Historical Newspaper Optical Character Recognition based on State Space Models

Christian Schultze

Born on the 31.08.1998, Munich

Master Lab Computer Science

Supervisor: Dr. Moritz Wolter
Second Reviewer: Prof. Dr. Reinhard Klein

HPC/A-Lab and Institute for Computer Science II

Faculty of Mathematics and Natural Sciences
Rheinische Friedrich-Wilhelms-Universität Bonn

February 25, 2025

Contents

Chapter 1

Introduction

Modelling long sequences is an important machine learning task. Being able to efficiently model sequences, is necessary for Natural Language Processing (NLP) applications like translation and text generation. These involve natural language sequences as input and output. Therefore, it is common to refer to models solving such tasks as sequence-to-sequence models.

The task of Optical Character Recognition (OCR) requires to read an image with written text and provide the corresponding characters. This can be done by identifying every single character in the image, extracting a bounding box. Each bounding box can then be processed to determine the character. However, OCR can be interpreted as a sequence-to-sequence task, making it possible to utilize progress made for NLP tasks. For this, one text-line inside an image will be represented as one input sequence. The output is the corresponding text sequence. Assuming all text-lines are more or less horizontal, the x-axis of a text-line crop represents the input sequence. Each element of that sequence contains all pixels of that respective column within the feature dimensions. Therefore, each character spans a number of elements in the sequence.

Modelling sequences can be done with a number of deep neural network types. This includes CNNs, RNNs and Transformers. However, convolutional models suffer from a limited context length, as well as an expensive inference, due to their nature not being sequential. The Attention Block in Transformers suffers from efficiency issues, as it scales quadraticly with the sequence length. Furthermore, RNNs and transformers suffer from the vanishing gradients problem. This is especially important for modelling very long sequences. Gu et al. on the other hand use an ODE to model long sequences within their State Space Models (SSM)s. ODEs are operators acting on functions and map one function to another. Therefore, they can map continuous signals from one to another and after discretization they map sequences to sequences. [3, 5]

The core of our OCR model the State Space Recognizer (SSR), is the mamba2 [3] SSM from Gu et al. Mamba2 is a very advanced version of the LSSL model (section 2.1), showing state of the art results, while being fast to compute and train [3]. Before the first mamba2 block within our SSR, the input

2D-crop is converted to a 1D-sequence. This data is processed through multiple mamba2 blocks within an encoder-decoder architecture. The encoder has the task to encode the image data and provide the entire input sequence to the decoder. The decoder has to generate a text-sequence after processing the entire encoder data and is designed to receive its own output for the generation of the next character. This is similar to many NLP models.

We train the SSR on the Chronicling Germany Historical Newspaper Dataset [9], providing historic german newspaper data, set in Fraktur font. The Fraktur font makes it challenging to modern readers and OCR models not specifically trained for this font. We compare the results with two other models trained on this dataset. The Test results of the LSTM and transformer are provided within the Chronicling Germany paper. [9]

Chapter 2

Foundations

2.1 LSSL

2.1.1 Introduction

The Linear State Space Layer [5] is defined by the discretization of an ODE. This leads to a recurrent view on the model (subsection 2.1.3), inheriting the efficient inference from RNNs. The recurrent view can be unrolled to a convolution (subsection 2.1.4), enabling parallelization during training. Additionally, Gu et al. use their HIPPO Matrices (section 2.2) to address the vanishing gradients problem. [5]

2.1.2 Continuous Computations

SSMs are ODEs, mapping a 1-dimensional function $u(t) : \mathbb{R} \to \mathbb{R}$ to the output $y(t)$ through a hidden state $x(t) \in \mathbb{R}^N \ N \in \mathbb{N}$.

$$\text{State equation } \dot{x}(t) = Ax(t) + Bu(t) \tag{2.1}$$

$$\text{Output equation } y(t) = Cx(t) + Du(t) \tag{2.2}$$

Updates of the hidden state $\dot{x}(t)$ (Equation 2.1) depend on the current input $u(t)$, as well as the state $x(t)$ itself. Matrix A controls the information flow from the past hidden state and Vektor B controls the information flow from the current input to the current hidden state. As SSMs are designed to model long range sequential data, Matrix A is of particular importance. All Information that has to be modelled over a long range, will repeatedly be modified by A. Output values $y(t)$ depend on the hidden state $x(t)$, as well as the current input $u(t)$. Vektor C controls the information flow from the hidden state to the output and Matrix D controls the information flow from $u(t)$ to $y(t)$. This can be viewed as a skip connection, which allows us to concentrate on the inner part with only Matrices A,B and C. [5, 2]

2.1.3 Discrete representation and recurrent view

To use the continuous ODE with discrete data, Gu et al. discretize it by a step size Δ. Applied to a continuous signal, this would mean sampling it with a resolution of Δ. One Discretization method is the Euler method (Equation 2.6). Rearranging gives the discrete matrices $\overline{A}$ and $\overline{B}$ (Equation 2.9) [5].

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) && (2.3)\\ y(t) &= Cx(t) && (2.4)\\ \text{Eulers discretization} & && (2.5)\\ x(t+\Delta t) &\approx x(t) + \Delta t\dot{x}(t) && (2.6)\\ &= x(t) + \Delta t(Ax(t) + Bu(t)) && (2.7)\\ &= (I + \Delta tA)x(t) + \Delta tBu(t) && (2.8)\\ &= \overline{A}x(t) + \overline{B}u(t) && (2.9) \end{aligned}$$

Matrix C does not need to be modified for discretization. Therefore $\overline{C} = C$. Gu et al. use the GBT, which is a generalized version of the Euler method. GBT with $\alpha = 0$ corresponds to the Euler method. Their discrete matrices differ from those in Equation 2.9, as they use $\alpha = \frac{1}{2}$. Now the model can be used on sequences u_k with the output y_k. This view is recurrent, because the hidden state x_k depends on the previous hidden state x_{k-1} (Equation 2.10). Previously, this was a continuous gradient for the hidden state whereas now we can compute one discrete hidden state from the previous one. The step size Δ is an important hyper parameter. [5]

$$\begin{aligned} x_k &= \overline{A}x_{k-1} + \overline{B}u_k && (2.10)\\ y_k &= \overline{C}x_k && (2.11) \end{aligned}$$

These equations enable efficient inference, requiring fixed computation and storage per time step, although the entire input sequence grows by one input each time step. Training with these equations however, has the same draw back as with RNNs. At the time of training all future inputs and outputs are known, which makes it inefficient to compute sequentially. [5]

2.1.4 Convolutional View

SSMs can be viewed as convolutions. It is possible to unroll the recurrent equations to a convolution with a 1D kernel $\overline{K}^{(L)} \in \mathbb{R}^L$ of length L. Unlike kernels used in CNNs, values of $\overline{K}^{(L)}$ will not be optimized individually. Instead, all values are calculated from the matrices $\overline{A}$, $\overline{B}$ and $\overline{C}$. To approach the derivation of $\overline{K}^{(L)}$, we will first concentrate on $\overline{A}$ and $\overline{B}$, as well as the computation of the hidden state (Equation 2.12). When explicitly calculating individual values of the input x_k, a repeating pattern can be observed. E.g. in Equation 2.17 the formula from Equation 2.16 reoccurs, with the indices u_k shifted by one. This corresponds to a convolution kernel $\overline{K}^{(2)} := (\overline{AB}, \overline{B})$ applied to the vector $u^{(2)} := (u_0, u_1)$, or $x_1 = \langle \overline{K}^{(2)}, u^{(2)} \rangle$. Crucially, $x_1 = \langle \overline{K}^{(2)}, u^{(2)} \rangle$ and $x_2 = \langle \overline{K}^{(3)}, u^{(3)} \rangle$ can be calculated independently and don't depend on each other, as long as all inputs are known beforehand. As this is true at time of training, this enables parallel computations during training. [5]

$$x_k = \overline{A}x_{k-1} + \overline{B}u_k \tag{2.12}$$
$$y_k = \overline{C}x_k \tag{2.13}$$
$$\text{Unrolling } x \tag{2.14}$$
$$x_0 = \overline{B}u_0 \tag{2.15}$$
$$x_1 = \overline{AB}u_0\overline{B}u_1 \tag{2.16}$$
$$x_2 = \overline{A}^2\overline{B}u_0\overline{AB}u_1\overline{B}u_2 \tag{2.17}$$

For training it is necessary to compute $y_k = \overline{C}x_k$ and optimize parameters depending on the calculated loss. For this, gu et al. skip the computation of the hidden state x and define a kernel $\overline{K}^{(L)} \in \mathbb{R}^L$, that depends on $\overline{A}$, $\overline{B}$ and $\overline{C}$ (Equation 2.18) and is applied to $u^{(L)} := (u_0, \cdots u_{L-1})$ to calculate y_L(Equation 2.19). [5]

$$\overline{K}^{(L)} := (\overline{CB}, \overline{CAB}, \overline{CA}^2\overline{B}, \cdots, \overline{CA}^{L-1}\overline{B}) \tag{2.18}$$
$$y_L = \langle \overline{K}, u^{(L)} \rangle \tag{2.19}$$

$\overline{K}^{(L)}$ can be used to compute all y_k and the corresponding loss independently, allowing parallelized training. All inputs and targets are known at time of training. In this case, $\overline{K}^{(L)}$ covers the entire history of inputs and is supposed to extract information that are relevant for the task. [5]

2.2 HIPPO

2.2.1 Overview

Like RNNs, SSMs suffer from the vanishing/exploding gradients problem. As powers of $\overline{A}$ are computed in the convolution kernel (Equation 2.18), random initializations of $\overline{A}$ lead to exploding or vanishing gradients. High-order Polynomial Projection Operator (HIPPO) matrices [6] are designed to memorize input history and address this issue. They are used as initialization for SSMs. Gu et al. derive HIPPO matrices by constructing a basis of a function Hilbert space out of Legendre polynomials. By projecting inputs on to that basis, they can approximate the previous history. Initializing Matrix A as a HIPPO matrix instead of a random matrix leads to greatly increased performance. [5]

2.2.2 Polynomial Approximation

Quality Assessment

To address the vanishing gradient issue, Gu et al. derive a method to approximate the cumulative history of an input $u(t)$ up to a time τ. For this, they first develop a way to asses the quality of such an approximation. τ is the current time, up to which the history has been recorded. $u(t) \in \mathbb{R}$ is the input function with $u_{\leq \tau} := u(t)|_{t \leq \tau}$ defined only up to τ. $u_{\leq \tau}$ is representing the history up to time τ. This notation differs from [6]. It is supposed to clarify the difference between t and τ and be more coherent with conventions in control theory. [5]

For an input function $u(t) \in \mathbb{R}$ with the cumulative history $u_{\leq t} := u(x)|_{x \leq t}$ Gu et al. define a distance in function space, to assess the quality of an approximation for $u_{\leq t}$ with respect to a time depended probability measure $\mu^{(\tau)}$ on $[0, \tau]$. $\mu^{(\tau)}$ is used as weight function to assign importance to parts of the cumulative history. E.g. assigning more value to the recent history than the past history. To assess the quality of an approximation, an inner product is defined, that integrates two functions $u(t), h(t) \in \mathbb{R}$ with respect to the measure $\mu^{(\tau)}$ (Equation 2.20). Therefore, $||u_{\leq \tau} - h(t)||_{L_2(\mu^{(\tau)})}$ (Equation 2.21) is the distance between the cumulative history and some function $h(t)$. This distance can be used to asses the quality of an approximation for the cumulative history. [6]

$$\langle u, h \rangle_{\mu^{(\tau)}} := \int_0^\tau u(t)h(t)d\mu^{(\tau)}(t) \tag{2.20}$$

$$||u||_{L_2(\mu^{(\tau)})} = \langle u, u \rangle_{\mu^{(\tau)}}^{\frac{1}{2}} \tag{2.21}$$

Gu et al. use polynomials of order N as approximation of $u(t)$. The coefficients $x^{(\tau)} \in \mathbb{R}^N$ of the polynomials represent the history at time τ. Let $\mathcal{G}$ be the set of polynomials of degree less than N, then we seek some $g^{(\tau)} \in \mathcal{G}$ that minimizes $\| u_{\leq \tau} - g^{(\tau)} \|_{L_2(\mu^{(\tau)})}$. Since the level of detail of the approximation can vary through time, e.g. only the recent past should be approximated accurately, the measure $\mu^{(\tau)}$ can change through time as well. [6]

Calculating Coefficients

The set of polynomials $\mathcal{G}$ is a subspace of the Hilbert space $\mathcal{H}_\mu$, with corresponding norm $||u||_{L_2(\mu)}$. The history can be represented by the N coefficients of the approximating polynomials in any basis of $\mathcal{G}$. Gu et al. use an orthonormal basis $\{g_n\}_{n<N}$ of $\mathcal{G}$ with size $N \in \mathbb{N}$, to enable the calculation of the coefficients $x_n^{(\tau)}$ by projection onto that basis. As $\{g_n\} : n < N$ is an orthonormal basis, the projection of the cumulated history $u_{\leq \tau}$ onto $\{g_n\}$, yields the coefficients in basis $\{g_n\}$ (Equation 2.22). [6]

$$x_n^{(\tau)} = \langle u_{\leq \tau}, g_n \rangle_{\mu^{(\tau)}} \tag{2.22}$$

In their LSSL, Gu et al. use a set of legendre polynomials as basis and derive the HIPPO operator, that is used to efficiently approximate and store the cumulated history (section 2.2.3). [5]

2.2.3 Derivation of HIPPO operators

Coefficient Dynamics

Gu et al. store the cumulative history by projecting it onto an orthonormal basis of order $N \in \mathbb{N}$. Let $\{p_n^{(\tau)}(t)\} : n < N$ be a sequence of orthogonal polynomials with respect to the time-varying measure $\mu^{(\tau)}$. $\{p_n^{(\tau)}(t)\}$ is scaled to form the orthonormal basis $\{g_n\}$ (Equation 2.23). λ_n is an additional scaling factor, that normalizes the polynomials. It does not change the orthogonality, as it only scales the dot product by λ_n^2 and therefore preserves orthogonality. [6, Appendix C, Page 22]

$$g_n^{(\tau)} = \lambda_n p_n^{(\tau)} \tag{2.23}$$

The polynomials $g^{(\tau)}$ are optimal, if they satisfy Equation 2.24 with the set of all polynomials $\mathcal{G}$. [6]

$$g^{(\tau)} = \text{argmin}_{g \in \mathcal{G}} ||u_{\leq t} - g^{(\tau)}||_{L_2(\mu^{(\tau)})} \tag{2.24}$$

The corresponding optimal coefficients $x_n^{(\tau)} \in \mathbb{R}$ can be calculated by integrating the product of u and $p_n^{(\tau)}$ (Equation 2.28), as defined by the inner product, used for projecting u onto the basis $\{g_n\}$ (section 2.2.2). [6, Appendix C, Page 22]

$$\begin{aligned}
x_n^{(\tau)} &= \langle u_{\leq \tau}, g_n \rangle_{\mu^{(\tau)}} && (2.25)\\
&= \int_0^\tau u g_n^{(\tau)} d\mu^{(\tau)}(t) && (2.26)\\
&= \int_0^\tau u g_n^{(\tau)} \omega^{(\tau)} dt && (2.27)\\
&= \lambda_n \int_0^\tau u p_n^{(\tau)} \omega^{(\tau)} dt && (2.28)
\end{aligned}$$

Gut et al. assume that the measures $\mu^{(\tau)}$ have a probability density function of $\omega^{(\tau)}(t) \in \mathbb{R}$. This allows us to rewrite the integration in Equation 2.26 with $d\mu^{(\tau)}(t)$ to the integral in Equation 2.27 with $\omega^{(\tau)}(t)dt$. Gu et al. use a more general approach, which allows for the use of non orthogonal basis polynomials. This is not needed in this case, which means we omit the tilting and assume that the scaling function and normalization constant are obsolete as $\mathcal{X} = \xi = 1$. [6, 5]

To apply the coefficient calculation (section 2.2.2) on a continuous input function $u(t) \in \mathbb{R}$, Gu et al. calculate the temporal derivation $\frac{d}{d\tau}x_n^{(\tau)}$. The hippo operator maps u to the optimal coefficients x. When processing the input, this needs to be updated continuously. Every new input changes the optimal coefficients. [6]

Gu et al. state, that $\frac{d}{d\tau}x_n^{(\tau)}$ can be expressed as an ODE, if $\frac{d}{d\tau}p_n^{(\tau)}$ and $\frac{d}{d\tau}\omega^{(\tau)}$ (Equation 2.31) have closed form solutions, that can be related back to the polynomials p_n.[6]

$$\frac{d}{d\tau}x_n^{(\tau)} = \frac{d}{d\tau}\lambda_n \int_0^\tau u(t)p_n^{(\tau)}\omega^{(\tau)}dt \tag{2.29}$$

$$\text{Partial derivatives:} \qquad = \lambda_n \int_0^\tau u(t)\left(\frac{\partial}{\partial\tau}p_n^{(\tau)}(t)\right)\omega^{(\tau)}(t)dt \tag{2.30}$$

$$+ \lambda_n \int_0^\tau u(t)p_n^{(\tau)}(t)\left(\frac{\partial}{\partial\tau}\omega^{(\tau)}(t)\right)dt \tag{2.31}$$

[6]

Scaled Legendre measure

Gu et al. use the Scaled Legendre measure to approximate the history while avoiding the vanishing gradient problem. It utilizes an orthonormal basis consisting of Legendre Polynomials (section 2.2.3) and uses a measure with varying width. Let $\mu^{(\tau)} = \frac{1}{\tau}$ on $[0, \tau]$ be the time dependant measure and $p_n^{(\tau)}(t) = \sqrt{(2n+1)}P_n\left(\frac{2t}{\tau} - 1\right)$ be the orthonormal basis on $[0, \tau]$ with P_n being the n-th Legendre Polynomial. Gu et al. calculate the closed form solutions for $\frac{d}{d\tau}p_n^{(\tau)}$ and $\frac{d}{d\tau}\omega^{(\tau)}$ and plug them into Equation 2.31. The result is the definition of A and B. Gu et al. use these to initialize their LSSL models, while scaling by $\frac{1}{\tau}$ (Equation 2.32). [6, Appendix D.3 Page 30]

$$A_{nk} = \begin{cases} (2n+1)^{\frac{1}{2}}(2k+1)^{\frac{1}{2}} & \text{if} n > k \\ m+1 & \text{if} n = k \\ 0 & \text{if} n < k \end{cases} \tag{2.32}$$

$$B_n = (2n+1)^{\frac{1}{2}} \tag{2.33}$$

Legendre Polynomials

Legendre Polynomials are defined as orthogonal polynomials $P_n(t) : n, m \in \mathbb{N}$ with respect to the measure $\mu(t) = 1$ on $[-1, 1]$, as well as $P_n(1) = 1$. Therefore, for $n \neq m$ they satisfy $\langle P_n, P_m \rangle = \int_{-1}^1 P_n(t)P_m(t)dt = 0$. For $n = m$ they are defined, so that $\int_{-1}^1 P_n(t)P_m(t)dt = ||P_n||^2 = \dfrac{2}{2n+1}$. Additionally $P_n(-1) = (-1)^n$ holds for all $n \in \mathbb{N}$. [6, Appendix B.1.1 Page 18]

To use Legendre Polynomials on the recent history $u_{\leq\tau}(t)$, Gu et al. scale them onto $[0,\tau]$ with the measure $\mu^{(\tau)}(t) = \frac{1}{\tau}$ on $[0,\tau]$. The corresponding transformation reads $t = \frac{2\tilde{t}}{\tau} - 1$ with scaled polynomials $\tilde{P}_n(\tilde{t}) = P_n(\frac{2\tilde{t}}{\tau} - 1)$. E.g. for the boundaries this preserves values. $\tilde{t}_1 = 0$, $\tilde{t}_2 = \tau$: $\frac{2\tilde{t}_1}{\tau} - 1 = -1$ and $\frac{2\tilde{t}_2}{\tau} - 1 = 1$. This allows for the substitution of the scaled polynomials on the inner product. The differential of the scaled integral changes from dt to $\frac{2d\tilde{t}}{\tau}$. Equation 2.34 shows, that the inner product value does not change through this scaling. This also applies to the previously defined squared norm (Equation 2.36). [6, Appendix B.1.1 Page 18]

$$\int_0^\tau P_n\left(\frac{2\tilde{t}}{\tau} - 1\right) P_m\left(\frac{2\tilde{t}}{\tau} - 1\right) \frac{2d\tilde{t}}{\tau} = \int_{-1}^{1} P_n(t)P_m(t)dt \tag{2.34}$$

$$2\int_0^\tau \tilde{P}_n(\tilde{t})\tilde{P}_m(\tilde{t})\mu^{(\tau)}(\tilde{t})d\tilde{t} = \int_{-1}^{1} P_n(t)P_m(t)dt \tag{2.35}$$

$$n = m : \int_0^\tau \tilde{P}_n(\tilde{t})\tilde{P}_m(\tilde{t})\mu^{(\tau)}(\tilde{t})d\tilde{t} = \frac{1}{2}||\tilde{P}_n||^2 = \frac{1}{2n+1} \tag{2.36}$$

Therefore, $||\tilde{P}_n|| = \frac{1}{\sqrt{2n+1}}$ and the normalized version of $\tilde{P}_n$ with respect to $\mu^{(\tau)}$ is $\sqrt{2n+1}\tilde{P}_n$. These form an orthonormal basis on $[0,\tau]$ and satisfy Equation 2.20. Thus, they can be used as polynomial basis for the derivation of HIPPO operators (subsection 2.2.3). [6, Appendix B.1.1 Page 18]

Chapter 3

Implementation

3.1 Overall

The Implementation is distributed over two Github repositories (Chronicling Germany Code[1] and State Space Recognizer[2]). This separates the code specific to training and processing newspaper pages from the core components of the SSM Model.

In both cases the main components are covered by tests, that are integrated into the corresponding Github repositories. This includes the Dataset for loading preprocessed newspaper pages on one hand and components of the state space recognizer on the other hand. The core Mamba-2 Block [3] is excluded from testing. Additional to the pytest coverage, the implementation has passed inspection by the linter pylint [1], as well as the static type checker mypy [7].

The implementation is supposed to load most configurations from a single yml configuration file. Most importantly, this include the model architecture. Defined are hidden dimensions, number of layers and number of blocks per layer, as well as whether downscaling is included in a layer. Furthermore, the vocabulary used for OCR is defined, as well as configurations for the tokenizer, preprocessing and inference together with hyper parameters for training.

3.2 State Space Recognizer

The SSR architecture is based on the Mamba-2 state space block from Gu et al. [3]. Mamba-2 is a very advanced version of the LSSL (section 2.1), combining state space models with attention in what they call State Space Duality. Mamba-2 shows to be efficient in training and inference, while being competitive with Transformers on language modelling tasks. [3]

The SSR uses Mamba-2 blocks within its encoder-decoder architecture, splitting image encoding and sequence-to-sequence decoding tasks. Each Mamba-2

[1] `https://github.com/Digital-History-Bonn/Chronicling-Germany-Code/tree/ocr_ssm`
[2] `https://github.com/Digital-History-Bonn/StateSpaceRecognizer`

block is followed by a fully connected layer with a single hidden layer. The encoder is designed to receive the cropped image of a single text line, being readable horizontally. We interpret the x-axis as a sequence with the y-axis as its feature dimensions. The fixed height of the y-axis for each crop is a hyper parameter proportional to the hidden dimension. After an initial 2D-convolutional downscaling, inputs are reshaped to be processed by the Mamba-2 SSM blocks.

The decoder is designed to output a sequence, whose length is not known at time of inference. This is due to the input data containing a single text line, without any information about individual characters. The generated sequence starts with a start-token and ends with an end-token. This is common in NLP tasks. The decoder has to process the encoded image data and generate the target sequence. Like other sequence-to-sequence models the decoder receives its own output back as input, enabling it to continue the sequence properly and output an end-token when there is nothing left to output.

For transformer architectures the cross attention enables processing of two different inputs. Typically, this is the encoded input, as well as the previous output. For SSMs there is only one input, so we feed the entire encoded image data to the encoder and then tell it to start generating by providing a start-token. Now each output is supplied back to the decoder via the same input it has previously received its image data. This means, the hidden states of the decoder have to be large enough to store all relevant information from the image. For the OCR task this is not of much concern, as a single text line in an image is expected to have a very limited amount of characters.

The Decoder has an embedding layer and a fully connected classifier, whose size depends on the used vocabulary. Each output is first classified as a specific character and then fed back into the decoder via the embedding layer.

3.3 Training

Training is handled by the PyTorch Lightning Module [4], which only needs minimal additional implementation for the training step. Data is split into train, validation and test dataset on the page level. All lines are extracted and preprocessed before training, creating one crop for each line. Training runs with mini-batches of at least size 32, while each crop is randomly augmented. Crops are padded on the x-axis to create a batch with uniform crop size. The padding index/value is excluded from loss calculation. Therefore, it does not contribute to gradients. One epoch of training means the model has seen the complete train dataset.

Preprocessing includes cutting out all line bounding boxes, as well as masking the resulting image to exclude all pixels outside of the line polygon. All crops that have been generated this way, are scaled to a fixed height to be compatible with the fixed hidden dimension of the SSR.

3.4 Inference

During Inference lines are extracted in a similar way to training and thus large portions of code are reused. Target-Strings are only included when evaluating models and targets are present. Like during training, crops are padded and combined to batches to maximize GPU utilization.

Inference is done in an autoregressive way. The decoder receives its previous output when generating the next one. Generation of outputs starts after all encoded image tokens have been processed by the decoder and it receives the start-token. Generating continues until an end-token has been generated or the maximum length has been reached. This applies to an entire batch. After generation the output sequences are converted to text and saved to an XML file.

Implementation wise the generation differs significantly from the forward step at time of training. The Mamba2 Block is implemented in a way, that the entire input sequence at time of training is supplied to the Mamba2 Blocks in the decoder in form of a 3-D Tensor. This includes all encoder-tokens as well as the embedded target sequence, that starts with a start-token after the last encoder token. During inference, the encoder-tokens are provided in the same way. After this however, the further sequence is not known and has to be generated. This means feeding only the start token to the decoder, receiving the output and feeding that back in, as long as it is not an end-token. Additionally, between each step of generation the hidden state has to be initialized with the appropriate values. For this, the Mamba2 block supports the allocation of inference cache, that has to be saved externally and provided with each input. Therefore, the inference cache is allocated before the encoder-tokens are fed to the decoder. For each further input the now updated inference cache is provided to the decoder. [3]

Chapter 4

Results

4.1 Data

The Chronicling Germany Historical Newspaper Dataset [9] is a collection of about 700 german newspaper pages from the time period between 1852 and 1924. For this work relevant are the baseline and corresponding text annotations. They enable us to crop individual text lines and train an OCR model to output the corresponding text-sequence. German newspapers from that period are set in the Fraktur font, therefore posing a challenge to modern readers and OCR models. Overall the dataset contains about 352.000 lines of text. The dataset is split into train(280,000 lines), validation(17,000 lines) and test(56,000 lines) data. [9]

All data is provided in form of XML files. Each newspaper page has its own XML file, containing region polygon data. Each text-region contains polygon data for each text-line, as well as corresponding text. [9]

4.2 Experiments

Besides the principal model architecture described in section 3.2, the number of parameters is an important factor for the ability of a model to learn a task. Within the present architecture the number of parameters depends on the number of layers and blocks in the encoder and decoder. We have conducted a hyper parameter search regarding these architecture details. Table 4.1 shows configurations and results for SSR models with increasing number of parameters. The results are evaluated on the test-part of the dataset, but within the training setting. Thus, these experiments do not include actually generating in inference mode. The number of layers and blocks in the encoder, as well as the number of blocks within the only layer of the decoder are modified between experiments. It is important to note, that after every encoder layer the sequential data is scaled down by a factor of two. At the same time the number of feature dimensions is doubled. The number of parameters of each block depends on the number of feature dimensions. Therefore, increasing the number of layers in the encoder will increase the number of parameters in all encoder blocks. The results are given as levenshtein-distance per character. The levenshtein-distance indicates the edit-distance from one text sequence to another. The better the result, the lower the distance.

Mostly, the levenshtein-distance decreases with increasing number of parameters. Experiments 1 in Table 4.1 show poor performance of the basic model with 0.5M parameters. Adding another block in both encoder and decoder (Experiment 2) increases performance significantly.

Another way of increasing the number of parameters is through increasing the number of layers in the encoder. Experiments 3 and 4 show the same performance, while experiment 3 having significantly less parameters. Further increasing the number of blocks in both versions leads to very good results in experiments 5 and 6. Again, it shows that increasing the number of blocks with a single encoder layer is more efficient regarding the number of parameters. Experiment 5 is considered to have the best model configuration, as the performance is within the standard deviation of experiment 6, while having less parameters.

Table 4.1: Model architecture hyper parameter search.

	Encoder		Decoder		
No.	Layers	Blocks	Blocks	Params	Distance
1	1	1	1	0.5M	0.34 ± 0.01
2	1	2	2	0.9M	0.07 ± 0.01
3	1	2	4	1.5M	0.04 ± 0.005
4	2	2	2	2.9M	0.04 ± 0.01
5	1	8	8	3.4M	0.023 ± 0.002
6	2	2	4	4.2M	0.022 ± 0.001

Evaluating this in inference mode on the test dataset yields good results as well, although not being able to match the performance of other OCR models trained on the same dataset. Table 4.2 shows the comparison between our SSR and the LSTM and Transformer model used in the Chronicling Germany paper [9]. While our model has fewer parameters, it shows a much higher rate of text lines with many errors. This might be due to a very limited amount of preprocessing in our implementation. Especially de skewing images based on the annotated baselines would be important. Our assumption, that all text lines are horizontal and creating crops accordingly, leads to poor performance when lines are curved or just tilted. However, reaching more than 30% of completely correct lines shows, that this OCR method has great potential.

Table 4.2: OCR results for our SSR and the LSTM and transformer models from Schultze et al. [9]. Results are given in Levenshtein-Distance, rate of correct lines and lines with many errors. Correct lines have a levenshtein distance of 0. Lines with many errors have a Levenshtein distance of less than 0.9 per character. Table Layout and values for the LSTM and transformer from [9]

Model	Params	Levenshtein-Distance	correct [%]	many errors [%]
LSTM	4.1M	0.02 ± 0.001	60.5 ± 0.34	8.1 ± 0.66
Transformer	28.8M	0.04 ± 0.01	56.2 ± 1.3	12.5 ± 2.3
SSR (ours)	3.4M	0.09 ± 0.01	32.8 ± 5.8	29.4 ± 2.3

4.3 Conclusion

This work shows the potential of State Space Models (SSM) outside the NLP field. We use the Mamba-2 Block from Gu et al. [3] to build a deep learning architecture, that shows good results when processing german historical newspaper OCR data. As it is set in Fraktur font, this data poses a challenge to modern readers and OCR models not specifically trained on the Fraktur font.

Our implementation interprets the line-based OCR task as a Sequence-to-Sequence task. This requires pre-annotated text-line polygons for each image, as well as test-sequence targets for training. Currently, this assumes, that all text-lines are horizontal within the x-y coordinate system of the corresponding image. For data not fulfilling this requirement, additional preprocessing has to be added in the future.

Abbreviations

CNN Convolutional Neural Network. 1, 4

GBT Generalized Bilinear Transfor. 4

HIPPO High-order Polynomial Projection Operator. 5, 7

LSSL Linear State Space Layer. 1, 7, 8, 10

LSTM Long Short Term Memory. 2, 14, 15

NLP Natural Language Processing. 1, 2, 11, 15

OCR Optical Character Recognition. 1, 2, 11, 13–15

ODE Ordinary Differential Equation. 1, 3, 4, 8

RNN Recurrent Neural Network. 1, 3–5

SSM State Space Models. 1, 3–5, 10, 11, 15

SSR State Space Recognizer. 1, 2, 10, 11, 13–15

Glossar

crop A crop is the result of cutting out a specific type of an image, creating a (usually) smaller image that can be processed further.. 1, 2, 11–14

Bibliography

[1] Python Code Quality Authority (PyCQA) and contributors. *Pylint: Static Code Analysis for Python*. `https://pylint.pycqa.org/`. Version as of 2025. 2003.

[2] *A Visual Guide to Mamba and State Space Models*. `https://www.maartengrootendorst.com/blog/mamba/`. Accessed: 2024-11-21.

[3] Tri Dao and Albert Gu. "Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality". In: *International Conference on Machine Learning (ICML)*. 2024.

[4] William Falcon. *PyTorch Lightning*. `https://github.com/Lightning-AI/lightning`. Accessed: 2025-02-17. 2019.

[5] Albert Gu et al. *Combining Recurrent, Convolutional, and Continuous-time Models with Linear State-Space Layers*. 2021. arXiv: `2110.13985 [cs.LG]`. URL: `https://arxiv.org/abs/2110.13985`.

[6] Albert Gu et al. *HiPPO: Recurrent Memory with Optimal Polynomial Projections*. 2020. arXiv: `2008.07669 [cs.LG]`. URL: `https://arxiv.org/abs/2008.07669`.

[7] Jukka Lehtosalo and contributors. *Mypy: Optional Static Typing for Python*. `http://mypy-lang.org/`. Version as of 2025. 2014.

[8] Minghao Li et al. *TrOCR: Transformer-based Optical Character Recognition with Pre-trained Models*. Sept. 2021. DOI: `10.48550/arXiv.2109.10282`.

[9] Christian Schultze et al. *Chronicling Germany: An Annotated Historical Newspaper Dataset*. 2024. arXiv: `2401.16845 [cs.DL]`. URL: `https://arxiv.org/abs/2401.16845`.

[10] Ebin Zacharias, Martin Teuchler, and Bénédicte Bernier. *Image Processing Based Scene-Text Detection and Recognition with Tesseract*. 2020. arXiv: `2004.08079 [cs.CV]`. URL: `https://arxiv.org/abs/2004.08079`.