$

The Symmetric Mean Absolute Percentage Error (SMAPE) measures the error in percentage terms, ranging from 0 to 100%. It is more robust towards outliers and is invariant

to linear scaling of the data. In the Web Traffic Time Series Forecasting Competition in Kaggle [GI17], this metric is used to evaluate the proposed forecasts.

$$\text{SMAPE}(y, \hat{y}) = \frac{100}{N} \sum_{i=1}^{N} \frac{|y_i - \hat{y}_i|}{|y_i| + |\hat{y}_i|} \tag{4.7}$$

Finally, the Mean Forecast Error (MFE) is computed to analyze the tendency of the model to over forecast (negative error) or under forecast (positive error). The ideal MFE is zero and it can be interpreted as a type of forecast bias. It is calculated as the average of the forecast error values:

$$\text{MFE}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i) \tag{4.8}$$

4.2.5 Training

The training dataset consists of the recorded data of 40 users, 9 are reserved for testing purposes. The test subset contains the data recorded for 10 users that do not appear in the training set to evaluate the generalization ability of the network. 20% of the testing data is used as validation set to tune the model's hyperparameters. An early stopping callback is called while fitting the model using the Keras library as regularization. The maximum number of epochs is set to 150, although the training automatically stops if the loss function of the validation set does not decrease during 15 consecutive epochs (`patience` = 15, `min_delta` = 0). The model checkpoint makes sure that only the best model (the weights allowing the proposed model to achieve the lowest loss on the validation set) is saved. These two callback functions are used to avoid overfitting and to find the optimal weights and amount of epochs for each model architecture. TensorBoard is used as visualization tool during training to plot the 3 metrics (MAE, RMSE and SMAPE).

As mentioned above, the MAE is used as loss function. Compared to the RMSE, it is more robust to outliers and easier to interpret. Instead of the classical Stochastic Gradient Descent (SGD) to update network weights iteratively based on the training data, the more modern Adaptive Moment Estimation (Adam) optimization algorithm is used. It was presented by Kingma et al. in 2014 [KB14]. The authors describe Adam as combining the advantages of both Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSprop). It computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients [KB14]. The attractive benefits of using Adam are its efficiency, little memory requirements and its superior effectiveness compared to other stochastic optimization problems. In Keras, the default configuration parameters are those recommended by [KB14] and they have shown to do well on most problems.

To train our models, we used the sliding window technique. Given the time series, we restructure the data to be used as a supervised learning problem. We do this by using

previous time steps as input variables and the next time steps as ground truth. We define a lookback time interval $T_W = 250\,\text{ms}$ and a prediction window $T_P = \max(\tau) = 1\,\text{s}$.

In Figure 4.10 the data restructuring is illustrated. The lookback window consists of the last 25 values, representing the head positions in the interval $[t - T_W, t]$. The output consists of 10 values, representing real head positions for the delay $\tau = [0.1\,\text{s}, 0.2\,\text{s}, ..., 1\,\text{s}]$. For each sample, the window is moved by one time step $\hat{\Delta}_t = 10\,\text{ms}$.

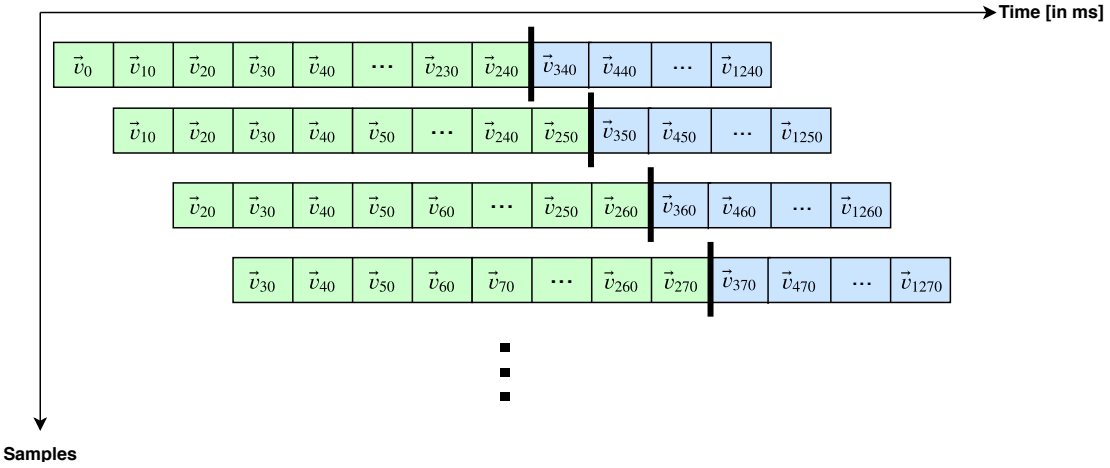


Figure 4.10: Visual representation of training input and output using the sliding window technique. For every sample, the input to the network is shifted by one time step ($\hat{\Delta}_t = 10\,\text{ms}$)

4.3 Evaluated Head Motion Prediction Models

In this section, we will present the different models that were examined in order to solve the Head Movement Prediction task. Let $\vec{v}_{t_1:t_2}$ be the head motion unit vectors from time t_1 to t_2, i.e. $\vec{v}_{t_1:t_2} = [\vec{v}_{t_1}, \vec{v}_{t_1+\hat{\Delta}_t}, ..., \vec{v}_{t_2}]$. Our problem can be phrased as a supervised learning multi-step regression problem, where the optimal function $f(\cdot)$ has to be determined to compute

$$\hat{\vec{v}}_{t+0.1\,\text{s}:t+1\,\text{s}} = f(\vec{v}_{t-T_W:t}). \tag{4.9}$$

The lookback window is set to $T_W = 250\,\text{ms}$ and the prediction window $T_P = 1\,\text{s}$.

4.3.1 Baseline Models

To evaluate the performance of our proposed Seq2Seq model, we will compare them to two standard methods which do not use neural networks. As stated before, we use 4 different metrics to measure performance (MAE, RMSE, SMAPE and MFE). As a naive approach, we predict that the head orientation remains constant, that is, we predict that the future head orientation vectors $\hat{\vec{v}}_{t+0.1\,\text{s}}$, $\hat{\vec{v}}_{t+0.2\,\text{s}}$, ... $\hat{\vec{v}}_{t+1\,\text{s}}$ are equal to the current one, $\vec{v}_t$.

As a second approach, we use Linear Regression (LR) to predict the future time series. We use *statsmodels*, a Python module that provides classes and functions for the estimation of different statistical models, including Linear Regression. We create three independent linear models, one for each Cartesian coordinate.

As a state-of-the-art comparison model, we use a neural network consisting of interleaved LSTMs and dense FFNN layers (see Figure 4.11). This model showed very good performance in [ABK+18], with a slight difference in the input data, as the coordinates were computed in yaw θ_t, pitch Φ_t and roll ψ_t angles instead of Cartesian unit vectors as in our work. We feed the model with a sequence $\vec{v}_{t-T_W:t}$ of Cartesian coordinates of past head orientations. After computing the normalized differences, there is a convolution layer with `filters` $= 10$ and `kernel_size` $= 10$ as a sort of low-pass filter to reduce the noise. The output is concatenated and serves as input for both an LSTM and a FFNN layer. The output of both layers is merged with their input and passed through a dense FFNN with $units = 9$. This process is repeated 5 consecutive times. The output of the last LSTM layer is passed through a final dense FFNN layer with $units = 30$ and finally remapped to generate the future head position vectors $\hat{\vec{v}}_{t+0.1\,\mathrm{s}:t+1\,\mathrm{s}}$

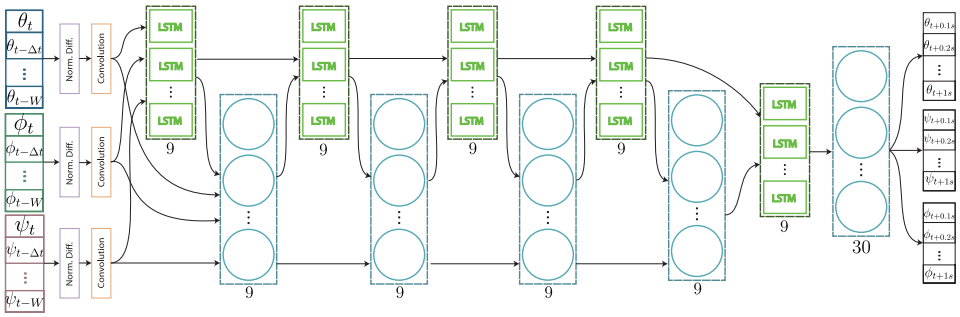


Figure 4.11: Neural network architecture proposed in [ABK+18] consisting of interleaved FFNN and LSTM layers. Although originally used with yaw θ_t, pitch Φ_t and roll ψ_t angles as input, we train the model with the Cartesian coordinates x_t, y_t and z_t.

4.3.2 Encoder-Decoder Seq2Seq Models

To implement the Encoder-Decoder Sequence-to-Sequence (Seq2Seq) model, we use Python's high-level neural networks API called Keras with Tensorflow running on the backend. Encoder and decoder consist of CuDNNLSTM layers, the LSTM implementation, backed by NVIDIA's CUDA Deep Neural Network library (cuDNN). Using this GPU-accelerated library, the networks can be trained considerably faster. Using this LSTM layers instead of other types of RNNs, we hope to achieve the best possible performance. When constructing the model architecture, we have two important hyperparameters to choose. On the one hand, we can vary the depth of the neural network. Stacking LSTM hidden layers makes the model deeper, adding levels of abstraction of input observations

over time. We will analyze different depths to verify if deeper, stacked LSTMs (see Figure 4.12) can improve performance. Thus, we train encoder and decoder with 1, 3 and 5 stacked LSTM layers. Another important hyperparameter is the dimensionality of the output space, that is, the dimensionality of the matrices $\boldsymbol{W}$ and $\boldsymbol{U}$. We vary the size of the output vector $d \in \{32, 64, 128\}$. The final hidden state, connecting the encoder and decoder, has a size of $n \times d$.

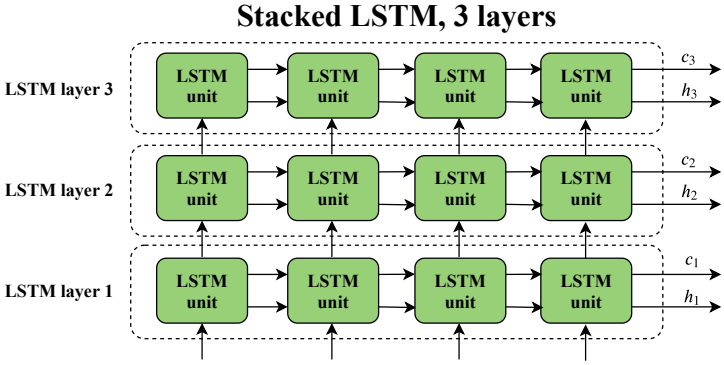


Figure 4.12: Stacked LSTM, consisting of 4 time steps and 3 layers. Stacking LSTM hidden layers makes the model deeper, adding levels of abstraction of input observations over time.

For the encoder, we maintain a constant input size $n \times 3$. As described before, we have a lookback window size $T_W = 250\,\text{ms}$ and, after linear interpolation, the data points have a constant time step length of $\hat{\Delta}_t = 10\,\text{ms}$. Thus, the encoder input size can be calculated as $n = \frac{T_W}{\hat{\Delta}_t} = 25$. The second dimension corresponds to the 3 coordinates of the Cartesian unit vector.

Before feeding $\vec{v}_t$ into the first LSTM layer, we apply normalized differencing:

$$\begin{aligned} \Delta\vec{v}_t &= \vec{v}_t - \vec{v}_{t-\hat{\Delta}_t} \\ \Delta\vec{v}_{norm,t} &= \frac{\Delta\vec{v}_t}{\max(|\Delta\vec{v}_t|)} \end{aligned} \tag{4.10}$$

As we actually predict the future normalized differences, in the end we remap to the absolute future orientation values as follows:

$$\hat{\vec{v}}_{t+n\cdot 0.1\,\text{s}} = \vec{v}_t + \sum_{k=t}^{t+n} \Delta\hat{\vec{v}}_{norm,k:k+0.1\,\text{s}} \cdot \max(|\Delta\vec{v}_t|), \tag{4.11}$$

where $n \in [1, 2, ..., 10]$. Note that we define $\Delta\vec{v}_{t_1:t_2} = \vec{v}_{t_2} - \vec{v}_{t_1}$.

Regarding the decoder input, we analyze different approaches. First of all, we set a constant decoder input of zeros, with a size of 10×3. Given that the weights in every LSTM unit

cell are equal, we expect the network's weights to optimize without the need of additional information from the decoder's input gate. We assume that the LSTM's hidden state h_t and cell state c_t are enough to propagate the information about the past from the encoder to the decoder to make good predictions. This reduces complexity and can potentially accelerate the training.

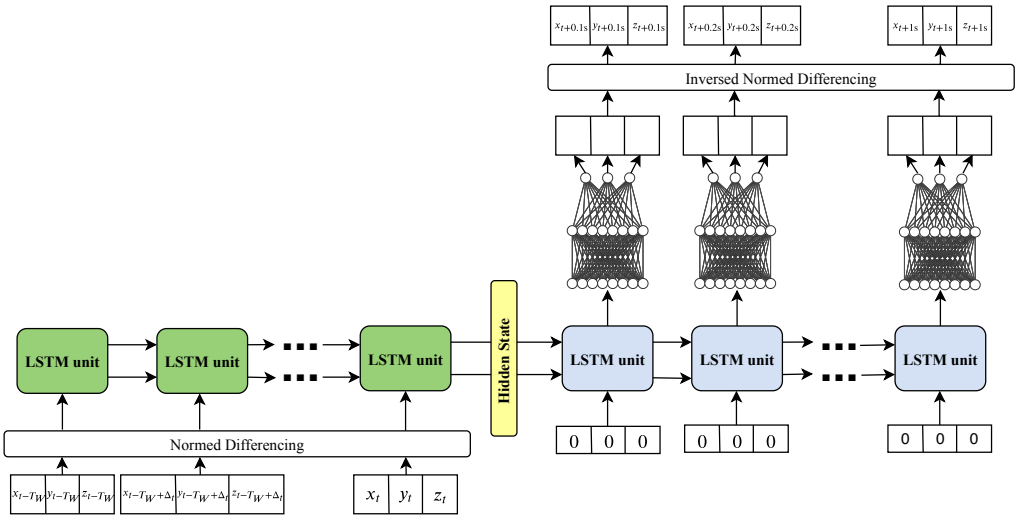


Figure 4.13: Implementation of an Encoder-Decoder Seq2Seq model. The decoder input is set to zero. The decoder is responsible for making predictions for future head positions using the hidden state h_t and the cell state c_t as initial states.

Another approach we test is to reinject the decoder's output for every LSTM unit step. The output of the previous time step can be valuable for the current unit. As every unit of the decoder generates a prediction of $\Delta\hat{\vec{v}}_{t:t+0.1\,\mathrm{s}}$ for a prediction time step of 100 ms, the next unit cell can use this data to generate a prediction for the next 100 ms. The first input of the decoder is $\Delta\vec{v}_{t-0.1\,\mathrm{s}:t}$. A possible problem of this approach is error propagation. If at one step the network's decoder generates an inaccurate prediction, the error propagates and affects future predictions. To train this model we use two different methods. On the one hand we train it using Teacher Forcing, where during training the true series $[\Delta\vec{v}_{t:t+0.1\,\mathrm{s}}, \Delta\vec{v}_{t+0.1\,\mathrm{s}:t+0.2\,\mathrm{s}}, ..., \Delta\vec{v}_{t+0.9\,\mathrm{s}:t+1\,\mathrm{s}}]$ is fed to the decoder. Intuitively, we are trying to teach the network how to condition on previous time steps to predict the next one. At inference time, the true values are replaced by the predicted values for each time step. Additionally, we also train the model reinjecting the predictions during training as well as inference mode.

Finally, we test the efficacy of adding an attention layer to surpass the information bottleneck that represents the context vector. We use the Keras layer implementation provided by [Gan19]. It implements the global attentional model proposed by Bahdanau et al.,

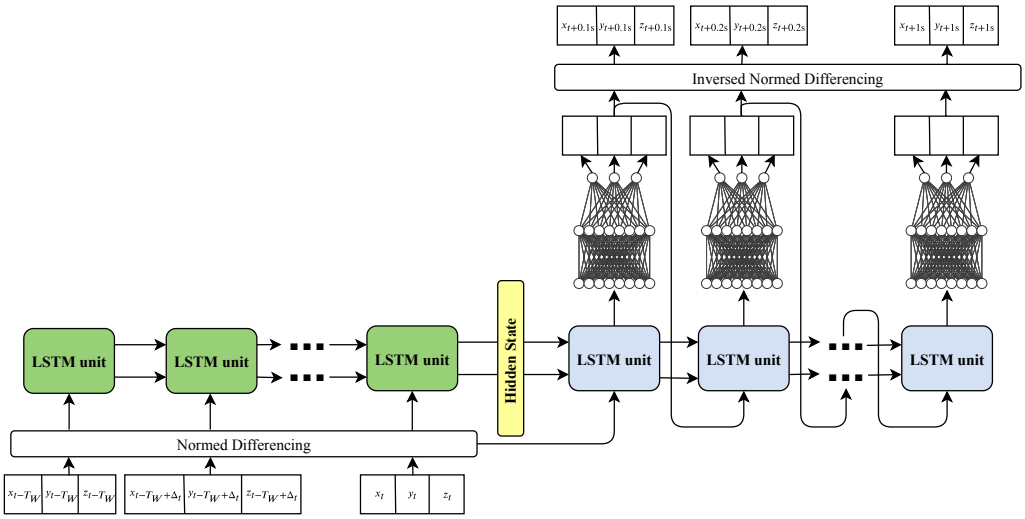


Figure 4.14: Visual representation of the Seq2Seq model. Only the input for the first LSTM unit is added externally. a vector of zeros as decoder input at every time step. The decoder's output is reinjected to the subsequent LSTM unit after passing two dense FFNN layers.

where the context vector becomes a weighted sum of all past encoder states. The weights are optimized during training. For the decoder we again use a vector of zeros as input for every time step.

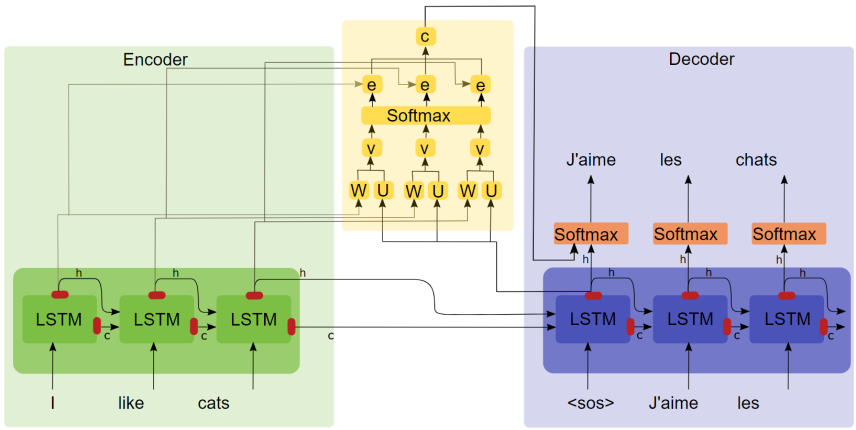


Figure 4.15: Illustration of an Encoder-Decoder model with an attention layer as context vector. The context vector becomes a weighted sum of all past encoder states.

4.4 Head Motion Prediction with Saliency Decoder Module

In order to test the efficacy of using saliency maps directly as input to the Encoder-Decoder Seq2Seq network architecture, we feed the encoder with the head orientation history $[\vec{v}_{t-T_W}, \vec{v}_{t-T_W+\hat{\Delta}_t}, ..., \vec{v}_t]$, while the decoder is fed with the saliency map SM_t at time t. Using a single image as input to our model makes it more memory efficient than other approaches at the cost of the information about temporal saliency. Nevertheless, the last frame visualized by the user is the most important one for the Head Movement Prediction task.

The saliency map is generated using the state-of-the-art model for saliency prediction Deep Gaze II. It achieves top performance in area under the curve (AUC) and is currently at the top position on the MIT Saliency Benchmark. Our dataset consists of 360° videos at 4K resolution and formatted in an equirectangular projection (monoscopic). However, the user can only see his current Field of View (FoV), which depends on his current head position. Thus, for every video visualized by every user, we generate the 2D perspective image of the FoV from the equirectangular panorama frame using the python tool called `Equirec2Perspec` [FE17]. The FOVE VR HMD has 100° of horizontal FoV. We use the tool to generate perspective RGB images of the FoV with a resolution of 512×288. We apply the DeepGaze II network to this image to get the FoV saliency map. We then rescale the image to the final 128×72. We apply a Gaussian blur with $\sigma = 5$ to regularize the predictions. The resolution of 128×72 keeps the standard aspect ratio of $16 : 9$ seen in HMDs like the FOVE VR, which has a resolution of 2560×1440. The videos have a frame rate of 30 fps, while the head motion data from the dataset is available at a rate of 100 Hz after the preprocessing. Thus, the head position is computed as the average of all the head positions recorded at each specific video frame.

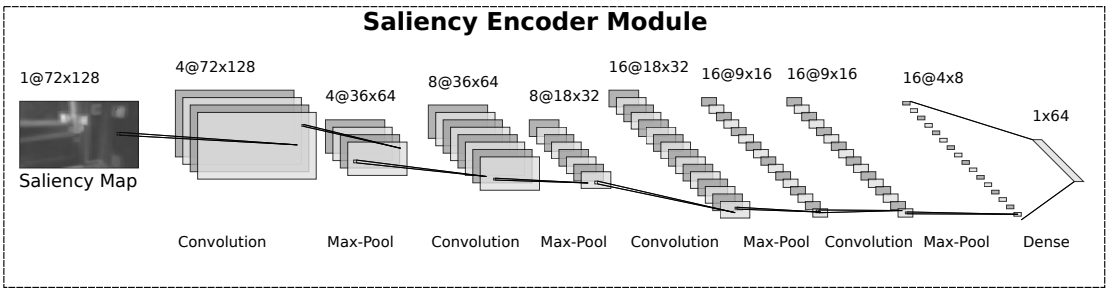


Figure 4.16: Proposed saliency encoder module responsible for extracting the features from the saliency map and downsampling the image. After applying a dense FFNN layer, the resulting vector is used as input for the first decoder time step.

We test a saliency encoder module consisting of 4 sequential CNN layers (see Figure 4.16, each of them followed by a Max-Pooling layer to iteratively downsample the image while the most important features are extracted. The convolutional layers have a kernel size of

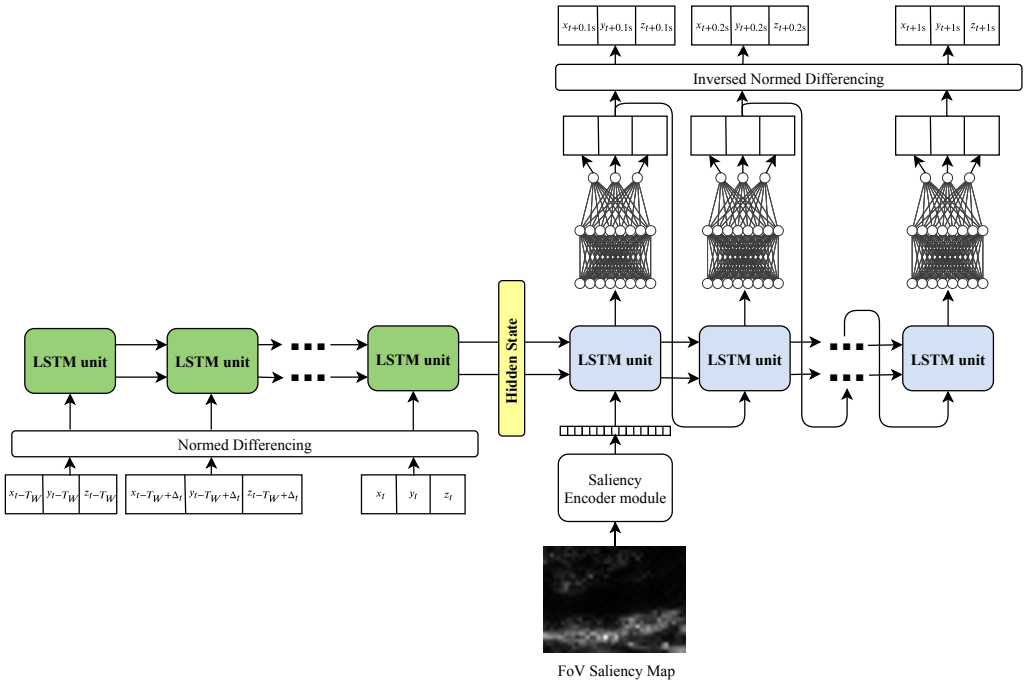


Figure 4.17: Encoder-Decoder Seq2Seq model using sensor- and content-based data as input. The saliency map of the saliency map at time t is downsampled in the proposed saliency encoder module and used as input for the first LSTM unit.

3×3 and $\{4, 8, 16, 16\}$ kernels in total. After each convoluational layer, the ReLu activation function is computed. The Max-Pooling layers have a pool size of (2, 2), strides of (2, 2) and we set `padding` = 'same'. The resulting feature maps are finally flattened and a dense feed-forward layer is applied. The saliency encoder output is fed to the first time step of the decoder of the Seq2Seq model (see Figure 4.17). The subsequent LSTM units are fed with the output of the previous unit, after passing through two dense FFNN layers.

4.5 Environment

4.5.1 Host Computer

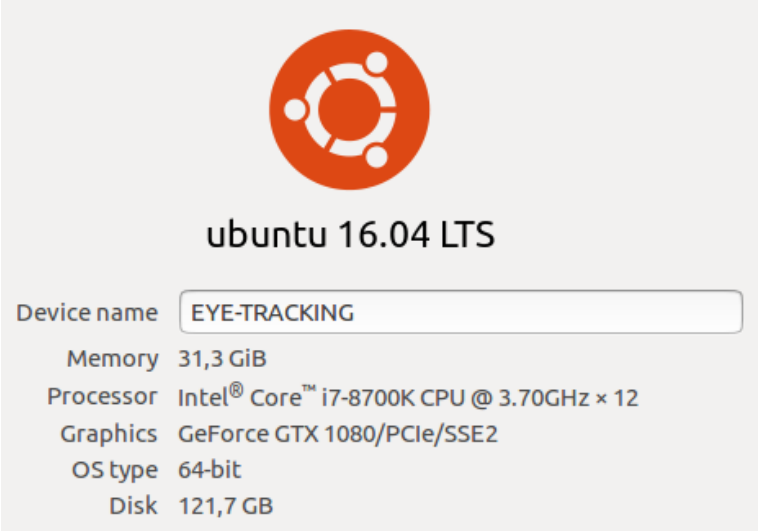


Figure 4.18: Host computer, used to train the models.

4.5.2 Modules and Libraries

- Python v3.5.2
- Keras v2.2.4
- Tensorflow GPU v1.8.0
- CUDA v9.0.176
- NVIDIA (R) CUDA Toolkit (release 7.5, v7.5.17)
- NVIDIA Driver v384.130
- cuDNN v7.6.1
- NumPy v1.16.4
- Tool to project equirectangular into perspective images [FE17]
- Keras Layer implementation of Attention [Gan19]

Chapter 5

Discussion

In this chapter, we are going to present and analyze the results of our experiments and evaluate them thereafter. As already mentioned, our main goal was to analyze if an Encoder-Decoder Sequence-to-Sequence model is fit for the Head Movement Prediction task. For that purpose, we tested different approaches and trained several model architectures to find the hyperparameters that lead to the best performance and finally compares them to two baseline models and a state-of-the-art neural network.

5.1 Hyperparameter Tuning

Hyperparameters are parameters whose values are decided before training and that modify high-level properties of the network such as its complexity or topology. We use grid search, the traditional approach to perform hyperparameter optimization. The first hyperparameter we analyze is the model's depth. The encoder and decoder are based on LSTM layers. Adding depth to neural networks has attributed to the success of Deep Learning in recent years. Given that LSTMs operate on sequential data, the addition of layers adds levels of abstraction to the input observations over time. A stacked LSTM can be defined as several, sequential LSTM layers, where the hidden layers provide the full sequence of outputs to the subsequent layer. We vary encoder and decoder with $\{1, 3, 5\}$ stacked LSTM layers. Additionally, we train the model for `units` $d = \{32, 64, 128\}$. The unit is the dimension of the inner cells of the LSTM. By increasing the dimension, we increase the number of parameters and thus the model complexity. We used MAE as our loss function as primary metric to compare the results.

To analyze the impact of the decoder input on the model output, we first train the Sequence-to-Sequence (Seq2Seq) network with the past sensor data $\vec{v}_{t-T_W:t}$ as encoder input and a vector of zeros as input for each time step of the decoder. Table 5.1 shows the Mean Absolute Error (MAE) for each time step, applied to the validation subset of our training data set. Results show how the error increases monotonously through time,

which is caused by the low autocorrelation at higher time steps and the accumulation of the error caused by previous predictions. Results are generally very close between different experiments. Models with bigger LSTM dimensions seem to perform better, while stacking LSTM layers does not seem to increase accuracy. The model with 1 LSTM layer with a dimension of 128 makes the best predictions for the first time step, while the model with 5 LSTM layers and 64 neurons has the best average MAE.

Table 5.1: **MAE** of the Encoder-Decoder Seq2Seq head motion predictions for different delays τ. The decoder input consists of a vector of zeros for every decoder time unit.

Decoder zeros \ Delay τ	0.1 s	0.2 s	0.3 s	0.4 s	0.5 s	0.6 s	0.7 s	0.8 s	0.9 s	1 s	Average
Seq2Seq [1-32]	0.00515	0.01403	0.02546	0.03813	0.05143	0.06506	0.07881	0.09255	0.10615	0.11955	0.05963
Seq2Seq [1-64]	0.00518	0.01405	0.02541	0.03798	0.05114	0.06463	0.07826	0.09189	0.10542	0.11875	0.05927
Seq2Seq [1-128]	0.00471	0.01347	0.02485	0.03751	0.05077	0.06436	0.07809	0.09181	0.10542	0.11882	0.05898
Seq2Seq [3-32]	0.00522	0.01396	0.02521	0.03769	0.05078	0.06422	0.07781	0.09142	0.10494	0.11827	0.05895
Seq2Seq [3-64]	0.00504	0.01362	0.02477	0.03722	0.0503	0.06376	0.07738	0.09103	0.10455	0.11788	0.05856
Seq2Seq [3-128]	0.00482	0.01325	0.02432	0.03675	0.04984	0.0633	0.07692	0.09055	0.10405	0.11736	0.05812
Seq2Seq [5-32]	0.00519	0.01394	0.02519	0.0377	0.05084	0.06432	0.07794	0.09155	0.10504	0.11834	0.059
Seq2Seq [5-64]	0.0049	0.0133	0.02426	0.03655	0.04954	0.06294	0.0765	0.09007	0.10349	0.11672	0.05783
Seq2Seq [5-128]	0.00578	0.01465	0.02581	0.03818	0.05113	0.06441	0.07784	0.091301	0.10469	0.11791	0.05917

In another model, we feed the first LSTM cell with $\Delta\vec{v}_{t-0.1\,\mathrm{s}:t}$ and iteratively reuse the output of each LSTM cell as input for the subsequent one, instead of having a constant value for the decoder inputs. Results show to be persistently better using this approach, especially for the prediction of the head motion at a low delay τ. Again, the model consisting of 5 LSTM layers with a dimension of 64 achieves the best performance.

Table 5.2: **MAE** of the Encoder-Decoder Seq2Seq head motion predictions for different delays τ. The decoder input for the first time step is $\Delta\vec{v}_{t-0.1\,\mathrm{s}:t}$ and each output is reused as input for the subsequent time step.

Reinject Output \ Delay τ	0.1 s	0.2 s	0.3 s	0.4 s	0.5 s	0.6 s	0.7 s	0.8 s	0.9 s	1 s	Average
Seq2Seq [1-32]	0.00479	0.01355	0.02488	0.03751	0.05079	0.06442	0.0782	0.09195	0.10555	0.11892	0.05906
Seq2Seq [1-64]	0.00473	0.01348	0.02481	0.03742	0.05065	0.06422	0.07793	0.09164	0.10523	0.1186	0.05887
Seq2Seq [1-128]	0.00495	0.01368	0.02494	0.03746	0.05063	0.06415	0.07785	0.09154	0.1051	0.11845	0.05887
Seq2Seq [3-32]	0.00548	0.01436	0.02573	0.03831	0.05145	0.0649	0.07847	0.09204	0.10548	0.11873	0.0595
Seq2Seq [3-64]	0.0045	0.01292	0.02407	0.03658	0.04977	0.06332	0.07703	0.09075	0.10435	0.11778	0.0581
Seq2Seq [3-128]	0.00469	0.01311	0.02418	0.03657	0.0496	0.06301	0.07658	0.09017	0.10363	0.11691	0.05785
Seq2Seq [5-32]	0.00534	0.01403	0.02517	0.03757	0.05062	0.06402	0.07757	0.09112	0.10457	0.11784	0.05879
Seq2Seq [5-64]	0.00467	0.01298	0.02396	0.03631	0.04935	0.06278	0.0764	0.09003	0.10352	0.11682	0.05768
Seq2Seq [5-128]	0.00487	0.01333	0.02442	0.03685	0.04992	0.06336	0.07697	0.09059	0.10411	0.11744	0.05819

In the following, we examine two methods that aim to improve the prediction performance and training time.

We test the efficacy of using Teacher Forcing (TF) as training method. At training time, the ground truth is fed to the decoder to teach the network how to condition on previous time steps to make predictions, while at inference time each LSTM cell output is reused as input at the following time unit. It is interesting to note that the predictions for delays

$\tau \in \{0.1\,\text{s}, 0.2\,\text{s}, 0.3\,\text{s}\}$ are very reliable, showing the best results for 1 stacked LSTM layer and 64 neurons (see Table 5.3). Nevertheless, the error accumulates faster for big delays, making it perform worse than the previous models. We assume this is due to the fact that the model is trained with the ground truth. The gradients of back propagation for big steps may thus be very small and the decoder, based on LSTM units, develops a *short memory*. The accuracy of this approach decreases with the system's depth.

Table 5.3: **MAE** of the Encoder-Decoder Seq2Seq head motion predictions for different delays τ. The decoder input for the first time step is $\Delta\vec{v}_{t-0.1\,\text{s}:t}$ and each output is reused as input for the subsequent time step. The model is trained using the Teacher Forcing method.

Delay τ / Teacher Forcing	0.1 s	0.2 s	0.3 s	0.4 s	0.5 s	0.6 s	0.7 s	0.8 s	0.9 s	1 s	Average
Seq2Seq [1-32]	0.00411	0.01335	0.02554	0.03929	0.05392	0.0691	0.08459	0.10021	0.11583	0.13134	0.06373
Seq2Seq [1-64]	0.00387	0.01274	0.02447	0.0377	0.05176	0.06634	0.08119	0.09616	0.1111	0.12596	0.06113
Seq2Seq [1-128]	0.00384	0.01277	0.02474	0.0383	0.05273	0.06772	0.08304	0.09854	0.11404	0.1295	0.06252
Seq2Seq [3-32]	0.00396	0.01308	0.02522	0.03893	0.05352	0.06868	0.08419	0.09989	0.11561	0.13128	0.06343
Seq2Seq [3-64]	0.00376	0.01256	0.0244	0.03789	0.05234	0.06742	0.08284	0.09844	0.11406	0.1296	0.06233
Seq2Seq [3-128]	0.00389	0.01307	0.0255	0.03975	0.05516	0.07142	0.08834	0.10579	0.12365	0.14185	0.06684
Seq2Seq [5-32]	0.02079	0.04252	0.061	0.09096	0.106	0.1336	0.1523	0.18064	0.19906	0.2378	0.12247
Seq2Seq [5-64]	0.01686	0.03525	0.05224	0.06897	0.08623	0.10181	0.11898	0.1341	0.15057	0.16487	0.09299
Seq2Seq [5-128]	0.01614	0.03144	0.05006	0.06651	0.08394	0.09989	0.11605	0.13099	0.14589	0.15965	0.09006

Finally, we use an attention layer between encoder and decoder to avoid the information bottleneck that may be caused by the context vector, which is responsible for summarizing the information about the past. With the attention layer, the hidden state between encoder and decoder becomes a weighted sum of all the past encoder states, not only the last one. The decoder receives a more extensive representation of the past to make more accurate predictions about the future. This approach with 1 LSTM layer and and 128 neurons shows to be the best performing model on the validation data (see Table 5.4).

Table 5.4: **MAE** of the Encoder-Decoder Seq2Seq head motion predictions for different delays τ. The decoder input consists of a vector of zeros for every decoder time unit. To avoid the bottleneck of the hidden state between encoder and decoder, we add an attention layer. Thus, the context vector becomes a weighted sum of all past encoder states.

Delay τ / Attention	0.1 s	0.2 s	0.3 s	0.4 s	0.5 s	0.6 s	0.7 s	0.8 s	0.9 s	1 s	Average
Seq2Seq [1-32]	0.00431	0.01311	0.02456	0.03733	0.05081	0.0647	0.07873	0.09277	0.10666	0.12028	0.05932
Seq2Seq [1-64]	0.00412	0.01273	0.02396	0.03652	0.04978	0.06342	0.07724	0.09108	0.10478	0.11825	0.05819
Seq2Seq [1-128]	0.00394	0.0123	0.02336	0.03576	0.04887	0.06241	0.07612	0.08985	0.10347	0.11687	0.0573
Seq2Seq [3-32]	0.00434	0.01278	0.02393	0.03642	0.04961	0.06318	0.07689	0.09061	0.1041	0.11759	0.05795
Seq2Seq [3-64]	0.00424	0.01253	0.02356	0.03597	0.04909	0.06262	0.07631	0.09	0.10357	0.11694	0.05748
Seq2Seq [3-128]	0.00439	0.01267	0.02367	0.03604	0.04912	0.06263	0.07632	0.09003	0.10362	0.11701	0.05755
Seq2Seq [5-32]	0.0042	0.01252	0.02358	0.036	0.04912	0.06264	0.07633	0.09004	0.10364	0.11703	0.05751
Seq2Seq [5-64]	0.00432	0.0127	0.0238	0.03628	0.04949	0.06313	0.07693	0.09074	0.1044	0.11783	0.05796
Seq2Seq [5-128]	0.00452	0.01283	0.02382	0.03617	0.04922	0.06268	0.07632	0.08998	0.10352	0.11686	0.05759

With respect to the proposed Sequence-to-Sequence module with integrated Saliency En-

coder, the results (see Table 5.5) show a decrease in performance in comparison to the models relying solely on sensor-related data. The decoder does not seem to take advantage of the downsampled saliency representation generated by the sequential CNNs, which can be due to the complexity of the image. A late fusion approach computing the location of the most salient pixel or the average between peaks might be more reliable as input to the decoder. Nevertheless, the fact that the majority of users did not have prior experience with VR may make them more incline to explore the whole visual scene rather than focusing on salient objects. The behavior of the users may vary after they get used to the new environment and the Head Movement Prediction task based on content-related features may be easier to model. Thus, a bigger dataset with subjects having already had experiences with VR may present better results for our proposed saliency encoder module.

Table 5.5: **MAE** of the Encoder-Decoder Seq2Seq head motion predictions for different delays τ. The saliency map of the FoV at time t is is fed to the saliency encoder module. The resulting vector is the input for the first LSTM time step and each output is reinjected to the subsequent cell.

Saliency Encoder \ Delay τ	0.1 s	0.2 s	0.3 s	0.4 s	0.5 s	0.6 s	0.7 s	0.8 s	0.9 s	1 s	Average
Seq2Seq [1-32]	0.00541	0.01471	0.02652	0.03959	0.05329	0.0673	0.08141	0.09543	0.10918	0.12254	0.06154
Seq2Seq [1-64]	0.0052	0.0154	0.02763	0.04093	0.05471	0.06874	0.08283	0.09682	0.11054	0.12386	0.06267
Seq2Seq [1-128]	0.00517	0.01386	0.02534	0.03811	0.05156	0.06532	0.07918	0.09294	0.10644	0.11955	0.05975
Seq2Seq [3-32]	0.005486	0.01475	0.02646	0.03947	0.05313	0.06709	0.08112	0.09502	0.10864	0.12188	0.06131
Seq2Seq [3-64]	0.00612	0.01524	0.02674	0.03938	0.05252	0.06577	0.07897	0.09205	0.10494	0.11762	0.05993
Seq2Seq [3-128]	0.0048	0.01335	0.02446	0.03697	0.05019	0.06377	0.07747	0.09107	0.10443	0.11741	0.05839
Seq2Seq [5-32]	0.00487	0.01363	0.02499	0.03774	0.05117	0.06494	0.07882	0.0926	0.10616	0.11932	0.05943
Seq2Seq [5-64]	0.00491	0.01345	0.0246	0.03716	0.05044	0.06405	0.07775	0.09139	0.10479	0.11783	0.05863
Seq2Seq [5-128]	0.00473	0.01314	0.02426	0.03682	0.05011	0.06376	0.07751	0.09119	0.10464	0.1177	0.05839

5.2 Evaluation

In this section we compare the performance of the proposed Encoder-Decoder Seq2Seq network to two baseline models and a state-of-the art model, which already showed to be successful in the Head Movement Prediction (HMP) task. We use the 4 metrics described previously (MAE, RMSE, SMAPE and MFE). We use both the Seq2Seq model that showed the lowest average MAE and the one with the lowest error for the first prediction, that is, at a delay of $\tau = 0.1\,\text{s}$. We evaluate all methods not only on their accuracy, but, given that the test dataset contains 9 new users, also on their generalization capabilities.

In Figure 5.1, we illustrate the MAE of the examined architectures for $\tau \in [0.1\,\text{s}, 0.2\,\text{s}, ..., 1\,\text{s}$. The results show that the proposed Seq2Seq using an attention layer between encoder and decoder exhibits the best performance compared to baseline models (Linear Regression and Naive approach) and the state-of-the-art model consisting of interleaved LSTM and FFNN blocks. This confirms the fitness of the encoder-decoder model for the Head Movement Prediction task. The similar slope of all methods for big delays ($> 0.6\,\text{s}$) is caused by the low autocorrelation the differenced time series has. Predicting head motion based solely

on past data points is thus limited to $\sim 0.6\,\mathrm{s}$. This can also be seen in the fact that for longer delays the absolute value of the predicted differences $|\Delta\hat{\vec{v}}_{t+\tau:t+\tau+0.1\,\mathrm{s}}|$ diminishes, as the unpredictability of the head motion increases. This can intuitively be explained by the changes in the visual scene of the user, adding new stimuli that may affect the user's movement.

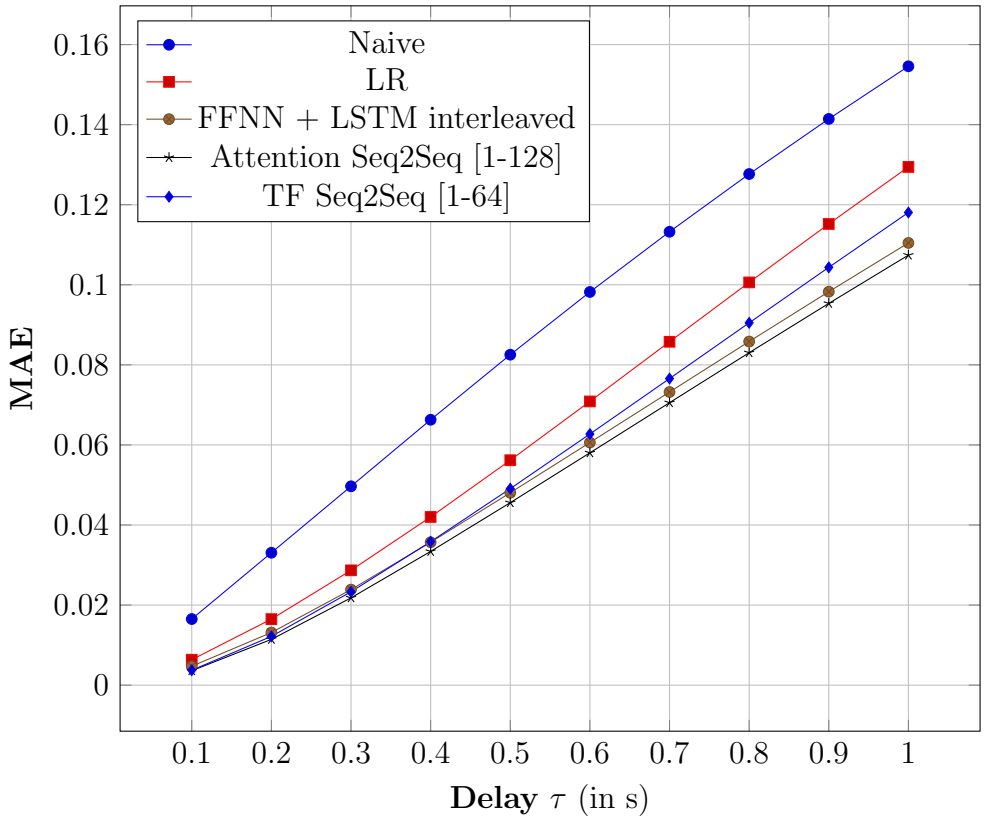


Figure 5.1: Mean Absolute Error (MAE) for delays $\tau \in [0.1\,\mathrm{s}, 0.2\,\mathrm{s}, ..., 1\,\mathrm{s}]$. The Seq2Seq architecture with 1 LSTM layer and 128 neurons leads to the best results.

Figure 5.2 and 5.3 show similar results. They project the metrics RMSE and SMAPE for $\tau \in [0.1\,\mathrm{s}, 0.2\,\mathrm{s}, ..., 1\,\mathrm{s}]$. Although the model trained using Teacher Forcing makes the best prediction for a delay $\tau = 0.1\,\mathrm{s}$, the Attention Seq2Seq model performs best in average. Nevertheless, the improvement it offers compared to the state-of-the-art FFNN+LSTM model is limited.

Figure 5.4 shows the Mean Forecast Error (MFE) for delays $\tau \in [0.1\,\mathrm{s}, 0.2\,\mathrm{s}, ..., 1\,\mathrm{s}]$. All the models but the one trained using Teacher Forcing have negative MFE values, meaning that the predicted head positions over forecast the actual observations.

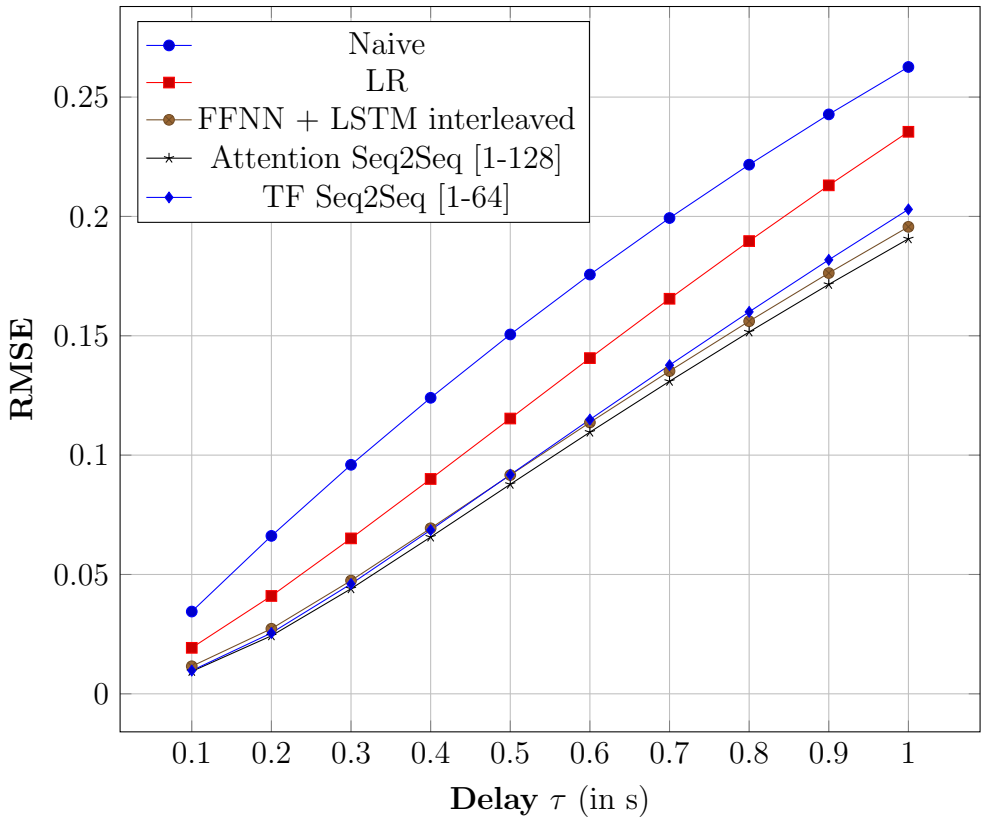


Figure 5.2: Root Mean Squared Error (RMSE) for delays $\tau \in [0.1\,\text{s}, 0.2\,\text{s}, ..., 1\,\text{s}]$. The Seq2Seq architecture with 1 LSTM layer and 128 neurons leads to the best results.

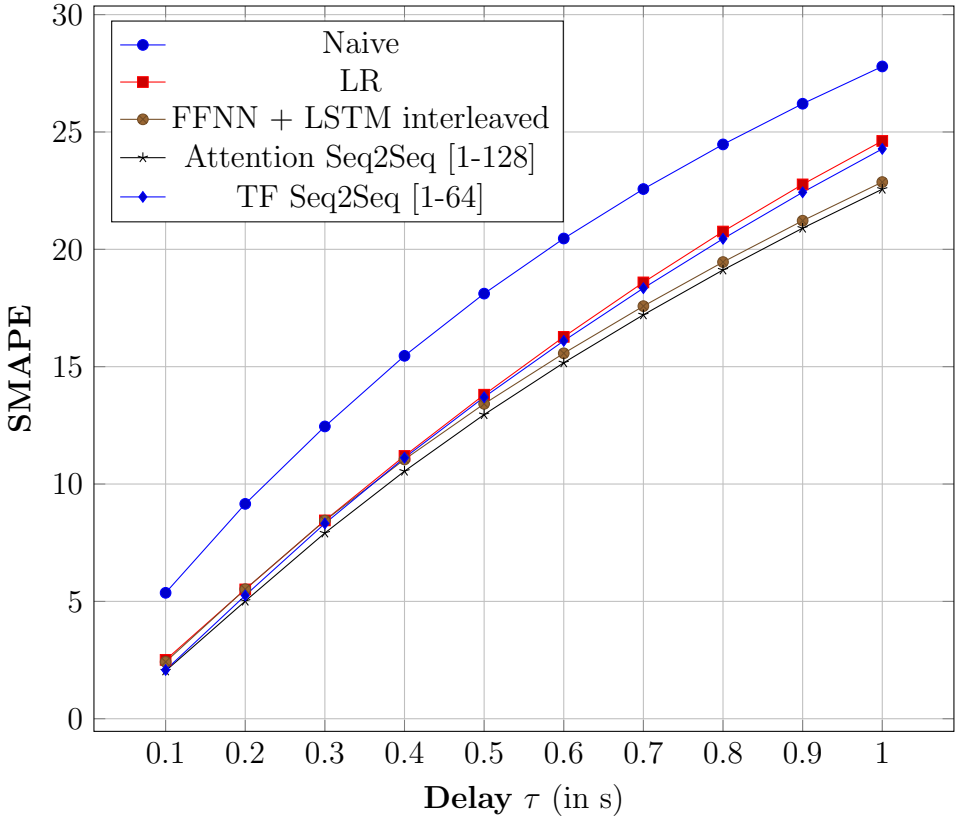


Figure 5.3: Symmetric Mean Absolute Percentage Error (SMAPE) for delays $\tau \in [0.1\,\text{s}, 0.2\,\text{s}, ..., 1\,\text{s}]$. The Seq2Seq architecture with 1 LSTM layer and 128 neurons leads to the best results.

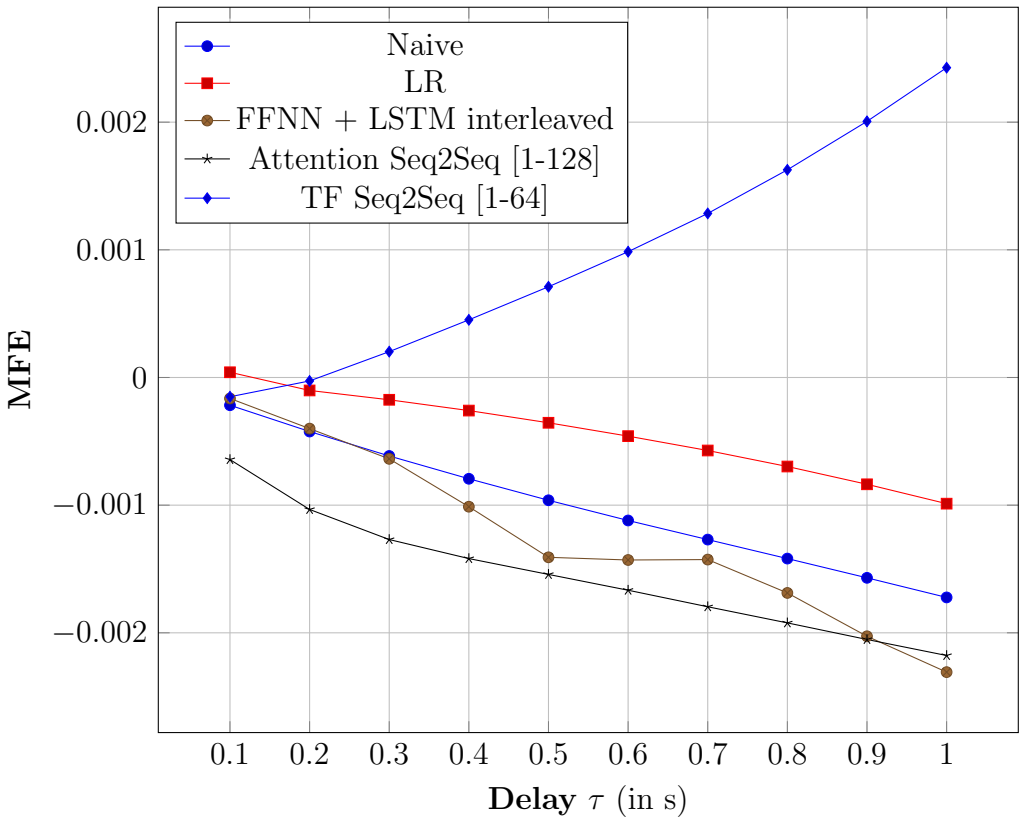


Figure 5.4: Mean Forecast Error (MFE) for delays $\tau \in [0.1\,\text{s}, 0.2\,\text{s}, ..., 1\,\text{s}]$. The models tend to have negative MFE values, meaning that the predicted head positions overshoot the actual observations.

Chapter 6

Conclusions

In this Bachelor's Thesis, we work on the Head Movement Prediction (HMP) task, which can help to increase video quality on a limited bandwidth and reduce the Motion-to-Photon (M2P) latency to allow VR users to live a satisfactory immersive experience without virtual reality sickness.
We examine a Deep Learning model based on the novel Encoder-Decoder Sequence-to-Sequence (Seq2Seq) architecture and evaluate different approaches regarding the input to the decoder module, the training method (Teacher Forcing) and the context vector connecting both modules. The main advantage of the Seq2Seq model is the flexibility it offers when dealing with many to many (m to n) time series forecasts. Although the TF-trained Seq2Seq model with output reinjection in the decoder achieves the best performance for the delay $\tau = 0.1\,\mathrm{s}$, our proposed model consisting of 1 stacked LSTM with 128 neurons for encoder and decoder has the highest average accuracy. The proposed model slightly outperforms prior state-of-the-art, which indicates the fitness of Seq2Seq models for this task.

Using saliency maps as a content-based feature has the potential to improve the head motion prediction. We develop a saliency encoder module based on Convolutional Neural Network layers to extract visual features of the saliency map of the user's current Field of View and use it as input to the decoder part of our proposed Seq2Seq model. Nevertheless, the prediction accuracy does not seem to profit from this early-fusion approach.

There is still room for improvement for predicting head motion. Future work may benefit from using gaze as an additional sensor-based feature. Regarding saliency maps, related studies indicate the increase of performance achieved when merging the saliency image with the original image or when applying optical flow to achieve temporal dependencies. Furthermore, a late fusion approach like computing the average between saliency peaks may be more reliable than letting CNNs extract the visual features automatically. Finally, DL networks greatly rely on the amount of data. Hence, a larger dataset may undoubtedly lead to improved predictions.

Appendix A

Appendix

A.1 Model Architectures

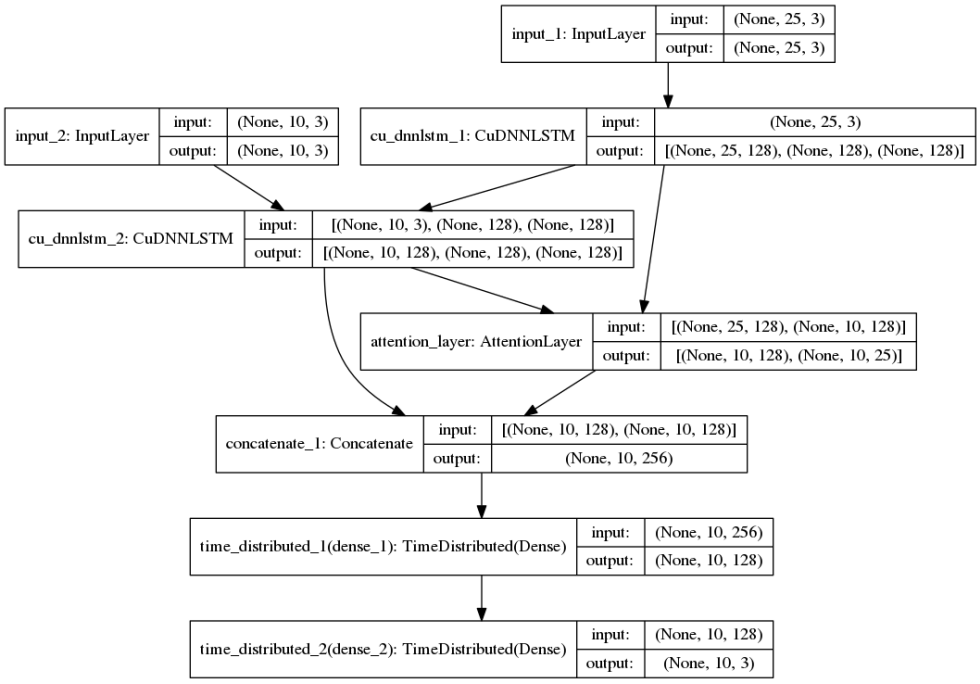


Figure A.1: Encoder-Decoder Sequence-to-Sequence [1-128] with attention layer and decoder zeros.

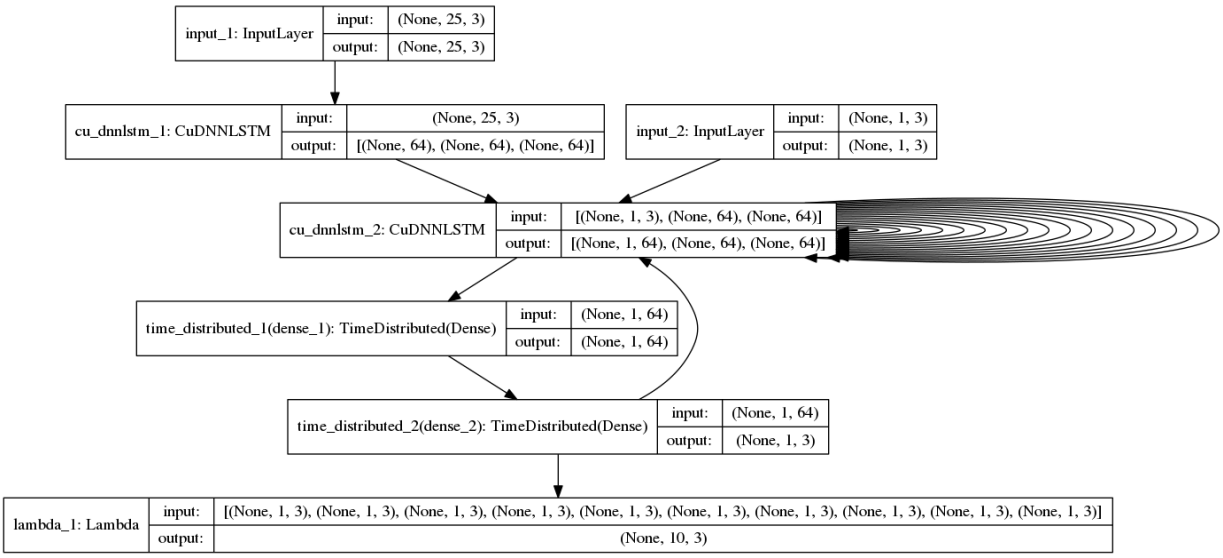


Figure A.2: Encoder-Decoder Sequence-to-Sequence [1-64] using output reinjection in the decoder.

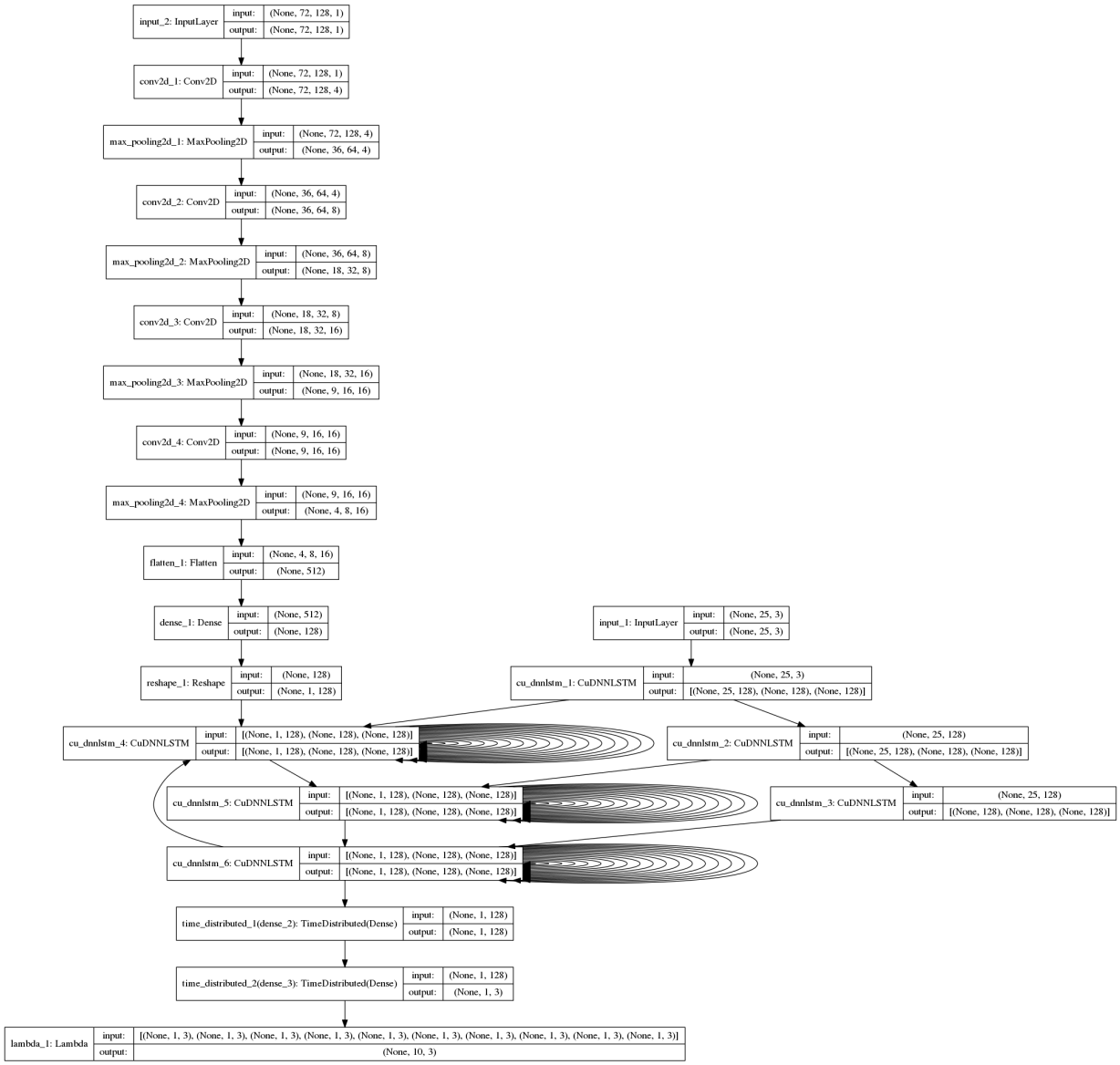


Figure A.3: Encoder-Decoder Sequence-to-Sequence [3-128] with the proposed saliency encoder module based on CNNs, using output reinjection in the decoder.

List of Figures

List of Tables

Acronyms

AdaGrad Adaptive Gradient Algorithm

Adam Adaptive Moment Estimation

ADF Augmented Dickey Fuller

ARIMA Autoregressive Integrated Moving Average

BPTT Backpropagation Through Time

CNN Convolutional Neural Network

CV Computer Vision

DL Deep Learning

DNN Deep Neural Network

FFNN Feedforward Neural Network

FoV Field of View

GAN Generative Adversarial Network

GRU Gated Recurrent Unit

HMD Head Mounted Display

HMP Head Movement Prediction

LR Linear Regression

LSTM Long Short-Term Memory

M2P Motion-to-Photon

MAE Mean Absolute Error

MFE Mean Forecast Error

MSE Mean Squared Error

QoS Quality of Service

RMSE Root Mean Squared Error

RMSprop Root Mean Square Propagation

RNN Recurrent Neural Network

Seq2Seq Sequence-to-Sequence

SGD Stochastic Gradient Descent

SMAPE Symmetric Mean Absolute Percentage Error

TF Teacher Forcing

VR Virtual Reality

Bibliography

[ABK+18] Tamay Aykut, Christoph Burgmair, Mojtaba Karimi, Jingyi Xu, and Eckehard Steinbach. Delay compensation for actuated stereoscopic 360 degree telepresence systems with probabilistic head motion prediction. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 2010–2018. IEEE, 2018.

[AD19] Ioannis Agtzidis and Michael Dorr. Getting (more) real: Bringing eye movement classification to hmd experiments with equirectangular stimuli. In *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*, ETRA '19, pages 18:1–18:8, New York, NY, USA, 2019. ACM.

[AEJK17] A Deniz Aladagli, Erhan Ekmekcioglu, Dmitri Jarnikov, and Ahmet Kondoz. Predicting head trajectories in 360 virtual reality videos. In *2017 International Conference on 3D Immersion (IC3D)*, pages 1–6. IEEE, 2017.

[AEWS08] Radhakrishna Achanta, Francisco Estrada, Patricia Wils, and Sabine Süsstrunk. Salient region detection and segmentation. In *International conference on computer vision systems*, pages 66–75. Springer, 2008.

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[BI13] A. Borji and L. Itti. State-of-the-art in visual attention modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):185–207, Jan 2013.

[BJB+] Zoya Bylinskii, Tilke Judd, Ali Borji, Laurent Itti, Frédo Durand, Aude Oliva, and Antonio Torralba. Mit saliency benchmark.

[BSI13] A. Borji, D. N. Sihite, and L. Itti. Quantitative analysis of human-model agreement in visual saliency modeling: A comparative study. *IEEE Transactions on Image Processing*, 22(1):55–69, Jan 2013.

[BT06] Neil Bruce and John Tsotsos. Saliency based on information maximization. In *Advances in neural information processing systems*, pages 155–162, 2006.

[BWZ+16] Yanan Bao, Huasen Wu, Tianxiao Zhang, Albara Ah Ramli, and Xin Liu. Shooting a moving target: Motion-prediction-based transmission for 360-degree videos. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 1161–1170. IEEE, 2016.

[CBSC16] Marcella Cornia, Lorenzo Baraldi, Giuseppe Serra, and Rita Cucchiara. A deep multi-level network for saliency prediction. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 3488–3493. IEEE, 2016.

[CvMG+14] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[EK16] David Pio Evgeny Kuzyakov, Sean Liu. Optimizing 360 video for oculus. `https://developers.facebook.com/videos/f8-2016/optimizing-360-video-for-oculus/`, 2016.

[FE17] Fu-En.Wang. Equirec2perspec. `https://github.com/fuenwang/Equirec2Perspec`, 2017.

[FLL+17] Ching-Ling Fan, Jean Lee, Wen-Chih Lo, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu. Fixation prediction for 360 video streaming in head-mounted virtual reality. In *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 67–72. ACM, 2017.

[Gan19] Thushan Ganegedara. Keras layer implementation of attention. `https://github.com/thushv89/attention_keras`, 2019.

[GI17] Voleon Google Inc. Web traffic time series forecasting. `https://www.kaggle.com/c/web-traffic-time-series-forecasting/overview/evaluation`, 2017.

[GLM17] Yanan Guo, Chongfan Luo, and Yide Ma. Object detection system based on multimodel saliency maps. *J. Electronic Imaging*, 26:23022, 2017.

[GV05] Dashan Gao and Nuno Vasconcelos. Discriminant saliency for visual recognition from cluttered scenes. In *Advances in neural information processing systems*, pages 481–488, 2005.

[GZ09] Chenlei Guo and Liming Zhang. A novel multiresolution spatiotemporal saliency detection model and its applications in image and video compression. *IEEE transactions on image processing*, 19(1):185–198, 2009.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[HSBZ15] Xun Huang, Chengyao Shen, Xavier Boix, and Qi Zhao. Salicon: Reducing the semantic gap in saliency prediction by adapting deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 262–270, 2015.

[HZ07] Xiaodi Hou and Liqing Zhang. Saliency detection: A spectral residual approach. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. Ieee, 2007.

[HZ09] Xiaodi Hou and Liqing Zhang. Dynamic visual attention: Searching for coding length increments. In *Advances in neural information processing systems*, pages 681–688, 2009.

[IB09] Laurent Itti and Pierre Baldi. Bayesian surprise attracts human attention. *Vision research*, 49(10):1295–1306, 2009.

[IK01] Laurent Itti and Christof Koch. Computational modelling of visual attention. *Nature reviews neuroscience*, 2(3):194, 2001.

[IKN98] L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259, Nov 1998.

[IMS+17] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2462–2470, 2017.

[JEDT09] T. Judd, K. Ehinger, F. Durand, and A. Torralba. Learning to predict where humans look. In *2009 IEEE 12th International Conference on Computer Vision*, pages 2106–2113, Sep. 2009.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[KU87] Christof Koch and Shimon Ullman. Shifts in selective visual attention: towards the underlying neural circuitry. In *Matters of intelligence*, pages 115–141. Springer, 1987.

[KWGB17] Matthias Kummerer, Thomas SA Wallis, Leon A Gatys, and Matthias Bethge. Understanding low-and high-level contributions to fixation prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4789–4798, 2017.

[LBB+98] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LH16] Nian Liu and Junwei Han. Dhsnet: Deep hierarchical saliency network for salient object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 678–686, 2016.

[LJHX08] Huiying Liu, Shuqiang Jiang, Qingming Huang, and Changsheng Xu. A generic virtual content insertion system based on visual attention analysis. In *Proceedings of the 16th ACM international conference on Multimedia*, pages 379–388. ACM, 2008.

[LMLCBT06] Olivier Le Meur, Patrick Le Callet, Dominique Barba, and Dominique Thoreau. A coherent computational approach to model bottom-up visual attention. *IEEE transactions on pattern analysis and machine intelligence*, 28(5):802–817, 2006.

[Loo16] Kinson Loo. Z cam s1 times square vr. `https://www.youtube.com/watch?v=fgZAvZ3kh7c`, Sep 2016.

[LYKA14] Steven M LaValle, Anna Yershova, Max Katsev, and Michael Antonov. Head tracking for the oculus rift. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 187–194. IEEE, 2014.

[LYS+10] Tie Liu, Zejian Yuan, Jian Sun, Jingdong Wang, Nanning Zheng, Xiaoou Tang, and Heung-Yeung Shum. Learning to detect a salient object. *IEEE Transactions on Pattern analysis and machine intelligence*, 33(2):353–367, 2010.

[LZLW19] Chenge Li, Weixi Zhang, Yong Liu, and Yao Wang. Very long term field of view prediction for 360-degree video streaming. In *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 297–302. IEEE, 2019.

[MV09] Vijay Mahadevan and Nuno Vasconcelos. Spatiotemporal saliency in dynamic scenes. *IEEE transactions on pattern analysis and machine intelligence*, 32(1):171–177, 2009.

[NI06] Vidhya Navalpakkam and Laurent Itti. An integrated model of top-down and bottom-up attention for optimizing detection speed. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 2049–2056. ieee, 2006.

[NYN18] Anh Nguyen, Zhisheng Yan, and Klara Nahrstedt. Your attention is unique: Detecting 360-degree video saliency in head-mounted display for head movement prediction. In *2018 ACM Multimedia Conference on Multimedia Conference*, pages 1190–1198. ACM, 2018.

[Ola15] Christopher Olah. Understanding lstm networks. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 8 2015.

[PFM+17] Junting Pan, Cristian Canton Ferrer, Kevin McGuinness, Noel E O'Connor, Jordi Torres, Elisa Sayrol, and Xavier Giro-i Nieto. Salgan: Visual saliency prediction with generative adversarial networks. *arXiv preprint arXiv:1701.01081*, 2017.

[PSGiN+16] Junting Pan, Elisa Sayrol, Xavier Giro-i Nieto, Kevin McGuinness, and Noel E O'Connor. Shallow and deep convolutional networks for saliency prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 598–606, 2016.

[SAA02] Albert Ali Salah, Ethem Alpaydin, and Lale Akarun. A selective attention-based method for visual pattern recognition with application to handwritten digit recognition and face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3):420–425, 2002.

[SIVA17] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[Sui17] Arthur Suilin. Web traffic time series forecasting - 1st place solution. `https://www.kaggle.com/c/web-traffic-time-series-forecasting/discussion/43795#latest-625371`, 2017.

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[Tor03] Antonio Torralba. Modeling global scene factors in attention. *JOSA A*, 20(7):1407–1418, 2003.

[XDW+18] Yanyu Xu, Yanbing Dong, Junru Wu, Zhengzhong Sun, Zhiru Shi, Jingyi Yu, and Shenghua Gao. Gaze prediction in dynamic 360 immersive videos. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5333–5342, 2018.

[ZTC09] Lingyun Zhang, Matthew H Tong, and Garrison W Cottrell. Sunday: Saliency using natural statistics for dynamic analysis of scenes. In *Proceedings of the 31st annual cognitive science conference*, pages 2944–2949. AAAI Press Cambridge, MA, 2009.

[ZTM+08] Lingyun Zhang, Matthew H Tong, Tim K Marks, Honghao Shan, and Garrison W Cottrell. Sun: A bayesian framework for saliency using natural statistics. *Journal of vision*, 8(7):32–32, 2008.