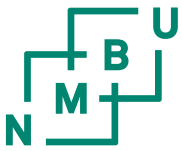

Norwegian
University of
Life Sciences

Master's Thesis 2024 30 ECTS
Faculty of Science and Technology

Exploration of Reservoir Computing and Artificial Neural Network Architectures as Saliency Detectors

Georg Vang
Environmental Physics and Renewable Energy

Abstract

This study explores the performance of reservoir-type and conventional artificial neural networks as saliency detectors, inspired by Li Zhaoping's hypothesis regarding the V1's role as a saliency detector. A comparative analysis evaluates various models on a simple input image time series task, focusing on their effectiveness in training and out-of-sample validation data.

Results indicate that while reservoir models exhibit higher training data loss compared to non-reservoir networks, they outperform other models on new data. Modifications to the reservoir structure show promise in improving both loss and Intersection over Union (IoU) score performance on the training and out-of-sample validation data. However, the study does not find that imposing a more visual cortex-like structure in the reservoir enhances its performance.

Further analysis suggests that while reservoir models offer advantages such as cost-effectiveness and robustness for changes in input data, they may lack the precision of more trainable models like CNNs. Limitations to the thesis include the scope of hyperparameter exploration and the lack of overfitting mitigation techniques like the use of dropout.

Future research should focus on refining all models to measure the potential performance on both training and out-of-sample data. This includes exploring ensemble methods and improving weight initialization techniques to enhance model precision and adaptability, leading to advancements in various real-world applications.

Sammendrag

Denne studien utforsker ytelsen til reservoartype og konvensjonelle kunstige nevrale nettverk som saliensdetektorer, inspirert av Li Zhaopings hypotese angående V1s rolle som en saliensdetektor. Sammenlignings analyse evaluerer ulike modeller med en enkel tidsrekke for inngangsdata, med fokus på modellenes effektivitet i trening og validering av data utenfor treningrommet.

Resultatene indikerer at mens reservoarmodeller viser høyere tap på treningsdata sammenlignet med tradisjonelle nettverk, overgår de modellene på nye data. Modifikasjoner av reservoarstrukturen viser potensiale om å forbedre både tap og ytelse av IoU på trenings- og valideringsdata som er utenfor treningsdata. Imidlertid finner studien ikke at påleggelse av en mer visuell cortex-lignende struktur i reservoaret forbedrer ytelsen.

Videre analyse antyder at mens reservoarmodeller tilbyr fordeler som kostnadseffektivitet og robusthet for endringer i inndata, kan den mangle presisjonen til mer trenbare modeller som CNN-er. Begrensninger i avhandlingen inkluderer omfanget av hyperparameterutforskning og mangelen på teknikker for å motvirke overtilpasning som bruk av dropout.

Fremtidig forskning bør fokusere på å forbedre alle modeller for å måle potensiell ytelse på både trenings- og valideringsdata utenfor treningsrommet. Dette inkluderer utforsking av ensemblemetoder og forbedring av vektinitialiseringsteknikker for å forbedre modellens presisjon og tilpasningsevne, noe som fører til fremskritt innen ulike anvendelser.

Acknowledgements

I am grateful to NMBU for allowing me to write this thesis.

I would like to express my gratitude to my supervisors, Gaute T. Einevoll, Alexander Johannes Stasik and Kosio Beshkov, for their guidance and support throughout the entire process of writing this thesis. Their expertise and patience have been instrumental in shaping this work.

Lastly, I want to acknowledge all my friends who have been a constant source of encouragement. Your friendship has made this journey memorable and meaningful.

Thank you to everyone who has contributed to the completion of this thesis. Your support and encouragement have been invaluable, and I am truly grateful for the opportunity to undertake this research.

Georg Vang
Ås, May 2024

Contents

List of Figures

List of Tables

1 Introduction

Understanding how the human mind perceives the visual world is a multifaceted puzzle. From photons hitting the eye to the intricate computations within the brain, there are a lot of unknowns [1]. Central to this process is the concept of attention selection - the visual cortex's ability to discern what is important in such a big pool of visual stimuli it receives. Li Zhaoping's hypothesis posits that the early visual system, specifically V1, plays a pivotal role in this attention selection process [2]. By computing a saliency map, V1 determines where in the visual field it is crucial to focus our attention.

Advancements in machine learning have led to the development of artificial neural networks, such as the expansive transformation models used in large language models today [3]. These networks, while simplified versions of their biological counterparts, can mimic complex cognitive processes. Their reliance on vast amounts of data poses challenges, particularly in avoiding overtraining - the phenomenon where the model learns the training data too well, compromising its ability to generalize to new data.

A new architecture that promises performance with high resistance to overtraining and less need for training data, is reservoir networks [4]. Unlike traditional artificial neural networks with trainable weights, reservoir networks leverage fixed internal dynamics, with only a small output layer adapting to map reservoir output to desired predictions. This approach aims to mitigate overfitting while promoting robustness and generalization.

Using knowledge from neuroscience and machine learning to explore different artificial neural network architectures the goal is to see how the different models act like a saliency detector. With a traditional way being the CNN model network [5], other models are explored to act as a more possible biological neural network counterpart. Lastly, reservoir-type networks are explored as a potential biological network with a high resistance to overfitting. The reservoir will also be modified to see if a configuration closer to V1 is beneficial or if there are other ways to improve its performance.

1.1 Previous work and Motivation

Primates like humans can recognise objects in scenes with hierarchically interconnected networks in the visual cortex [6]. The likeness to deep neural networks and our visual cortex might be useful to teach us how the neural networks of our visual cortex work. This is because the ability to modify, save and rerun the networks in computer software is much simpler than in an awake primate brain. By making lots of models to replicate a similar task to what the visual cortex does, information about the visual cortex architecture can be learned. Testing multiple architectures can then be compared to each other.

The Convolutional Neural Network (CNN) architecture draws inspiration from studies of biological vision [7]. Research has revealed that certain neurons in the visual cortex respond selectively to specific visual features, such as orientation. This insight served as the foundation for the development of primitive CNNs. Extensive studies have demonstrated that deep CNN architectures can achieve performance levels comparable to human visual recognition capabilities [7]. So using a CNN model would seem like a baseline for making saliency detectors.

The process of determining the significance of various visual inputs mirrors the way our visual cortex prioritizes different regions for focused attention. CNN models excel at capturing and representing these hierarchical visual features, enabling them to simulate human visual processing mechanisms effectively [5]. With the extensive studies done on CNNs and their relationship to the visual system in biology, testing CNNs on a task to find saliency, the important input information, should then also produce similar characteristics as the visual system.

On the other hand, reservoir computing is proficient in learning dynamic systems characterized by intricate temporal and spatial relationships, often with minimal training data [4]. Then having a reservoir as the saliency detector can yield results as minimal previous knowledge is required for the model to make predictions. And if the task of the saliency detector is to find something dangerous, needing previous knowledge might not be an option.

While reservoir computing has demonstrated efficacy, the absence of definitive design rules for the reservoir weights poses a challenge in ensuring optimal predictive performance. The performance of a reservoir heavily relies on the initialization of its weight matrix, a pivotal factor that can determine whether the reservoir functions effectively or yields suboptimal outcomes [8]. Given that all weights in the reservoir are set during initialization, this stage significantly influences the reservoir's dynamics, ultimately shaping its ability to serve as a saliency detector.

Brain experiments show different neurons select for specific orientations in the visual cortex [9]. They also indicate that if the neurons go unused, the connections become weaker. Blakemore and Cooper limited the visual stimuli in the development stages of the visual cortex in cats. The result was that the cats did not recognize objects perpendicular to the stimuli in the environment they had developed their visual cortex in [9]. Thus, if a model is similar to the visual cortex, it would be reasonable to assume similar behaviour in the models as in the visual cortex. Limiting the training data to just one orientation and testing the models on test data perpendicular to the training data should then yield similar results.

2 Theory

Biological neurons have largely inspired the development of machine learning algorithms to make a simplified artificial neuron. As more artificial neurons were combined to form artificial neural networks and with improved architecture, the networks started to be more useful. With more useful artificial neural networks more interest also came to research computer science and computer vision. So these advanced artificial neural networks with similarities to the biological neural networks mean the artificial neural networks can be useful in neuroscience [7]. With the resulting similarities, neuroscience can learn how neural networks work from these machine learning algorithms.

In this thesis, artificial neural networks are trained to function as saliency detectors. A model is a good saliency detector when it can find what is important in the image.

2.1 Neuroscience

This explanation simplifies the visual cortex to give enough information to explain choices made later in the thesis, such as the customization to make the modified reservoir and the *line* dataset.

The primary visual cortex, known as V1 is the region responsible for early visual processing [10]. The signal from the retina goes to the back of the brain. Looking in the primary visual cortex information enters layer 4, which then sends information to layer 2/3. The connections between layers 4 and 2/3 go both ways, with 66% bias towards layer 2/3 [11]. This structure will be attempted to copy in the modified reservoir neural network. Copying the structure will benefit the reservoir if such a structure is beneficial to act as a saliency detector.

Note, that the names of the six layers do not reflect information flow, but on the physical layers of the brain. Layer 1 would be on the surface and then layer 6 would be the deepest layer. Layers 2 and 3 are often not distinguished as they are very similar [2].

Experiments suggest that the V1 needs stimulus in the early stage of development to register that stimulus later in life. C. Blakemore and G. F. Cooper limited the visual stimuli in the development stages of the visual cortex to cats while their visual cortex was in development. After the development of the visual cortex, the cats were released into a normal living room where the cats struggled to perceive edges perpendicular to the visual stimuli they got while in their development stages [9]. So if an artificial neural network architecture is similar to that of V1 it should have a similar response. Therefore if networks are trained on exclusively vertical stimuli, the networks should not be able to find horizontal stimuli under testing. This will be the basis of the *line* dataset.

2.1.1 Neuron

Processing signals in the brain is the work of the neurons. A neuron can be split into three simplified parts; The dendrites are the input to the neuron centre. This is where the neuron receives information about its surroundings. The soma is the centre of the neuron. This is where ion currents enter and the neuron either sends an impulse or not. The end is the axons being responsible for the output. The axons send the signal out of the neuron for a new dendrite to receive [12].

Neurons send signals depending on their voltage over the membrane or *membrane potential* that changes from received signals. A neuron can get excitatory or inhibitory signals, meaning the potential goes up or down depending on the signals they receive, and it is a combination of signals the neuron receives that determines if the neuron sends an output signal or not [12].

2.1.2 Networks of Neurons

Although the mechanics of a single neuron are quite well understood, the mechanics of networks of neurons work are not [1]. Simulations of networks of neurons come to a limit, where they can either simplify the neurons or make the network smaller neither producing an overview of real neural networks. Detailed measuring of neurons in an active brain is also limited, due to the size of the neuron and the sheer size of our brain network [1]. Therefore simplifications can be used to make an artificial neural network in the branch of machine learning.

2.2 Machine Learning

2.2.1 Perceptron

The following subsection follows book [13] closely. McCulloch and Pitts looked at the biological neuron to make *perceptron* [14]. A simple neuron with input signal $\vec{x}$, weights $\vec{w}$ and an activation function $\phi(...)$. Where the output is calculated with the equation

$$\hat{y} = \phi(\vec{w}^\top \vec{x} + b). \tag{2.1}$$

Where $\hat{y}$ is the Perceptron estimate on the label and b is the bias. The problem is finding the optimal weights to solve the given problem. To optimize the weights in equation (2.1), one can use

$$\Delta\vec{w} = \eta(y - \hat{y})\vec{x}. \tag{2.2}$$

Where $\Delta\vec{w}$ is the change in weights, η is the learning rate. y and $\hat{y}$ are the class label and estimated class label respectively [13].

Equation (2.2) provides a simple update function to work towards the optimal weights for one neuron. Thereby being able to linearly separable data. Adding more neurons to make a neural network can separate more complex connections.

An artificial neural network consists of multiple neurons that individually take in a vector (here either the original input vector or the output vector of the previous neural layer) and multiply it with their weights. Then they all get summed up and passed through an activation function. Several stacks of nodes make a deeper neural network, sometimes called a deep neural network. These deep neural networks can be useful in predicting data. One common artificial neural layer is called the dense layer and is just a stack of neurons where all neurons read all the neurons from the previous layer and is calculated with the equation

$$\hat{\vec{y}} = \phi(\mathbf{W}\vec{x} + \vec{b}). \tag{2.3}$$

$\hat{\vec{y}}$ is the predicted label vector, $\mathbf{W}$ is the weight matrix and $\vec{b}$ is the bias vector.

The only change from equation (2.1) to (2.3) is the rank of the variables and the output. To make a deep neural network equation (2.3) is just applied several times. A more advanced weight optimiser was needed for a bigger and deeper network.

Back-propagation optimises the equation (2.2). In back-propagation, the derivative of the activation function is taken backwards from the prediction, and every weight is then nudged in the direction of the lowest error. The amount the neuron gets pushed is determined by the size of the error and the learning rate

$$\Delta \mathbf{W} = -\eta \nabla J(\mathbf{W}). \tag{2.4}$$

Where $\Delta \mathbf{W}$ is the change in the weight matrix, η is the learning rate and $\nabla J(\mathbf{W})$ is the gradient of the cost function with the weights, $\mathbf{W}$ [13].

The result of equation (2.4) is that the weight change is in the direction of the fastest decrease in the cost function, later called loss. The equation is called gradient descent [13].

2.2.2 Convolutional Neural Network

Convolutional neural networks (CNN) use kernels that slide over the input tensor and multiply their kernel weights with the input images underneath. The output of these kernels should therefore contain spatial information about the input image making it more computationally efficient than the standard neurons in the dense layer. The output is then a new tensor called a feature map.

$$\mathbf{F}_{m,n} = \sum_j \sum_k \mathbf{A}_{m-j,n-k} * \mathbf{K}_{j,k}. \tag{2.5}$$

Where $\mathbf{F}$ is the feature map, m and n are the indexes for the feature map, j and k are the indexes for the kernel, $\mathbf{A}$ is the input image and $\mathbf{K}$ is the kernel. More about the CNN network can be read in [13].

2.2.3 Recurrent Neural Network

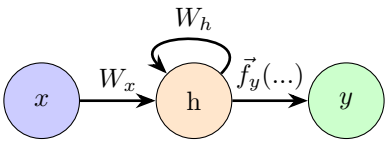

Figure 1: Simplified node representation of a Recurrent Neural Network (RNN) architecture with input weights, recurrent weights and the activation function

Recurrent neural networks (RNN) are as the name suggests, recurrent. Recurrence means that the network layer can receive connections from its layer in the previous time step. The connection from the previous timesteps makes the neurons able to take their past into account and is therefore especially good at handling time series data. The RNN can be explained mathematically with the formula

$$\vec{h}_{t+1} = \vec{f}_h(\mathbf{W}_x \vec{x_t} + \mathbf{W}_h \vec{h_t} + \vec{b_h}). \tag{2.6}$$

Where $\vec{h}_{t+1}$ is the hidden state vector at time step $t+1$, $\vec{f}_h(...)$ is the activation function for the hidden state, $\mathbf{W}_x$ is the weight matrix for the input vector $\vec{x_t}$ at time step t, $\mathbf{W}_h$

is the weight matrix for the previous hidden state vector $\vec{h_t}$ and $\vec{b_h}$ is the bias vector for the hidden state. Figure 1 shows a node diagram off the RNN. The hidden state is the next layer input.

2.2.4 Reservoir Networks

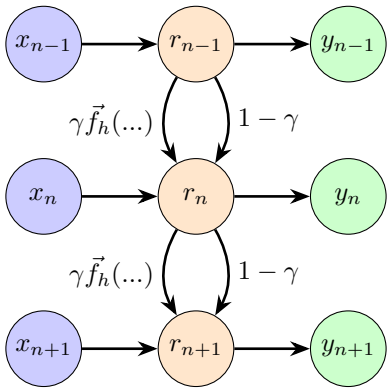

Figure 2: Simplified node representation of an unfolded reservoir

The Reservoir network in figure 2 is based on [8] and is similar to an RNN network. The main difference is that the reservoir weights are not updated, instead having a small trainable layer after the reservoir to map the reservoir state to the output. This implementation has also added a pass-through rate to equation (2.6) making the updated equation for the state of a simple reservoir.

$$\vec{r}_{n+1} = (1-\gamma) * \vec{r}_n + \gamma * \vec{f}(\mathbf{W}_r\vec{r}_n + \mathbf{W}_x\vec{x}_n + \vec{b}). \tag{2.7}$$

where $\vec{r}_{n+1}$ is the next step in the reservoir state vector, γ is the pass-through rate, $\vec{f}(...)$ is the activation function, $\mathbf{W}_r$ is the recurrent reservoir weight matrix, $\vec{r}_n$ is the reservoir state at time step n, $\mathbf{W}_x$ is the input weight matrix, $\vec{x}$ in the input vector at time-step n and $\vec{b}$ is the bias vector.

Note that the weights are not trainable but semi-randomly assigned at initialisation, $\mathbf{W}_r$ is chosen so it is an orthogonal matrix where $\mathbf{W}_r\mathbf{W}_r^\intercal = \mathbf{I}$, this is done with initializers explained more in section 2.2.7.

Keeping the eigenvalues absolute values at one ensures that even with repeated multiplication of $\vec{r}$ some neuron values do not result in zeros or infinity. The activation function also constrains the values of the neurons in the reservoir. $\vec{r}_0$ and $\vec{b}$ are set as small vectors sampled from uniform distributions so that they are not zero nor the main contributor to $\vec{r}$, and γ is a hyper-parameter.

The dense layer is used to map the reservoir to the desired output. The idea is by having a big network that is not trainable but calculates different dynamics in a reservoir, a small trainable network can find relevant patterns in the activity of the reservoir, and the network as a whole has less ability to optimize for the specific problem and has to learn the general rules of the training data. It can therefore be more resilient to changes in the data.

2.2.5 Modified Reservoir

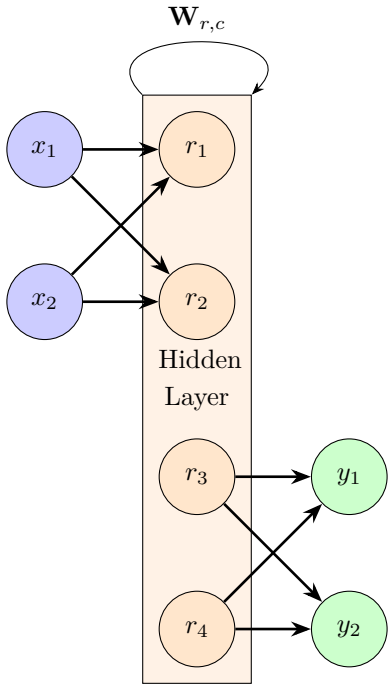

Figure 3: The arrows show the connections of the modified reservoir, the input nodes are only connected to the first half of the hidden layer, the hidden layer is connected to itself with the weights from figure 4 and only the last half of the reservoir is connected to the output.

To make the modified reservoir, the reservoir is split into two parts to mimic the L4 and L2/3 in the visual cortex. This requires custom changes to the input, recurrent weights and output. Already having an input matrix it can be modified with a matrix $\mathbf{W}_{x,c} = \left(\begin{smallmatrix} \mathbf{I} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} \end{smallmatrix}\right)$.

Where $\mathbf{I}$ is the identity matrix, and $\mathbf{O}$ is a zero matrix. Changing the input to be $\mathbf{W}_{x,c}\mathbf{W}_x\vec{x}_n$ in equation (2.7). This results in the input information only affecting the first half of the reservoir. The output can be modified similarly by making a matrix $\mathbf{W}_{y,c} = \left(\begin{smallmatrix} \mathbf{O} & \mathbf{I} \end{smallmatrix}\right)$, reducing the output size to half of the reservoir by only taking information from the second layer.

The modification of $\mathbf{W}_r$ is more involved. To mimic the connections in L4 and L2/3 as in [11], the weights need to make the two layers mostly connected to themselves and somewhat connected to each other. To ensure the stability of the network the weight matrix should have eigenvalues, whose absolute value is not above one. This will ensure that the values in the reservoir $\vec{r}$ do not go to infinity with consecutive self-multiplication. Though a gate of activation function compensates for this, that ensures the values do not supersede the absolute value of one.

By letting $\mathbf{W}_t$ and $\mathbf{W}_b$ be two orthogonal matrices, and placing them in the top left and bottom right corner of weight matrix $\mathbf{W}_{r,c}$ that otherwise is zeros, it is ensured that $\mathbf{W}_{r,c}$ is also orthogonal.

$$\mathbf{W}_{r,c}\mathbf{W}_{r,c}^\top = \left(\begin{smallmatrix} \mathbf{W}_t & \mathbf{O} \\ \mathbf{O} & \mathbf{W}_b \end{smallmatrix}\right)\left(\begin{smallmatrix} \mathbf{W}_t & \mathbf{O} \\ \mathbf{O} & \mathbf{W}_b \end{smallmatrix}\right)^\top = \left(\begin{smallmatrix} \mathbf{W}_t\mathbf{W}_t^\top & \mathbf{O} \\ \mathbf{O} & \mathbf{W}_b\mathbf{W}_b^\top \end{smallmatrix}\right) = \left(\begin{smallmatrix} \mathbf{I} & \mathbf{O} \\ \mathbf{O} & \mathbf{I} \end{smallmatrix}\right) = \mathbf{I}. \tag{2.8}$$

This makes $\mathbf{W}_{r,c}$ a recurrent weight matrix that only the first half of the input will affect the first half of the output, and only the second half will affect the second half of the output. At the same time, both changes will be stable with multiple multiplications.

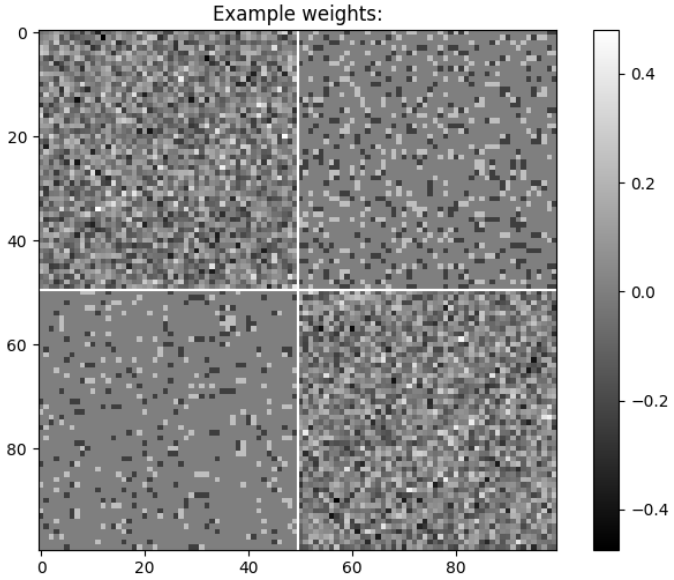

Figure 4: An example of the recurrent weight matrix for the modified reservoir $\mathbf{W}_{r,c}$. Note that the weight density and value are higher on the on-diagonal quadrants than on the off-diagonal quadrants. Note the density is also lower in the bottom left than the top right. This makes the matrix have a forward bias. This example is of a 100×100 matrix.

By populating the zero diagonal with a small number of weights representing the connection between the first and second layer, ensuring that the absolute value of each weight and the amount of weights on the off-diagonal is small and sums to a total zero, the stability is affected but this can be controlled by how much of the off-diagonals have non-null weights and the absolute value of the weights in the off-diagonal.

With all the modifications the new equation for updating the reservoir is

$$\vec{r}_{n+1} = (1-\gamma) * \vec{r}_n + \gamma * \vec{f}(\mathbf{W}_{r,c}\vec{r}_n + \mathbf{W}_{x,c}\mathbf{W}_x\vec{x}_n + \vec{b}). \tag{2.9}$$

Where $\vec{r}_{n+1}$ is the next step in the reservoir state, γ is the pass-through rate, $\vec{f}(...)$ is the activation function, $\mathbf{W}_{r,c}$ is the modified recurrent reservoir weight matrix, an example of this can be seen in figure 4. r_n is the reservoir state at time-step n, $\mathbf{W}_{x,c}\mathbf{W}_x$ is the modified input weight matrix, x is the input and $\vec{b}$ is the bias vector.

The output of the reservoir is being modified with

$$\vec{y} = \mathbf{W}_{y,c}\vec{r}_n. \tag{2.10}$$

Where $\vec{y}$ is the output vector, $\mathbf{W}_{y,c}$ is the output matrix and $\vec{r}_n$ is the state of the reservoir. The nodes resulting from equations (2.9-2.10) are visualized in figure 3 where a smaller example for the modified recurrent weights shown in figure 4.

2.2.6 Convolutional LSTM

Long-Short Term Memory (LSTM) is a version of RNN, where four gates are used to increase stability and give the neurons the ability to remember more [13]. Instead of modifying the RNN to a reservoir, one can modify the recurrent neural network to adapt the features of a convolutional neural network.

First, the equations for a conventional LSTM, equations follow [13]. Being a modification of the RNN explained it has similarities. The hidden state and the input goes to the sigmoid output gate.

$$\vec{o}_t = \sigma(\mathbf{W}_{x,o}\vec{x}_t + \mathbf{W}_{h,o}\vec{h_t} + \vec{b_o}). \tag{2.11}$$

Where $\vec{o}_t$ is the output vector at time-step t. $\mathbf{W}_{x,o}$ and $\mathbf{W}_{h,o}$ are the weights for the input $\vec{x}_t$, and the hidden state $\vec{h_t}$ respectively. The $\vec{b_o}$ is the output gate bias. However, the output vector does not become the output before it does a point-wise multiplication with the internal state.

The internal state makes the LSTM able to remember longer time series. To compute the internal state two more gates are introduced. First, a forget gate to remove old unneeded information, which is a sigmoid of the hidden state and the input,

$$\vec{f_t} = \sigma(\mathbf{W}_{x,f}\vec{x}_t + \mathbf{W}_{h,f}\vec{h_t} + \vec{b_f}). \tag{2.12}$$

With all gates weights and bias equivalent from the output gate in equation (2.11). Then an input gate decides what information should be added to the internal state from an input node, the input gate is calculated with this equation

$$\vec{i_t} = \sigma(\mathbf{W}_{x,i}\vec{x}_t + \mathbf{W}_{h,i}\vec{h_t} + \vec{b_i}). \tag{2.13}$$

To calculate the input node, the equation uses hyperbolic tangent as the activation function

$$\vec{n} = tanh(\mathbf{W}_{x,n}\vec{x_t} + \mathbf{W}_{h,n}\vec{h_t} + \vec{b_n}). \tag{2.14}$$

With the input node weights and bias equivalent from the output gate equation (2.11). Finally, the internal state can be updated with the equation

$$\vec{c}_{t+1} = \vec{f_t} \odot \vec{c}_t + \vec{i_t} \odot \vec{n}. \tag{2.15}$$

Where $\vec{c}_t$ is the internal state, $\odot$ is element-wise multiplication, $\vec{f_t}$ and $\vec{i_t}$ are the vectors from the forget gate from equation (2.12), and the input gate from equation (2.13). $\vec{n}$ is the vector from the input node from equation (2.14). With the internal state calculated in equation (2.15), the hidden state can be updated for the next time-step

$$\vec{h}_{t+1} = \vec{o}_t \odot tanh(\vec{c}_{t+1}) \tag{2.16}$$

Where $\vec{h}_{t+1}$ is the hidden state at time-step $t+1$, $\vec{o}_t$ is the vector from the output gate from equation (2.11) and $\vec{c}_{t+1}$ in the internal state for the time-step $t+1$.

To modify the LSTM to be a convolutional LSTM the input vector becomes an input matrix, and the neurons in the LSTM are used as the CNN as in the equation (2.5). To adapt to the input matrix all the tensor parameters in the normal LSTM have to be one rank higher. This configuration of the neuron model is a more integrated version of the combination model CNN-RNN so more complex spatial information can be processed, more info about the convolutional LSTM can be found in [15].

2.2.7 Weight Initializers

The correct choice of weights is crucial for a performing model, especially for the reservoir-type models that can not update their weights afterwards [8].

A random uniform initializer was used for the initialization of the parameters of the reservoir. The probability is calculated with the formula

$$P = \frac{1}{b-a}. \tag{2.17}$$

Where P is the probability distribution, b and a are the maximum and minimum values of the weights.

Equation (2.17) shows that the probability of the weights is the same for all possible values. This distribution was chosen to make the bias and start state of the reservoir only have a small influence on the reservoir's output.

The recurrent weights need stability with repeated multiplication therefore the orthogonal matrix weight initializer was chosen. The orthogonal initializer generates an orthogonal matrix with the QR decomposition of a matrix of random numbers drawn from a normal distribution. The output will have orthogonal columns [16]. This is expressed in the following equation

$$\mathbf{A} = \mathbf{QR}. \tag{2.18}$$

Where the matrix $\mathbf{A}$ is made to the set size with random numbers, then using the Gram–Schmidt process it gets decomposed to $\mathbf{QR}$, where $\mathbf{Q}$ is the orthogonal matrix and $\mathbf{R}$ is an upper triangular matrix [17].

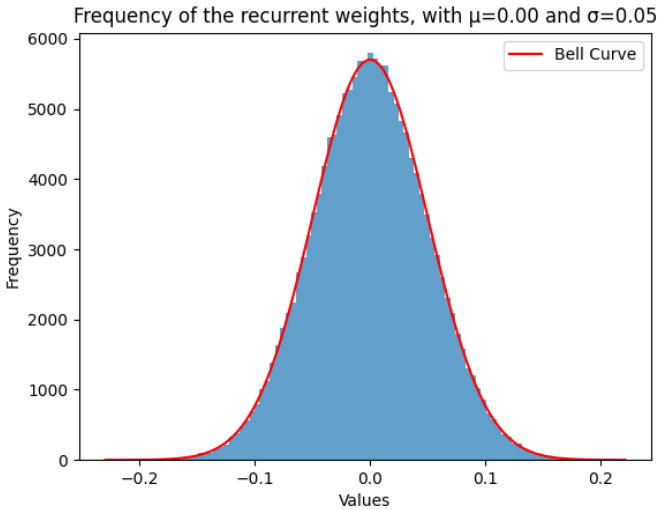

Figure 5: Shows the recurrent weight distribution used in the reservoir-type networks, the red line is the normal distribution with a mean of zero and a normal distribution of 0.05.

Figure 5 shows the distribution of weights from $\mathbf{Q}$ from equation (2.18), and is the one used for the reservoir models. Here it is also shown that the resulting orthogonal matrix follows closely a normal distribution that is indicated with the red bell curve.

For the input weights to the reservoirs, the Glorot normal initializer was chosen. The Glorot normal initializer generates a truncated normal distribution with a mean at zero with the standard deviation calculated with the equation

$$\sigma = \sqrt{\frac{2}{W_{in} + W_{out}}}. \tag{2.19}$$

Where W_{in} is the number of inputs for the weight and W_{out} is the number of inputs for the weight [18]. One benefit found with using this distribution is to give faster convergence as it has more balanced gradients.

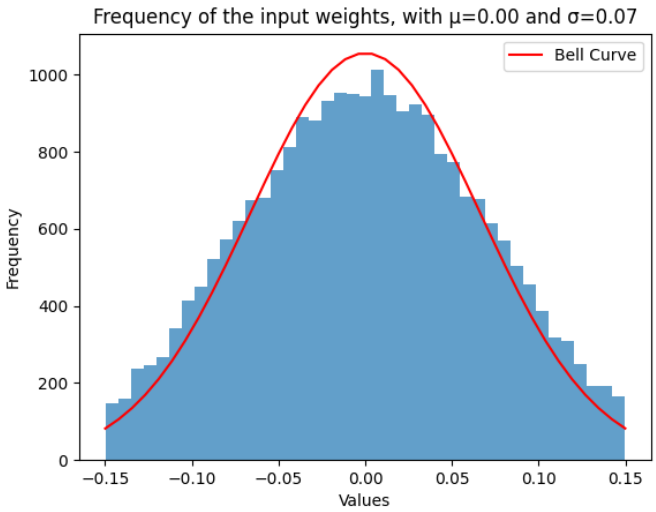

Figure 6: Shows the input weight distribution used in the reservoir-type networks. The distribution comes from the Glorot initializer. The red line is the normal distribution with a mean of zero and a normal distribution of 0.07.

As seen in figure 6, the Glorot normal gives a flatter distribution than the normal distribution with the standard deviation from equation (2.19). This truncated distribution means more weights have values further away from zero and fewer values at or near zero than what would have been expected from a normal distribution.

2.3 Metrics

Several metrics give a bigger picture of how the models act under training. The model needs a metric to optimise the trainable weights in the model, for this loss will be used. Although computers easily read loss it is not an easily understandable metric for humans as it is only comparable to different models doing the same task. To get an understanding of the model's performance, some human-readable metrics will be used as well.

2.3.1 Loss

The metric to minimise is loss. The model uses a mean square error

$$MSE = \frac{1}{n} * \sum_{i=1}^{n} (\mathbf{Y}_i - \hat{\mathbf{Y}}_i)^2. \tag{2.20}$$

Where n is the number of samples, $\mathbf{Y}_i$ is the true value and $\hat{\mathbf{Y}}_i$ is the predicted value [13].

Since the object is small in the input, the contribution to the loss is much greater from the non-object part of the input than the object part of the input. Mitigating the focus can be done by applying a bias to where the object is, so the contribution from the errors in the object is the same as the non-object portion.

$$Weighted\ MSE = \frac{1}{n * k} * \sum_{j=1}^{k} \sum_{i=1}^{n} \delta_{i,j} (\mathbf{Y}_{i,j} - \hat{\mathbf{Y}}_{i,j})^2. \tag{2.21}$$

Where n is the number of rows, k is the number of columns, and $\delta_{i,j}$ is the inverse of the fill factor if the object is at $i,\ j$. $Y_{i,j}$ is the true value at $i,\ j$ and $\hat{Y}_{i,j}$ is the predicted value at $i,\ j$.

Equation (2.21) can be easily calculated by calculating the fill factor, ff. Then weighted the errors, $Weighted\ SE$ to then take the average to get the $Weighted\ MSE$

$$ff = \frac{1}{n_T} * \sum_{i=1}^{n_T} Y_i. \tag{2.22}$$

$$Weighted\ SE = (\frac{1}{ff} * \mathbf{Y} + (1 - \mathbf{Y})) * (\mathbf{Y} - \hat{\mathbf{Y}})^2. \tag{2.23}$$

$$Weighted\ MSE = \frac{1}{n_T} \sum_{i=1}^{n_T} Weighted\ SE_i. \tag{2.24}$$

Here the n_T is the number of entries in the tensor $\mathbf{Y}$, $\mathbf{Y}$ is the true labels tensor and $\hat{\mathbf{Y}}$ is the predicted tensor from the model. Note that $Weighted\ SE$, $\mathbf{Y}$ and $\hat{\mathbf{Y}}$ are tensors of rank 5.

$Weighted\ MSE$ from (2.24) gives a good metric for telling the code what to minimise but does not give insight into how well the model is at predicting.

2.3.2 Intersection over Union

A good human-readable metric goes from zero to one, with one being perfect. The metrics used here are taken from Keras, more can be read in citation [19] or in Keras documentation.

One metric that does this is Intersection over Union (IoU) which is simply calculated by

$$IoU = \frac{TP}{TP + FP + FN}. \tag{2.25}$$

Where TP are the true positives, FP are false positives and FN are false negatives. The name explains that the intersection is where the prediction and the true values overlap, over (or divided by) the union of the predicted and true values. An example of this can be seen in figure 7.

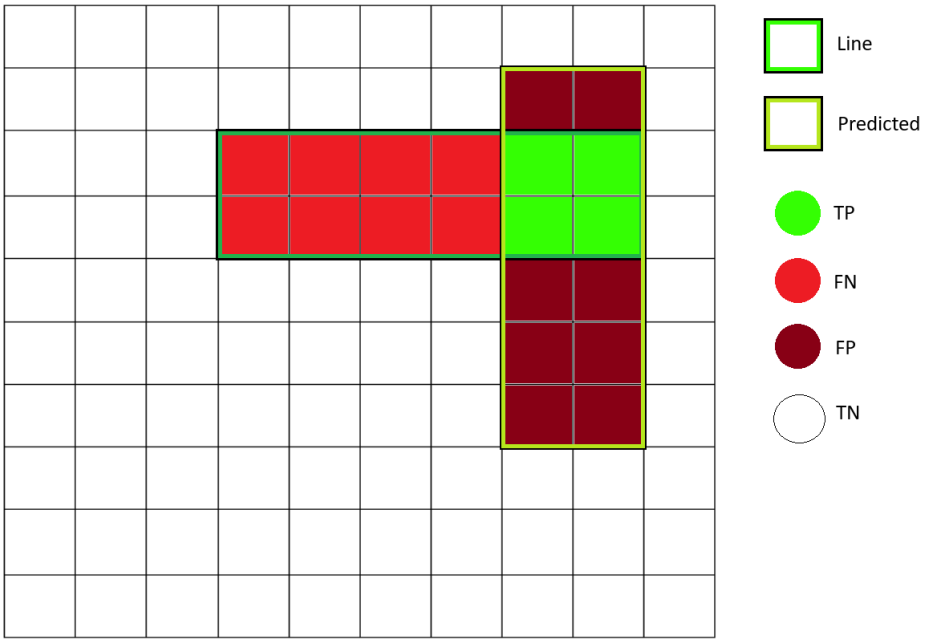

Figure 7: An example with a picture size 10×10 and a line that is 2×6. The pixels are labelled TN, FN, TP and FP, meaning true negative, false negative, true positive and false positive respectively. The prediction placed the line there 4 pixels are true positives, 4 false negatives and 4 false positives, using equation (2.25) it gets an IoU of 0.33

One problem with Intersection over Union is that it is a classification metric, meaning it expects a one or a zero. The model's output is somewhere in between reflecting how certain the model is for that specific pixel. To fix this a cutoff was implemented at 0.5, meaning if one pixel is at 0.6 this is read as a positive for the classification metrics. Getting an IoU below one does not tell what kind of mistake the model makes, precision and recall do.

2.3.3 Precision and Recall

Precision and Recall are also classification metrics. They will use the same cutoff for positive and negative labels. Precision asks; given any predicted positive pixel, what is the probability that it is a true positive? This can be written as

$$Precision = \frac{TP}{TP + FP}. \tag{2.26}$$

Where TP are true positives and FP are false positives.

Precision only measures true and false positives therefore a prediction that is a small subset of the real positives gives a high precision score even though a lot of the real positives can be falsely predicted as negative.

Recall, on the other hand, asks given any real positive pixel, what is the probability that it is predicted to be positive? This can be written as

$$Recall = \frac{TP}{TP + FN}. \tag{2.27}$$

Where FN are false negatives. The recall score falls with false negatives. Calculating recall from figure 7, recall becomes $\frac{4}{4+4} = 0.5$.

Precision or recall alone does not give a good metric for how good the model is, but together they make a clearer picture of the model's behaviour.

2.4 Smoothing kernel

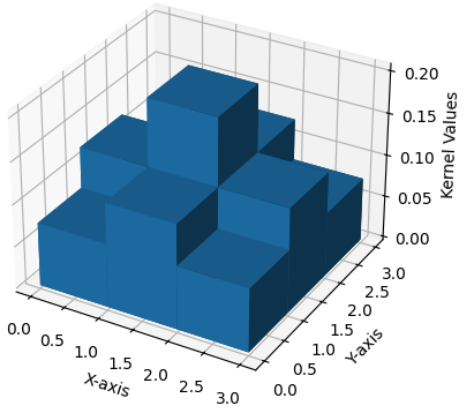

Figure 8: A visualisation of the averaging kernel that is made from in equations (2.28-2.29) to make the input have less noise.

To reduce the noise in the input picture a kernel is made. With a simplified normal distribution, the noise reduction kernel is

$$P(x, y, \sigma) = exp\left(-\frac{(x-1)^2 + (y-1)^2}{\sigma}\right). \tag{2.28}$$

Where $exp(...)$ is the exponential function, x is the x-axis, y is the y-axis and σ scales the sharpness of the kernel [20].

To normalize the function, normalization is applied,

$$kernel = \frac{P(x, y, \sigma)}{\sum P(x, y, \sigma)}. \tag{2.29}$$

With $P(...)$ coming from equation (2.28).

Equation (2.29) ensures that the sum of the matrix before and after the kernel is the same.

3 Method

3.1 Use of AI tools

In the process of writing the thesis help from language models was used. Flowcharts and code are made with the help of ChatGPT 3.5 and spellchecking was helped with Grammarly.

3.2 Code Explanation

To compare various artificial neural network architectures, it was necessary to produce both input and the desired output data, alongside metrics enabling a comparison between the models. This is done with Python and the code is available on GitHub. More details on hardware and dependencies are in Appendix B.

Models were made to check their ability to be saliency detectors. They were trained with input data checked against the desired output data. How the data were made, the metrics implemented and the training done is explained. The choices for how the artificial neural networks are made are also in this section

3.2.1 Input and Desired Output Data

The input data to the models are a time series of different images where the object's location is constant and becomes clearer with each step, with a linear fade. The background is just noise made from uniformly random values between zero and one and smoothed over by the kernel shown in figure 8 and there is a new background for every time step. Many parameters can be changed in the input data, the table A.1 in Appendix A shows the variables.

After the background is made the object of interest is placed. There is no line in the first step (step 0), whereas from step 1 to step 9, the line gradually becomes opaque with pixel values at zero. The line's position is chosen randomly so the line's top left corner has an equal probability of being in every possible position so that the whole object fits inside the input.

The desired output data is just the line's position on the image. Step 0 has no line and is, therefore, a blank picture, with the other steps having a line of value one for all pixels the line takes. This time series makes $\mathbf{Y}$ in equations (2.20-2.24).

3.2.2 Metrics

To implement the metrics the were some minimal changes, the loss follows the customizations made in equations (2.22-2.24) to make the loss increase the importance of the line in the predictions. This rapidly increases the convergence of the models.

To better understand the model's behaviour per timestep, a custom class was made to have the IoU score per time step. This was done by separating the timesteps in both the predicted and desired tensors. Then a threshold is applied making the two tensors either ones or zeros. Then the equation (2.25) can be used to provide the metric.

A threshold was applied for precision and recall then equation (2.26) and (2.27) were used respectively. All the metrics were run on the data, averaged per epoch and saved.

3.2.3 Training and Validation

Given the sizes of the datasets the models were trained on, it was beneficial to produce generators. Generators produce the data as it is needed meaning that each batch of data only exists while the model is training on it. Two generates were made, one for training data and the other for out-of-sample validation data.

The out-of-sample validation data is data that is outside the space of the training data and gives an insight into how overspecialized the model has become. An in-sample validation is not measured as it results in the same scores as the training data. The models are run with 20 batches of training data and one batch of out-of-sample validation data, this leaves most of the runtime to training while keeping the uncertainty of the out-of-sample validation data small.

3.3 Choice of Model Architecture

The different models have hyper-parameters that need to be optimised for the best performance, but first, the size of all models was decided at close to one Mb of parameters per model. Parameters like layer size and the total number of layers were optimised by hand and were done without the out-of-sample data so the tuning did not optimise for the out-of-sample validation that is supposed to be unknown data for the model. This was not a complete search of the hyper-parameter space as there are a lot of parameters, the training time is long and due to limited resources. The absolute optimal performance is not the aim of the paper, this is an exploration of different models, and to see how the reservoir type of model compares.

The models implemented and used are;

Model Name	Short explanation
(CNN)	basic CNN model
(RNN)	basic RNN model
(Res)	basic reservoir model
(CNN-RNN)	CNN followed by RNN
(Mod-Res)	Modified reservoir with eq. (2.9)
(Deep-Res)	A reservoir two layers deep
(De2-Res)	Deep-Res where the second layer input is trainable
(CNN-Res)	CNN added in front of the reservoir
(CNN-LSTM)	CNN in front of the convolutional LSTM

Table 1: The names for models used later in the plots for the models.

3.3.1 CNN

The CNN layers do not have an input for time so the network is run for every timestep.

The chosen architecture starts with few but big kernels. This makes each filter see a lot of the picture without adding too many parameters to the beginning of the network. For the next layer, a smaller kernel size and a larger number of kernels were chosen to make a more abstract feature space. This pattern was continued for four layers, then a last layer was added to bring the feature maps back to one dimension, as the output is one colour only.

The padding was chosen to be *same* for all layers in the network, meaning that the size of the feature maps is the same for the whole network.

A U-net type architecture was also tried, this network does not use padding in the same way and instead decreases the size of each feature map to a bottleneck to then up-sample it with decoder steps. The U-net did not achieve a lower loss nor did it train faster. The idea of using a U-net is that they are commonly used in pixel segmentation, a task similar to this problem. More about U-nets can be found in [21].

3.3.2 RNN

The LSTM neurons were used for the network. The input was flattened and then the network starts with an LSTM layer of 110 neurons that can find simple correlations in the input. The second layer is 150 neurons to make the more complex calculations for the network. The LSTM neurons have many parameters so only two layers are needed to reach the size limit.

The networks got lower loss when the neuron number was close to the input size, some other numbers of neurons were tested, but the testing did not show any indication that they would be better in terms of a lower loss. A GRU network was also looked at but did not indicate a lower loss. A GRU network aims to do the same as an LSTM with two gates instead of four, making it more efficient, more can be found in [13].

The dense layer was added at the end of the network to work as the pixels in the output image similar to what is seen in figure 14.

3.3.3 Reservoir

The considerations for optimizing the reservoir differ from the other models, as the reservoir size is only constrained by the size limit for all networks. Therefore the number of neurons is the biggest possible with a dense layer mapping the reservoir to the output. Optimizing the selection of the weights in the reservoir becomes more important, this is where the initializers are used. Different rules for initializing the random weights were tested. Small random distributions for the bias and the start state of the reservoir $\vec{r_0}$ weights gave good results, as they do not overshadow the input. The GlorotNormal was used for the input weights. With the orthogonal initializer used for the recurrent weights. The pass-through rate was not optimized during training.

3.3.4 CNN-RNN

As a combination of two previous networks, the relative size of the two networks is smaller than the basic versions. The CNN-RNN has a lot of different architectural choices that can affect the model's characteristics. Two layers in each network part were chosen to have a network with half CNN and half RNN to get aspects of both types of networks. To keep down the weights in the model max-pooling was used. This parameterless layer takes in a map and returns the maximum in the designated kernel space [7].

The max-pooling layer cuts what is hopefully unnecessary information and makes room for a bigger RNN network. Two layers of LSTM with the same size, followed by a dense layer for the output make the whole network the right number of weights.

3.3.5 Modified Reservoir

The modified reservoir is a one-layer network, so the network size is just the biggest for the size limit. The same hyper-parameters as the reservoir were used so the models could be more easily compared. The strength of the off-axis weights, top left and bottom right quadrants, was optimised by hand. All the interconnected weights were of a constant value with half of them negative so the sum of all values added is equal to zero for ease of implementation.

Other hyperparameters specific to the modified reservoir, which were optimized manually, included determining the number of weights connecting the two halves of the reservoir, as shown in figure 4. The ratio between the feed-forward and feed-backwards was also explored with little difference so the ratio was kept at 33% as in paper [11].

3.3.6 Deep Reservoir

Splitting the reservoir in two by making two separate reservoirs with a feed-forward connection between them makes the deep reservoir. The thought is a deeper reservoir can make more complex connections in the input image without the limiting structure of the modified reservoir. The reservoir performed the best with two identical reservoirs, tests with asymmetrical and deeper networks were tried without a better performance. So two layers both at 250 neurons were chosen.

Two different models came out of this architecture, the first model the dense layer reads from both reservoirs to avoid the loss of information by forcing the input through both reservoirs. This network was named the deep reservoir.

The second model has reservoirs with a trainable layer in between. This deep reservoir has a normal reservoir followed by a reservoir with a trainable input weight matrix. So the information goes into the first normal reservoir and then to a second reservoir with its input weights set as trainable. This resulting network has two distinct layers where the connective weights are trainable, though this network still has no feedback to the first layer like the modified reservoir. This reservoir has a dense layer that reads the last reservoir to the output, this network will be nicknamed deep reservoir 2 or *de2-res* for short.

3.3.7 CNN-Reservoir

Using the same idea from the CNN-RNN the CNN-Reservoir was made with two CNN layers with the first layer having a big kernel, and the second layer having double the filters. Max-pooling was used for the same reason as the CNN-RNN. The rest of the network is the biggest possible reservoir with a dense layer at 100 neurons at the end to map the reservoir to the prediction. The backpropagation still works even though the reservoir is not trainable as the code still knows the weight matrix values and can go backwards through it.

Another idea of having the CNN-RNN and the CNN-Reservoir similar in architecture is that the later comparison will make more sense. The difference is not in the CNN layer but in whether a reservoir can benefit from being used in an abstract space.

3.3.8 CNN-LSTM

The conv-LSTM is an expensive layer to use in terms of parameters, so delaying the use and starting with a two-layer CNN seems beneficial. Then the conv-LSTM can work in a more abstract space. To avoid upsampling the picture padding of the image was used, more about upsampling is in [13].

Having the same layer amount for both network types and then a layer to collapse the feature space to one dimension, gave good results. The cost of running this model resulted in less hyper-parameter tuning, but the idea was to follow the CNN model to make them comparable.

4 Results

The models are run with the same amounts of time series, not the same training time, the comparison metrics are valid for all models as they are solving the same problem with the same number of matrices. Their learning rate was also kept the same for this reason.

4.1 Performance

Each run has two sets of data, one for training and one to test if the network is becoming overspecialised for training data. With two sets of different data combinations, the first training data is a rotating line of 6×2 resulting in 90 unique line positions and off-validation data that consists of a rotating block of 4×3. This dataset of training and out-of-sample validation data will be called *box*. The second set of training data and out-of-sample validation data consists of a non-rotating line of 6×2 and a non-rotating line of 2×6, this combination will be called *line*.

Every model was trained with the same training parameters: batch size = 500, number of batches per epoch = 20 and epochs = 100, the out-of-sample validation data has the same batch size but only one batch per epoch. These parameters came from being short enough to train multiple models yet have most of the models learn close to saturation. This gives the plots a knee point and is close to showing the peak performance of the models. As an example, these parameters in the *box* dataset have the models see 1,000,000 time series or 10 million individual pictures. Given that there are 90 unique positions for the line, the models have seen all positions for the line about 11 thousand times. Note that the models are evaluated on the block as well, but it can not be said to have seen them as it did not learn from seeing the block.

By splitting the time series and making IoU scores for each step in the time series, the different model's characteristics become clearer. It is also clear that some of the models perform differently on loss, a regression metric, then on IoU, a classification metric. The difference has two main causes, some can be attributed to some models predicting fewer pixels than others because of the loss being heavily weighted for true positives, and giving heavy loss to false positives. This is counter to IoU, as IoU is not weighted at all. The other is uncertainty. If the model is uncertain but still predicts above the cutoff, the loss function will give a higher loss while the IoU score still is high.

The models differ by the type of parameters, the reservoir-type models have non-trainable parameters while the other models only have trainable parameters. It is the total number of parameters that are kept close to each other.

Model	Free parameters
CNN	192,897
RNN	240,900
Res	24,060
CNN-RNN	250,524
Mod-Res	13,560
Deep-Res	30,060
De2-Res	77,560
CNN-Res	20,116
CNN-LSTM	283,028

Table 2: Number of free parameters per model. A short explanation of the models can be found in table 1. Though the total number of parameters should be the same for all models, this is only the trainable weights. Note that for non-reservoir models free parameters and total parameters are the same.

Table 2 shows the free parameter count of the models or the number of trainable weights in the network. The trade-off of having non-trainable weights on the reservoirs means the network should not be able to customize its features as well as the others but the reservoir's mechanics should find the system's dynamics. The CNN model is also somewhat smaller than the other models, this was not seen as a problem as the thesis only aims to indicate if the reservoir is viable. The smaller size of the CNN model seems not to decrease its performance as can be seen in sections 4.2.1 and 4.3.1.

4.2 The Box Dataset

Setting the models to train on finding a rotating line 6×2 and having the out-of-sample validation data as a 4×3 rotating block, the plots in this section show how fixated the models are on the shape they are trying to find, or how easily they become overspecialized.

Training time on the two datasets can be seen in Appendix C, CPU usage was not recorded directly. The memory speed seemed to be the bottleneck for all models except for CNN and CNN-LSTM where the CPU was the bottleneck.

4.2.1 CNN

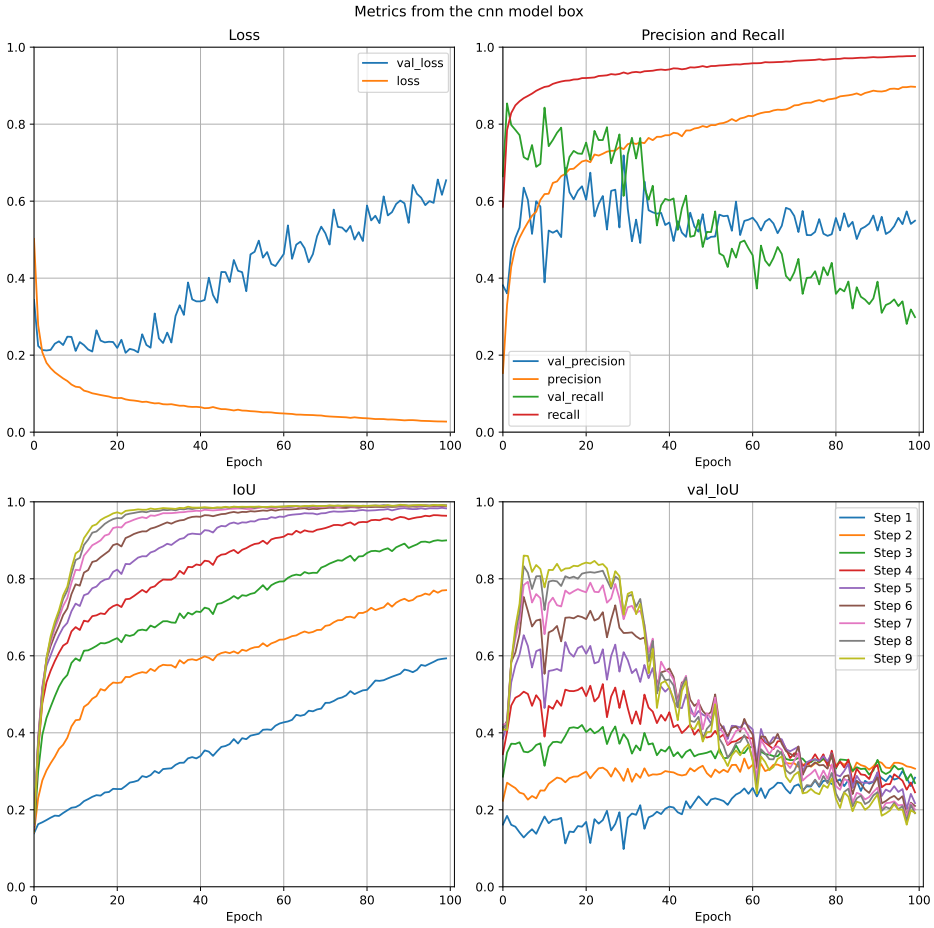

Figure 9: Plots of the CNN model during training, with loss (top left) calculated with equation (2.24). With blue as out-of-sample validation (val) and orange as training data. Precision and recall are in the top right plot with val precision in blue and training in orange. Val recall is in green and training is in red. They are calculated with equation (2.26) and (2.27) respectively. The IoU score is per time-step (step 0 is excluded as it has no true positive) calculated with equation (2.25), the line starts the weakest at step 1 and is fully there at step 9. Training IoU in the bottom left plot and val IoU in the bottom right plot.

Figure 9 shows the measured performance metrics during training, which are split into training data that the model remembers and out-of-sample validation data that the model does not remember. The CNN model does very well, the loss is below 0.05, lower than all other models tested. Both precision and recall are approaching 1 with their knee point somewhere around epoch 5 to 10. The IoU training scores do not bunch up like other models that have a form of recurrence and one clear distinction is in the IoU step 1. All the other models have a low IoU step 1 score while the CNN can get a score of 0.6 with a continued linear increase. It would be reasonable to think that the model is searching for the darker average in the picture and optimising the filters with more epochs.

Looking at the out-of-validation data for the CNN model, the model spikes up fast and keeps the high score for about 20 epochs before overspecializing on the line and achieving a

lower score on the block. Step number 9 starts on top but after 60 epochs it is at the bottom, with the flowing steps scoring higher, suggesting that as the boundaries become more clear the model is more certain that the line is not there. Which is correct, it is a block.

Why the CNN model does so well in a time series can be because the picture is quite small compared to the kernel size used. The big kernels can then imitate the line and calculate where the kernel sum is the darkest per step. If this is the case it would explain why step 9 in out-of-sample validation IoU is lower than the previous steps.

4.2.2 RNN

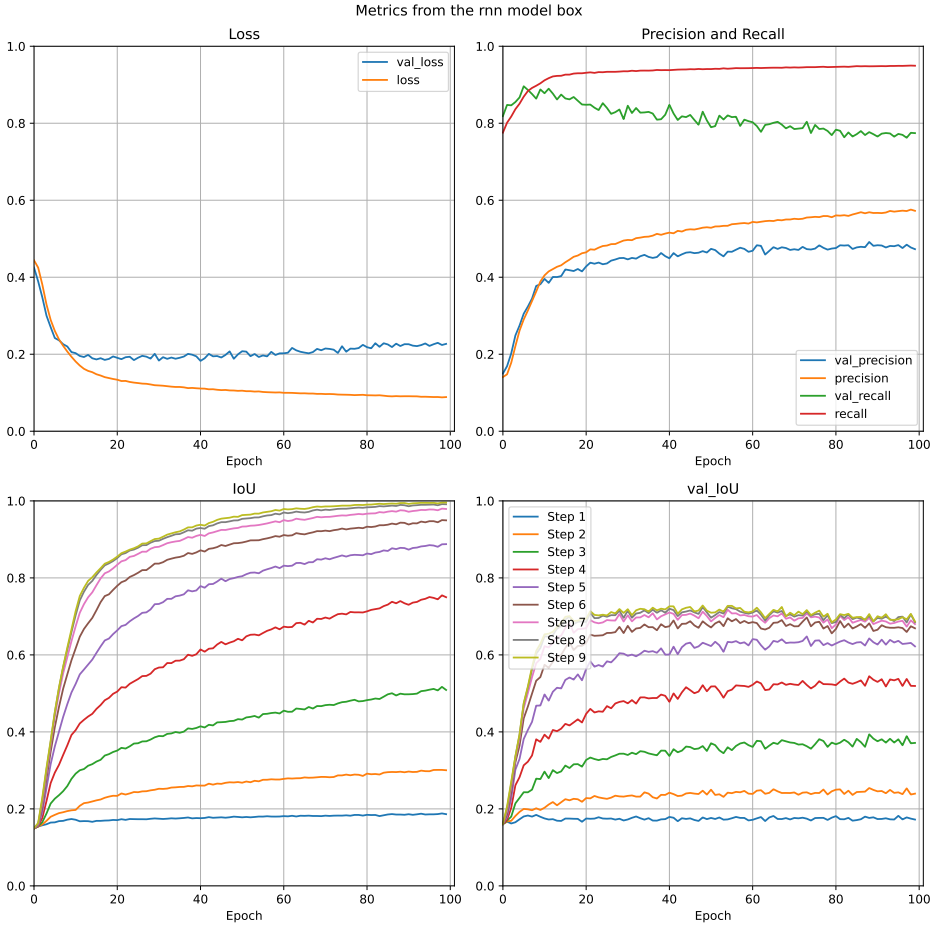

Figure 10: Plots of the RNN model during training, with loss (top left) calculated with equation (2.24). With blue as out-of-sample validation (val) and orange as training data. Precision and recall are in the top right plot with val precision in blue and training in orange. Val recall is in green and training is in red. They are calculated with equation (2.26) and (2.27) respectively. The IoU score is per time-step (step 0 is excluded as it has no true positive) calculated with equation (2.25), the line starts the weakest at step 1 and is fully there at step 9. Training IoU in the bottom left plot and val IoU in the bottom right plot.

The difference between precision and recall is the expected behaviour for all models. Returning to how the loss is weighted means that false negatives are very costly, but overshooting and having false positives are not as costly. The line accounts for half of the loss but is small

compared to the rest of the image (12 %). Giving the true positives/false negatives more than 8 times the importance of false positives.

The training data in figure 10, shows that the RNN model performs worse and learns slower than the CNN model having the loss knee-point at epoch 15. The fact that the RNN model learns slower can be somewhat attributed to the fact that the whole CNN model is training per time step while the RNN model is only training the whole network per 10 steps or one time per time series.

The lowest loss the model reaches is 0.1, significantly higher than the CNN model in figure 9, and the biggest difference is the lower precision of the model. This means that the model predicts the line to be in a bigger area than the actual position of the line. The IoU scores before step 7 are also way lower and there is almost no improvement on step 1, suggesting that the model uses the fact the line stays still and becomes darker and darker to consecutively make a better prediction.

Looking then at the data to see if the model has specialised on the out-of-sample validation data, one can see that the RNN model outperforms the CNN model. The RNN model is more consistent than the CNN model, this is expressed in their out-of-sample validation loss. The RNN model's out-of-sample validation loss also goes below 0.2, something that the CNN model did not. The RNN model's out-of-sample validation loss slowly increases with more training so with more training it does specialize more but not to the degree of the CNN.

The RNN out-of-sample validation IoU score for steps 7 and up keeps the peak value of 0.7 throughout the epochs as opposed to the CNN, suggesting that the RNN has learned something more general about the task than the CNN. The out-of-sample validation IoU score is significantly lower at epoch 20 than the CNN model, the loss is lower for the RNN than the CNN model, this shows the different behaviour of a regression and a classification metric. The RNN model has a fundamentally different calculation method than the CNN model. This becomes clear when looking at the IoU plots broken down per step. While the RNN gains some information per step, the CNN out-of-sample validation has step 9 at the bottom of all IoU scores at the end of the training.

4.2.3 Reservoir

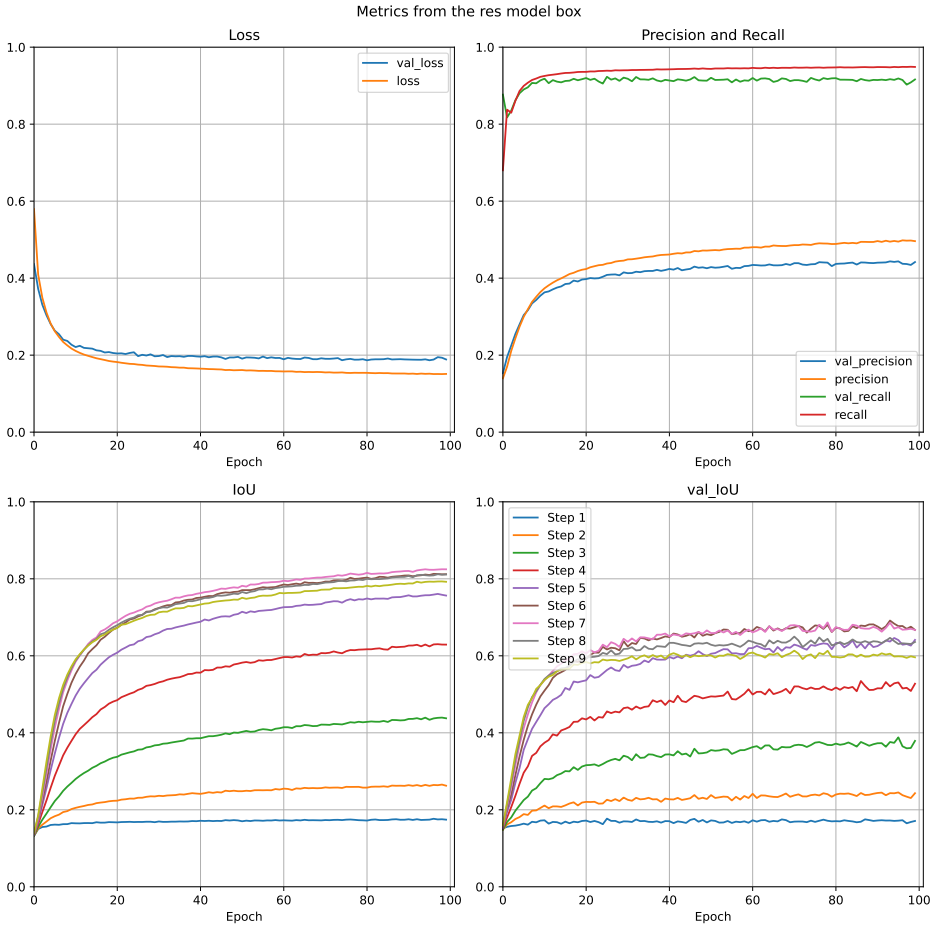

Figure 11: Plots of the reservoir model during training, with loss (top left) calculated with equation (2.24). With blue as out-of-sample validation (val) and orange as training data. Precision and recall are in the top right plot with val precision in blue and training in orange. Val recall is in green and training is in red. They are calculated with equation (2.26) and (2.27) respectively. The IoU score is per time-step (step 0 is excluded as it has no true positive) calculated with equation (2.25), the line starts the weakest at step 1 and is fully there at step 9. Training IoU in the bottom left plot and val IoU in the bottom right plot.

The similarities between figures 11, the reservoir, and 10, the RNN, are clear. This implementation of the reservoir is in principle an RNN layer with frozen weights so this is expected. The RNN did reach an IoU of one while the reservoir stagnated at 0.8. The input dynamics are not represented in the reservoir. Despite the worse training performance, the reservoir keeps its recall higher than the RNN on the out-of-sample validation. This results in a non-increasing loss and a non-decreasing IoU score. This demonstrates the reservoir's resilience to change in data. One interesting quirk of both IoU plots is that the latest step, step 9, is not the one with the highest score. One would assume that the step with the most information is the easiest to predict, but this can be a result of the random initialisation of the model, so the option of having the highest step 9 IoU score would result in a lower score for earlier steps and ultimately in a higher overall loss.

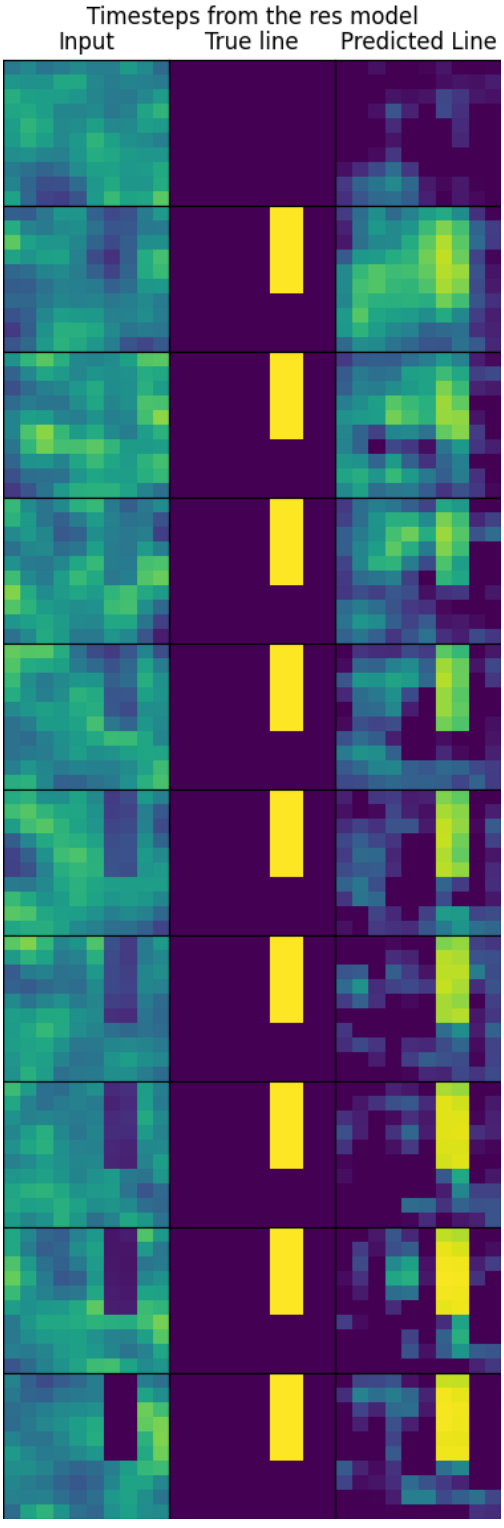

Figure 12: The first column is the input time series. The middle column shows the desired output time series. The last column is the output of the model. The rows show the progression of the line from step 0 to step 9

Figure 12 shows the images the code and the model produce. The first column shows how smooth the input image background is and the line fading in. The second column shows the true label $\mathbf{Y}$ from equations (2.22-2.24) and is the desired output. The last column shows the output of the model, here the rows show the progression of the model as the line becomes clearer.

The first row has no line (step 0) and the model knows this as the model has time as an input. It should predict that there are not any lines in the input image but is not able to produce a black picture with the reservoir weights. The second row (step 1) has a line but the model can not find the line's location. Looking at the input image it is unclear what logic is behind the prediction but it seems to have located some of the darker spots in the image.

Looking at step 2, row number three, the reservoir still can not give a precise location for the line. The model predicts a large block in the middle where the lightest lines are ignored. At step 3 the model should have an IoU score of 0.4, and it has predicted most of the line's position, although it is hard to know what is above the threshold. The limit in the IoU score seems to mostly come from false positives as it predicts a circle next to the line.

The less clear the pixels the more uncertain the model is of the pixels, even in the last steps there is still some uncertainty even though the last row in the first column is clear. This shows that the reservoir struggles to give clear boundaries from start to finish. The input information seems to be too much interfered with in the reservoir to make strong boundary predictions.

4.2.4 CNN-RNN

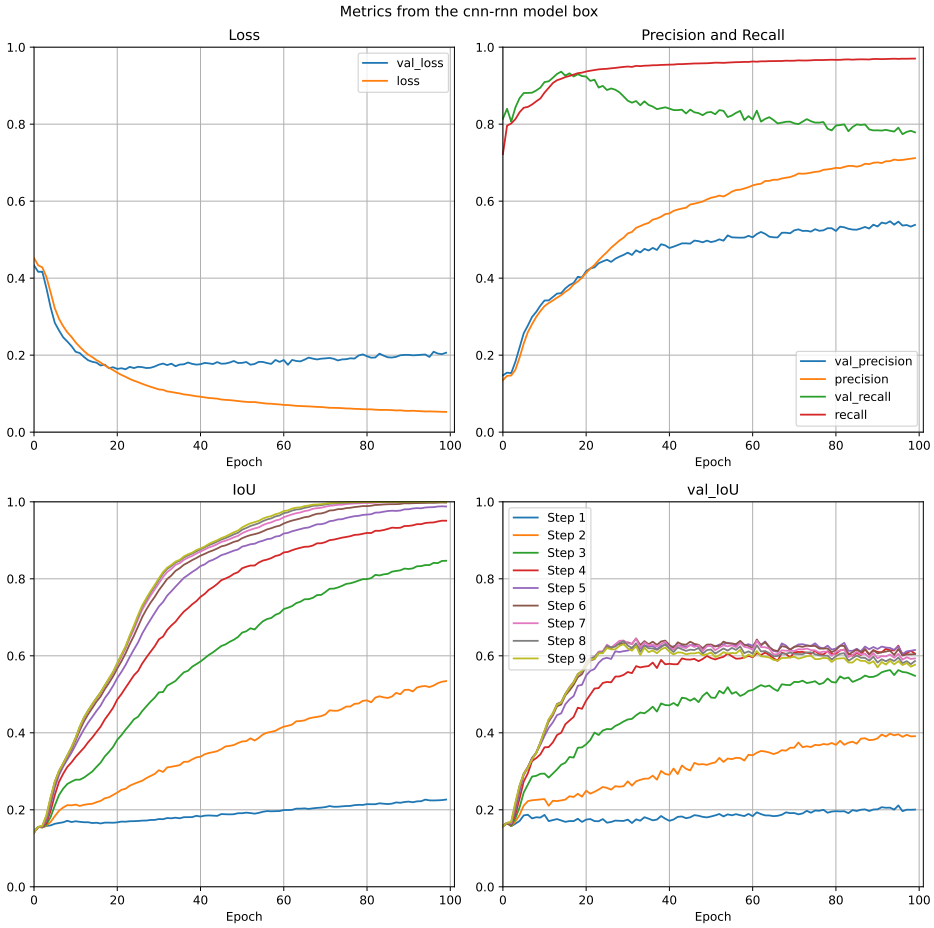

Figure 13: Plots of the CNN-RNN model during training, with loss (top left) calculated with equation (2.24). With blue as out-of-sample validation (val) and orange as training data. Precision and recall are in the top right plot with val precision in blue and training in orange. Val recall is in green and training is in red. They are calculated with equation (2.26) and (2.27) respectively. The IoU score is per time-step (step 0 is excluded as it has no true positive) calculated with equation (2.25), the line starts the weakest at step 1 and is fully there at step 9. Training IoU in the bottom left plot and val IoU in the bottom right plot.

Following the CNN with an RNN network results in figure 13. This results in an interesting plot that is somewhat of a superposition of the networks, and it once again becomes clear that the two networks are training at two different speeds. The IoU score has an early knee point around epoch 10 and then again at epoch 30. The model loss score also ends opp between the CNN and the RNN models.

The CNN-RNN model has a lower recall than the CNN model and does not get the impressive step 1 high IoU score as the CNN but beats the RNN model. Going over to the out-of-sample validation data the combination has a less overspecialized network. It outperforms both individual networks on the loss while not being far worse on the IoU score than the RNN network.

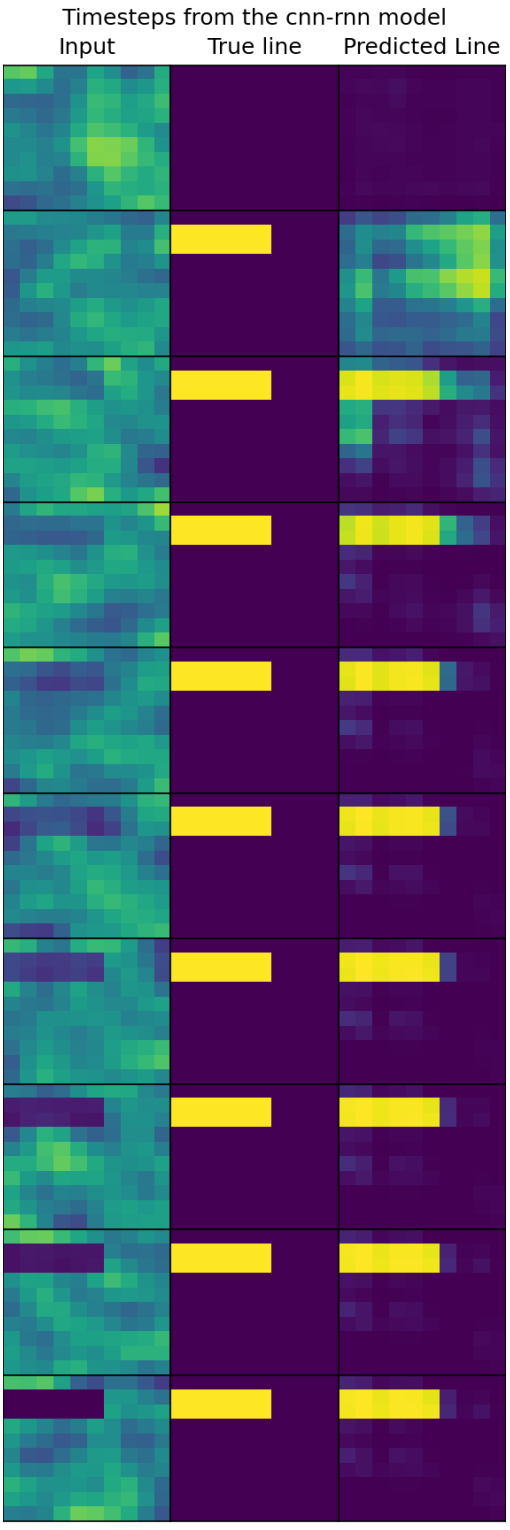

Figure 14: The first column is the input time series. The middle column shows the desired output time series. The last column is the output of the model. The rows show the progression of the line from step 0 to step 9

Figure 14 shows the images the CNN-RNN model produces. Looking at the input image it seems to be looking for the darker edges in the input image and therefore the model predicts two lines one vertical and one horizontal.

By step 2, row number three, the model should have an IoU score of about 0.5. Note this is an average so not necessarily true for this time series. The model has the location of the line but also predicts a horizontal line that if the values are over 0.5 will decrease the IoU score. This miss also shows the mismatch in precision and recall seen in figure 13. The green colour is around 0.5 in value so it is hard to know what pixels are counted by the classification metrics but it seems reasonable to give step 2 an IoU score of around 0.5.

The IoU scores tell us the biggest improvement is in steps 2 and 3, where step 2 looks good enough that the 12 strongest pixels are correct with step 3 removing most of the uncertainty. Even if the classification metrics show a perfect model for one timestep, the loss if it was split per timestep would not show a perfect score of zero for the later timesteps as there are still non-zero values for pixels outside the line.

4.2.5 Modified Reservoir

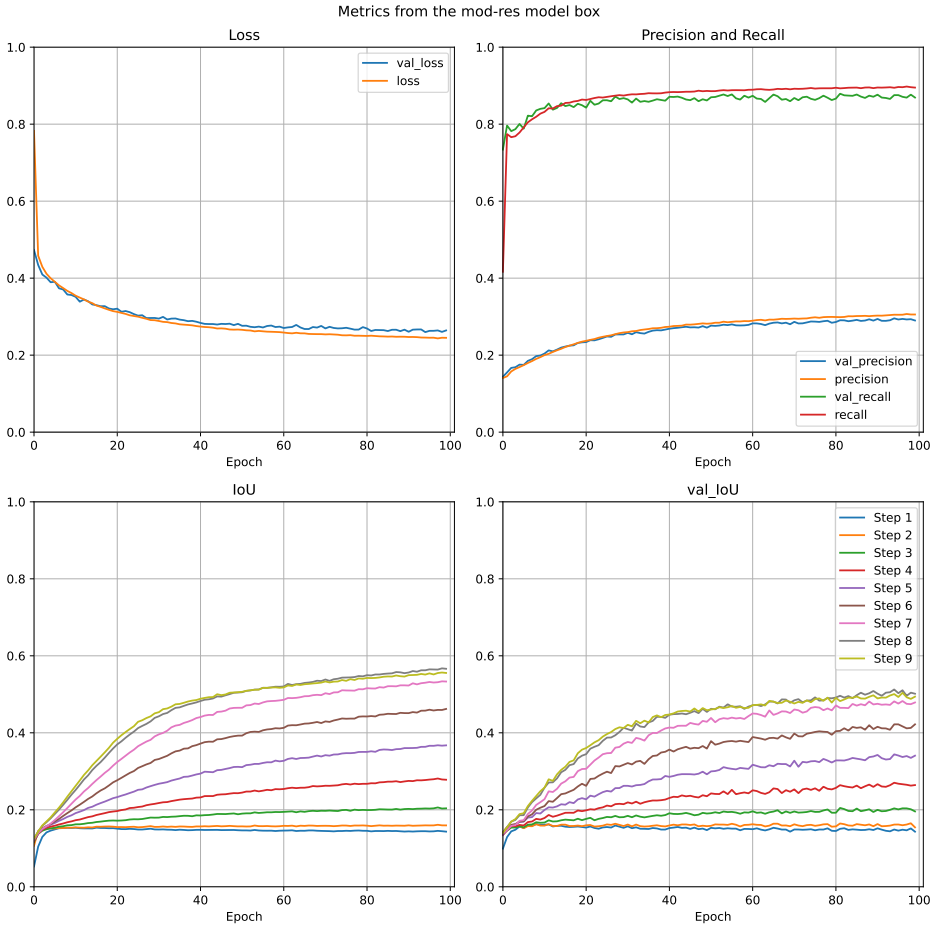

Figure 15: Plots of the modified reservoir model during training, with loss (top left) calculated with equation (2.24). With blue as out-of-sample validation (val) and orange as training data. Precision and recall are in the top right plot with val precision in blue and training in orange. Val recall is in green and training is in red. They are calculated with equation (2.26) and (2.27) respectively. The IoU score is per time-step (step 0 is excluded as it has no true positive) calculated with equation (2.25), the line starts the weakest at step 1 and is fully there at step 9. Training IoU in the bottom left plot and val IoU in the bottom right plot.

Restricting the reservoir model to look like the weights shown in figure 4, gives less trainable weights for the dense layer to map the reservoir's output. This naive approach to making the reservoir weights does result in a working model, although worse than the other models on loss. Figure 15 shows the training progression of the modified reservoir. The difference between the training and out-of-sample validation is very small and more optimized weight rules would result in a better working reservoir. The model learning slower can be because of the difficulty of mapping the reservoir output to a prediction. Learning rate and optimizers are not modified per model to be able to compare them and to the time constraints. The modified reservoir has the highest training loss of all models tested, probably because this model has the smallest amount of trainable weights and the modified reservoir does not seem well fit for the problem at hand.

4.2.6 Deep Reservoir

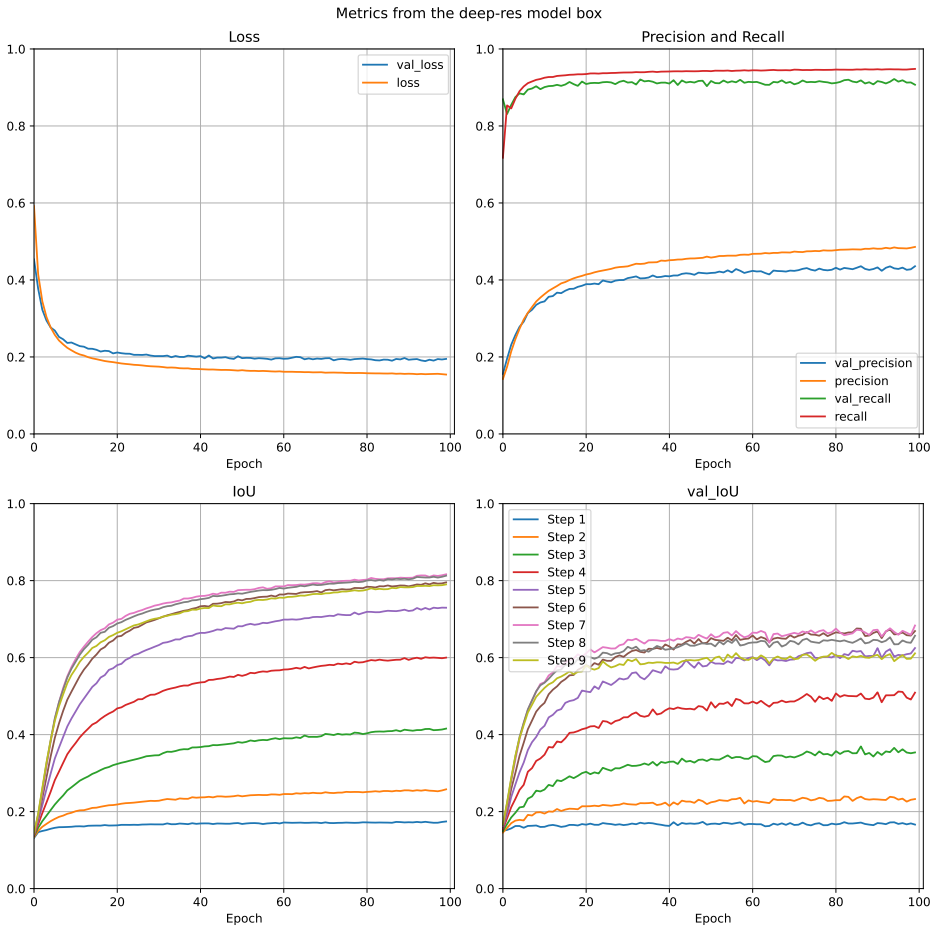

Figure 16: Plots of the deep reservoir model during training, with loss (top left) calculated with equation (2.24). With blue as out-of-sample validation (val) and orange as training data. Precision and recall are in the top right plot with val precision in blue and training in orange. Val recall is in green and training is in red. They are calculated with equation (2.26) and (2.27) respectively. The IoU score is per time-step (step 0 is excluded as it has no true positive) calculated with equation (2.25), the line starts the weakest at step 1 and is fully there at step 9. Training IoU in the bottom left plot and val IoU in the bottom right plot.

Splitting the reservoir into two parts, to create a deeper reservoir so the network can find more complex combinations does not seem to improve any metric as can be seen in figure 16. The change in the network results in more trainable weights in the dense layer. Table 2 shows the number of parameters in the respective models. The increase could lower the loss, but the deeper reservoir produces the same plot as the normal reservoir in figure 11. These changes to the reservoir architecture did not improve the loss or IoU but the second version of the deep reservoir, the deep reservoir 2, tells a different story.

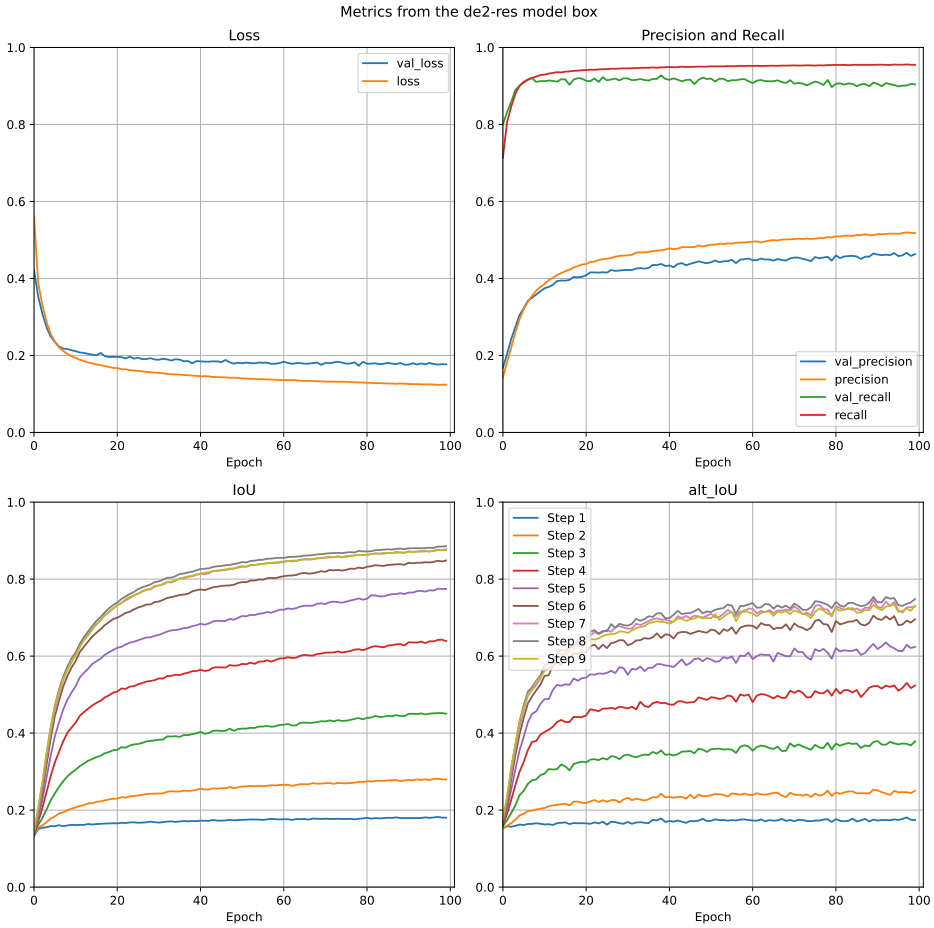

Figure 17: Plots of the deep reservoir model during training, with loss (top left) calculated with equation (2.24). With blue as out-of-sample validation (val) and orange as training data. Precision and recall are in the top right plot with val precision in blue and training in orange. Val recall is in green and training is in red. They are calculated with equation (2.26) and (2.27) respectively. The IoU score is per time-step (step 0 is excluded as it has no true positive) calculated with equation (2.25), the line starts the weakest at step 1 and is fully there at step 9. Training IoU in the bottom left plot and val IoU in the bottom right plot.

Changing the reservoir to the deep reservoir 2 with a trainable connective layer in between improved its performance compared to the normal reservoir seen in figure 11. Figure 17 shows this performance improvement in a lower loss than the reservoir and as higher training IoU scores. The *de2-res* peak Iou training score has increased by one point, to 0.9. Its performance increase comes from both an increase in precision and recall. This increase in training performance has translated to higher validation scores, still showing resistance to over-specialization. The peak IoU validation score of the normal reservoir is 0.7 while the deep reservoir 2 is at 0.75, an increase of 0.05.

4.2.7 CNN-Reservoir

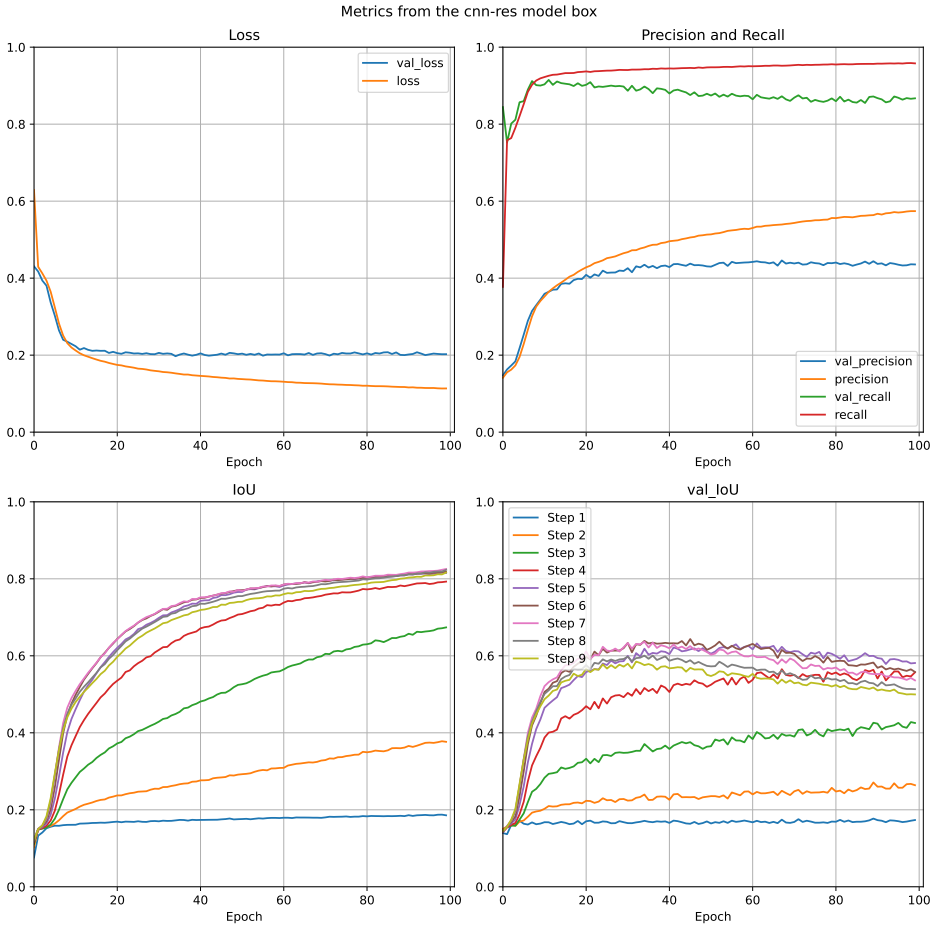

Figure 18: Plots of the CNN-Reservoir model during training, with loss (top left) calculated with equation (2.24). With blue as out-of-sample validation (val) and orange as training data. Precision and recall are in the top right plot with val precision in blue and training in orange. Val recall is in green and training is in red. They are calculated with equation (2.26) and (2.27) respectively. The IoU score is per time-step (step 0 is excluded as it has no true positive) calculated with equation (2.25), the line starts the weakest at step 1 and is fully there at step 9. Training IoU in the bottom left plot and val IoU in the bottom right plot.

Adding a CNN network to the front of the reservoir, but keeping the total network the same size, the network produces figure 18. The CNN-Reservoir improves on the earlier steps compared to the normal reservoir as can be seen in figure 11. It is allowing the network some improvement in training loss. The CNN-Reservoir network does not improve the peak IoU score as it seems the reservoir makes a lot of noise in the output. Borrowing the earlier performance of the CNN network also gave the CNN-Reservoir a worse IoU score on the out-of-validation data. It seems this combination borrowed both networks' good and bad aspects, meaning it overspecializes like the CNN network, and its peak performance is decreased like the reservoir network.

4.2.8 CNN-LSTM

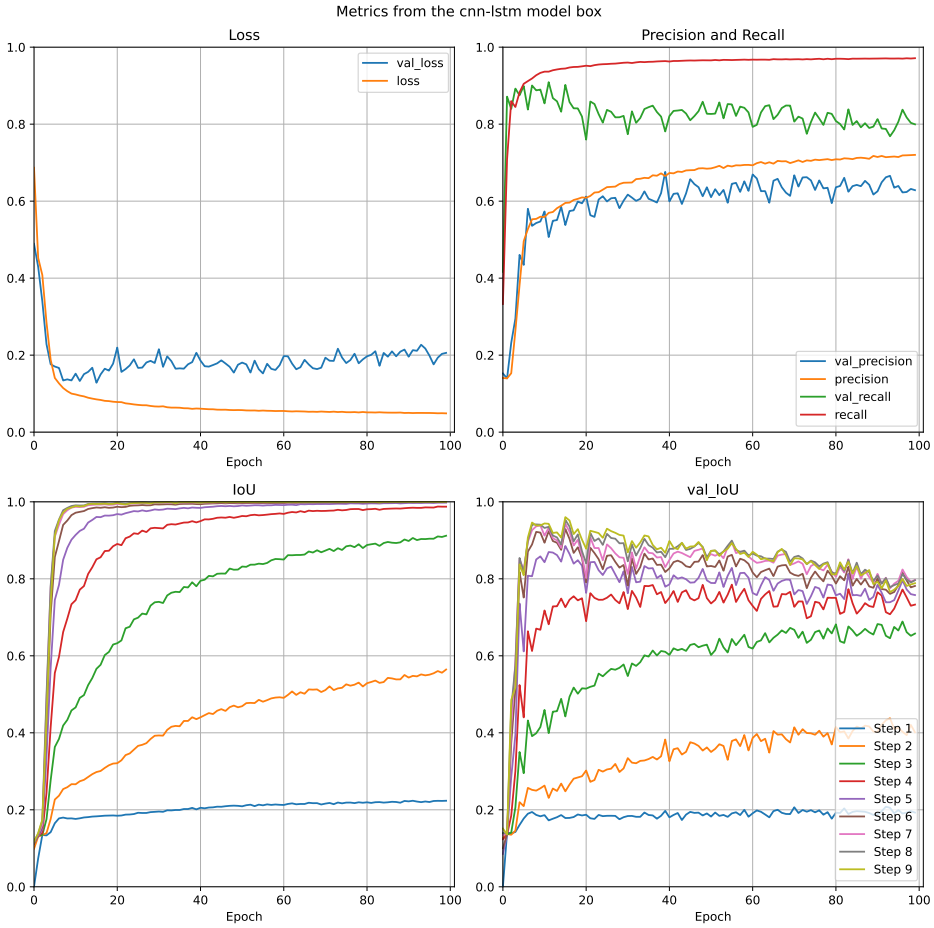

Figure 19: Plots of the CNN-LSTM model during training, with loss (top left) calculated with equation (2.24). With blue as out-of-sample validation (val) and orange as training data. Precision and recall are in the top right plot with val precision in blue and training in orange. Val recall is in green and training is in red. They are calculated with equation (2.26) and (2.27) respectively. The IoU score is per time-step (step 0 is excluded as it has no true positive) calculated with equation (2.25), the line starts the weakest at step 1 and is fully there at step 9. Training IoU in the bottom left plot and val IoU in the bottom right plot.

Changing the CNN-RNN network to a CNN network followed by a convolutional LSTM network does not inherit the early predictions of the CNN network, but beats the CNN-RNN network in terms of loss. Comparing the CNN-LSTM model in figure 19 with the CNN-RNN model in figure 13, the out-of-sample validation data characteristics are similar, with the performance on the out-of-sample validation data being higher on the CNN-LSTM. The model shows the difference between the regression metric, aka loss, and the classification metrics, as the out-of-sample validation data loss is the same as the CNN-RNN while having higher IoU scores. The CNN-LSTM learns extremely quickly, training could have stopped at epoch 10 and had close to saturation results, with the best out-of-sample validation score of all models.

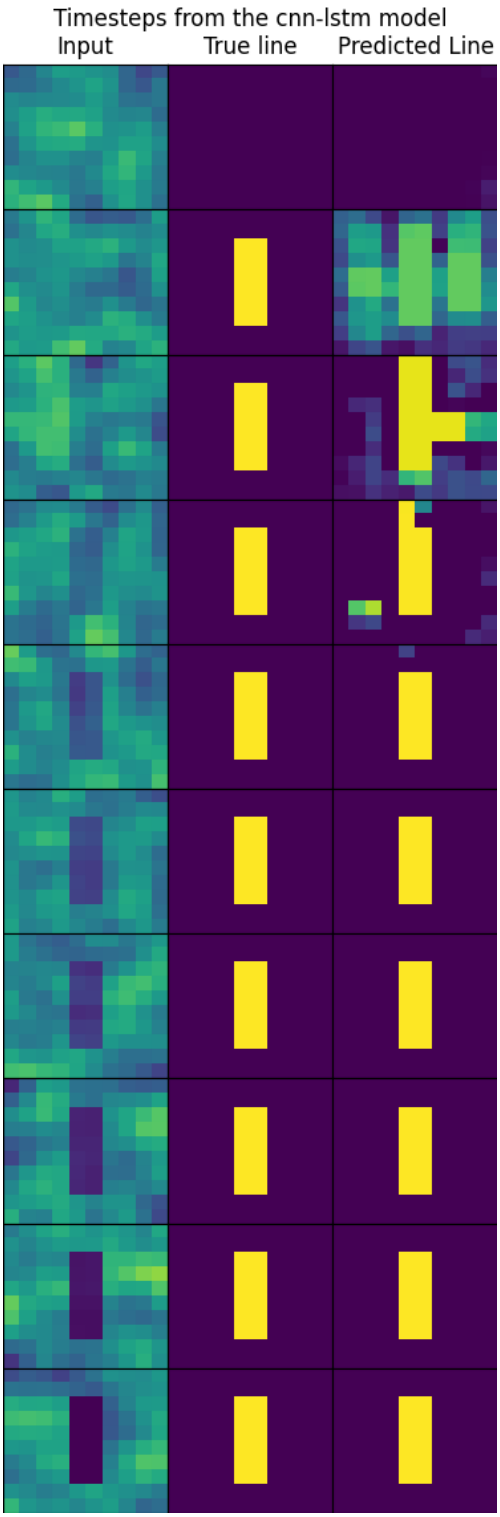

Figure 20: The first column is the input time series. The middle column shows the desired output time series. The last column is the output of the model. The rows show the progression of the line from step 0 to step 9

The CNN-LSTM seems to be the only model to produce a completely blank picture for the first step. This is likely because the last neuron responsible for the output has a time input. The second row has a line that is not visible, yet the model's prediction is in the right location with two other lines next to it. Looking at the input image it seems to be looking for the spots where the summing of the line's shape would produce the lowest sum.

In the third row, step 2, the line is not clear but it is likely in the middle as the model predicts, even though the model predicts a shape that is impossible for the line to be it predicts it with full certainty as if the model has a cutoff in the network. Step 2 predicted output shows a line that is too long but in the right position with a block sticking out, looking at the input it would be reasonable to predict a similar pattern. With all the consecutive steps being clear the model predicts them better till step 4 where it has a perfect IoU score.

From the input it is also possible that the model learns where the model is not, as step 1 has a clear black line for the brightest line in the input image, this then shows up in step two where the model predicts on both sides of the line but is certain that there is not any line in-between, this would also explain the weird half line in step 2 that is too short to be a real line.

The CNN-LSTM model seems to predict a more binary than the CNN-RNN, which is clearer in the later steps of the figure. In step 2 the model makes two distinct line predictions with sharp edges, while the CNN-RNN in figure 14 makes a line prediction with more unclear boundaries.

4.2.9 All models

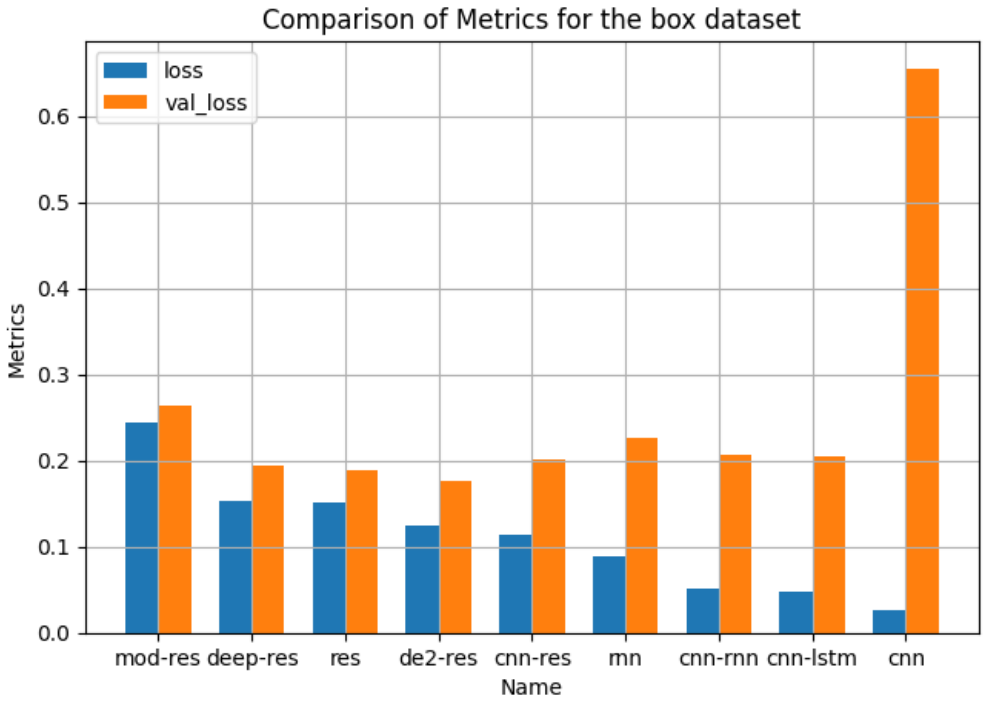

Figure 21: Comparing the loss at epoch 100 of all trained models, where blue is training data and val_loss is the out-of-sample validation. Here the out-of-sample validation is the rotating box

Figure 21 shows the training loss and the out-of-sample validation loss on the *box* dataset, sorted with the highest training loss on the left. The figure shows the Mod-Res performing the worst and the CNN performing the best. The figure shows the pair-wise similarities between the CNN-RNN, CNN-LSTM and the reservoir and the deep reservoir, where the loss and out-of-sample validation is almost the same.

The figure also places all the reservoir-based models on the left and all the non-reservoir-based models to the right, meaning the reservoir-based models all perform worse on training data. There is a different story looking just at the *box* out-of-sample validation loss. All models except for the CNN and the modified reservoir perform very similarly, with the RNN network performing the worst of the good group and the deep reservoir 2 performing the best.

This looks like a minimal difference, as the out-of-sample validation has a bigger uncertainty, but from the IoU plots, it is clear that the deep reservoir 2 is the best out-of-sample detector. The modified reservoir does worse than the rest by a significant margin with a loss of about 0.05 points higher than the group. Worst of all is the CNN model which has a loss of over 0.6, more than double that of the modified reservoir.

4.3 The Line Dataset

Going from line to block does not seem like a big change when looking at the loss in figure 21. Changing the training data to be a non-rotating 6×2 line, and out-of-sample data to be a non-rotating 2×6 line can be a bigger difference. Now that the networks are training on a line that is perpendicular to the out-of-sample validation line.

4.3.1 CNN

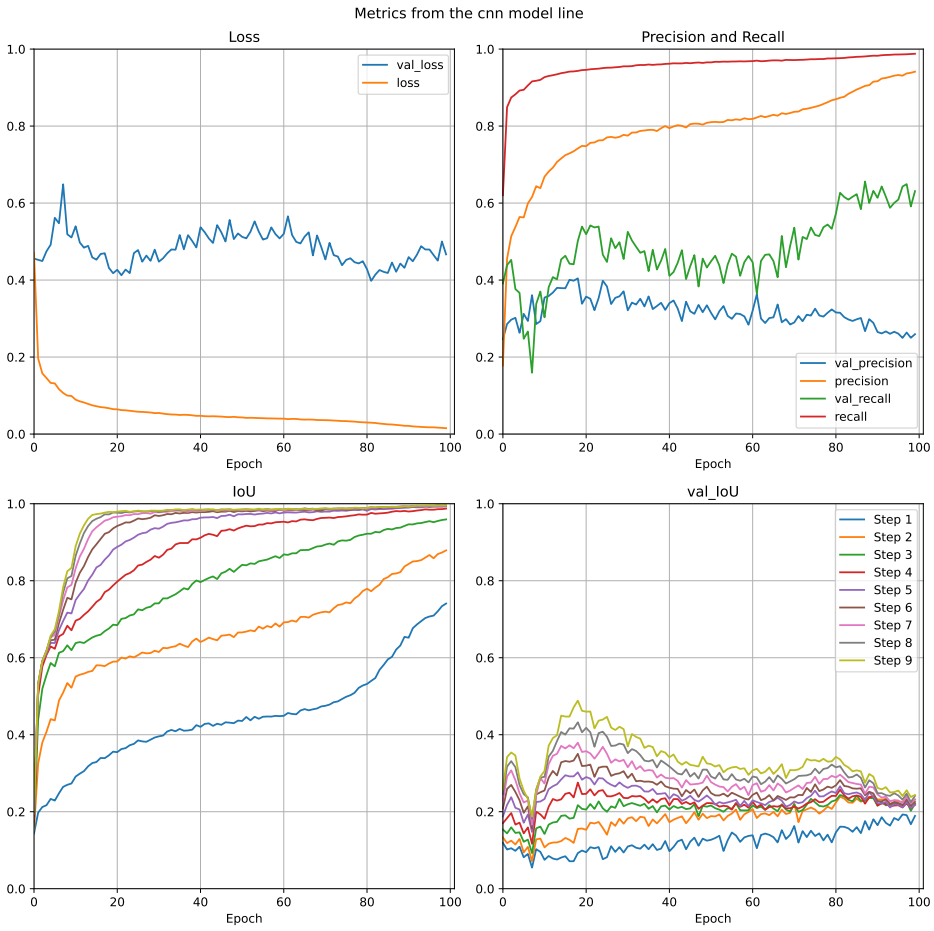

Figure 22: Plots of the CNN model during training, with loss (top left) calculated with equation (2.24). With blue as out-of-sample validation (val) and orange as training data. Precision and recall are in the top right plot with val precision in blue and training in orange. Val recall is in green and training is in red. They are calculated with equation (2.26) and (2.27) respectively. The IoU score is per time-step (step 0 is excluded as it has no true positive) calculated with equation (2.25), the line starts the weakest at step 1 and is fully there at step 9. Training IoU in the bottom left plot and val IoU in the bottom right plot.

Figure 22 shows a faster training of the CNN model on the *box* dataset in figure 9, this is explained by the training data being a subset of the *box* training dataset. Instead of having a linear increase on step 1 like the CNN in figure 9 it has an S shape. The CNN model is exceedingly good on training data, with a 0.75 IoU score on step 1, while steps 3 and up are perfect scores. The CNN model gets a recall above 0.9 around epoch 10 and then increases

the precision with an increase at epoch 70. This increase is not easily seen on the loss plot suggesting that the sharp increase in precision is the model allocating firstly the surrounding pixels with a value just above 0.5 and then lowering it below the threshold.

The increased performance on training data shows the out-of-sample data losses even more. Looking at the metrics from the *line* data, the bigger difference in the datasets means the model can not find the line at the end of the training, and looking at the loss of around 0.5 it does not perform worse, or in other words, it overspecialises with the training data. The CNN has a bump at epoch 20 on IoU scores to a score of 0.5 on step 9, a similar bump can be seen on both dataset combinations. This increase is the model learning some correlation that is more transferable to the out-of-sample validation datasets. It also indicates that this dataset combination is more different than the *box* dataset.

4.3.2 RNN

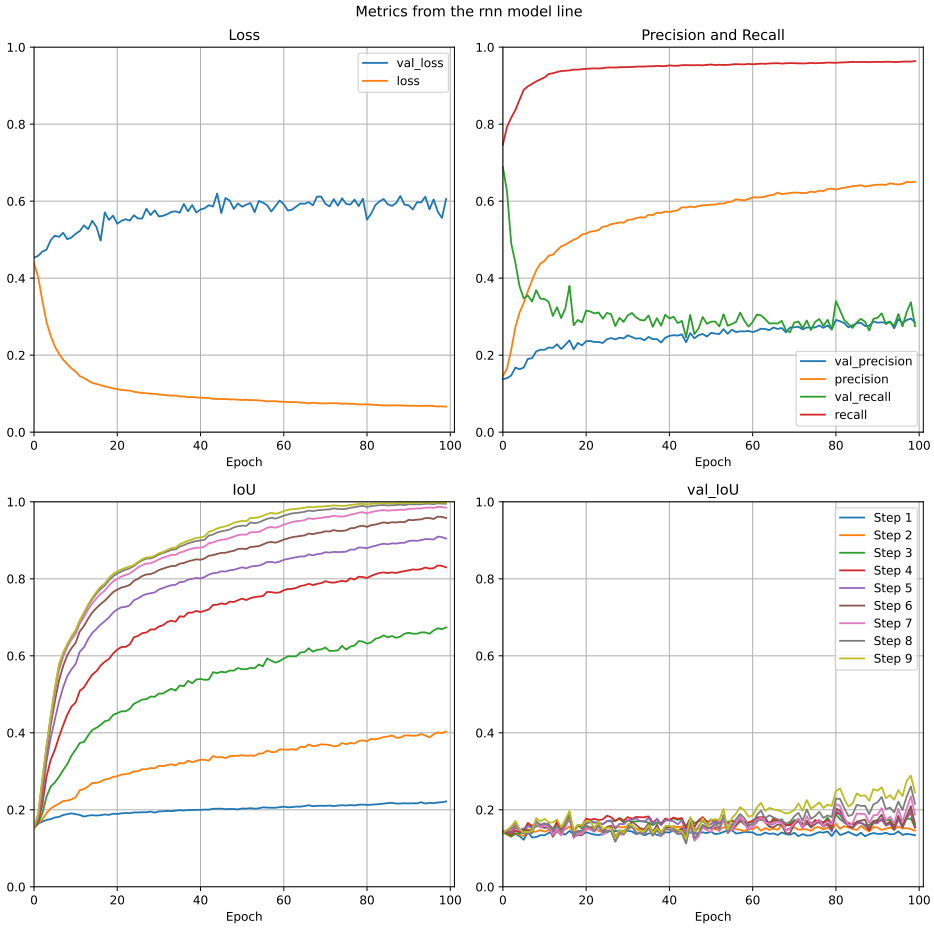

Figure 23: Plots of the RNN model during training, with loss (top left) calculated with equation (2.24). With blue as out-of-sample validation (val) and orange as training data. Precision and recall are in the top right plot with val precision in blue and training in orange. Val recall is in green and training is in red. They are calculated with equation (2.26) and (2.27) respectively. The IoU score is per time-step (step 0 is excluded as it has no true positive) calculated with equation (2.25), the line starts the weakest at step 1 and is fully there at step 9. Training IoU in the bottom left plot and val IoU in the bottom right plot.

The RNN model in figure 23 has a similar but faster increase in IoU scores than the RNN model in figure 10, the RNN model learns the training data faster and gets a lower overall loss in the *line* datasets. Still, the RNN network performs worse than the CNN network on the *line* dataset. On this dataset, however, it is the RNN network that ends with a higher loss on the out-of-sample validation, not the CNN.

The training data on the RNN model is not very interesting as it is very similar to the RNN model on the *box* dataset. The small increase in the IoU out-of-sample validation scores at the very end is different from the other models but it still is a very low score at just 0.3 on step 9. There is not a visible improvement in loss either. It might be that to find the earlier steps in the training data, it has found some connection that also fits somewhat with the out-of-sample validation.

4.3.3 Reservoir

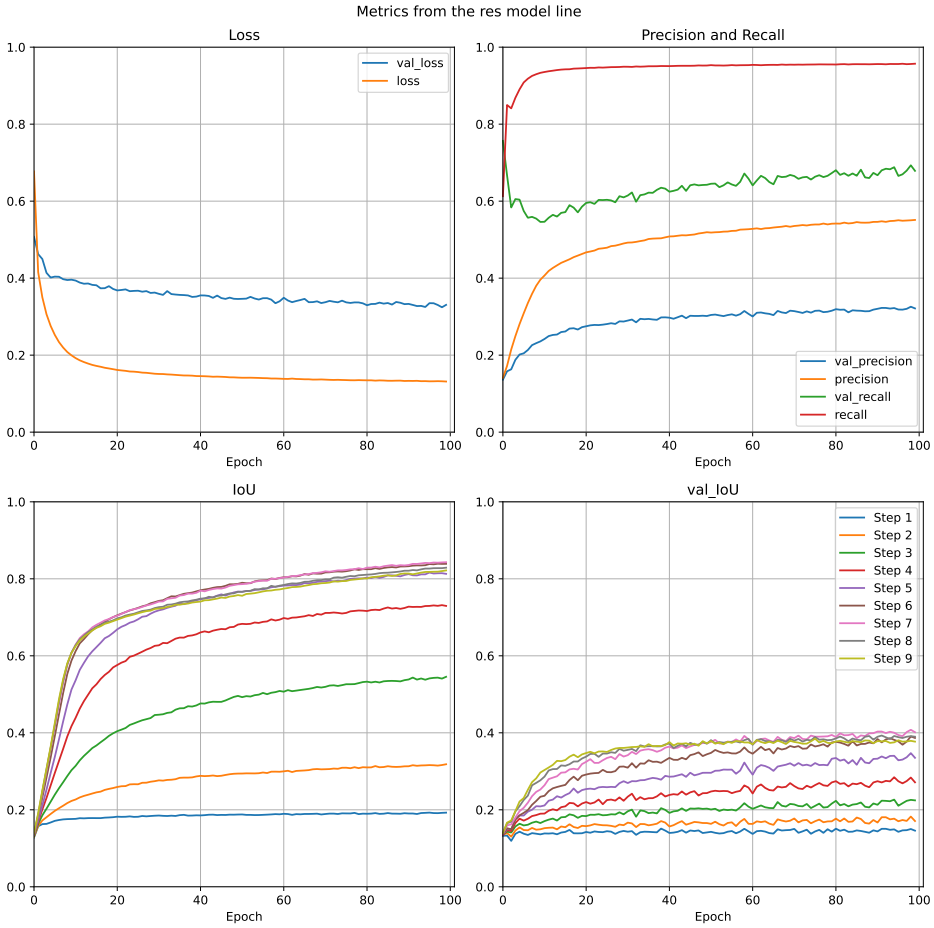

Figure 24: Plots of the reservoir model during training, with loss (top left) calculated with equation (2.24). With blue as out-of-sample validation (val) and orange as training data. Precision and recall are in the top right plot with val precision in blue and training in orange. Val recall is in green and training is in red. They are calculated with equation (2.26) and (2.27) respectively. The IoU score is per time-step (step 0 is excluded as it has no true positive) calculated with equation (2.25), the line starts the weakest at step 1 and is fully there at step 9. Training IoU in the bottom left plot and val IoU in the bottom right plot.

The reservoir in figure 24 reaches beyond 0.8 on the training IoU score. Otherwise similar to the reservoir training on the *box* dataset. The interesting part is the loss and IoU scores on the out-of-sample validation data, although not reaching the same height as the CNN in figure 22 which peaked at 0.5, the reservoirs out-of-sample validation IoU score reached 0.4 with consistency and does not decrease with training. The reservoir's loss is also closer to 0.3 compared to CNN's 0.5, a big difference in performance.

Even with such a big difference between the training data and the out-of-sample data, the reservoir keeps a consistently lower loss than the other models. The reservoir also tells that there is a bigger difference in the *line* dataset, than in the *box* dataset by the bigger difference in the performances in loss, precision, recall and IoU.

4.3.4 CNN-RNN

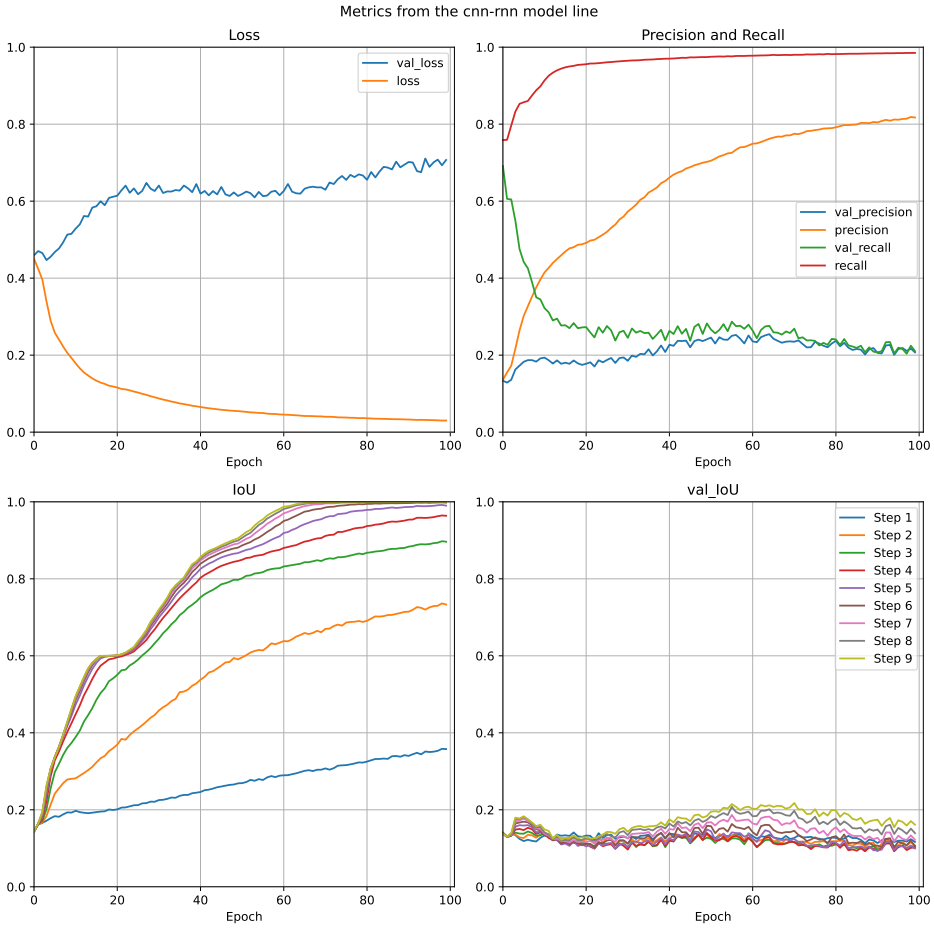

Figure 25: Plots of the CNN-RNN model during training, with loss (top left) calculated with equation (2.24). With blue as out-of-sample validation (val) and orange as training data. Precision and recall are in the top right plot with val precision in blue and training in orange. Val recall is in green and training is in red. They are calculated with equation (2.26) and (2.27) respectively. The IoU score is per time-step (step 0 is excluded as it has no true positive) calculated with equation (2.25), the line starts the weakest at step 1 and is fully there at step 9. Training IoU in the bottom left plot and val IoU in the bottom right plot.

The CNN-RNN in figure 25 does well on the training dataset as it achieves a loss below 0.1. Looking at the training it seems to have distinct learning phases. A similar phenomenon is faintly seen on the CNN for the *line* dataset in figure 22 and on and on the *box* dataset the CNN-RNN model's training in figure 13. The first training phase is from epoch 0 to 20, the next is from 20 to 40 and the last is from 40 to 60.

Why the model is training in phases is hard to explain, but it can be that one part of the network gives a large decrease in loss and that optimization is largely disconnected from the other optimization areas. For example, maybe changing the first layer in the CNN network can decrease loss without the need to change the rest of the network. Therefore the network training focuses on that part til that part is saturated, and then the training algorithm finds

a new way to train itself. It should be noted that for each phase the rate of increase in IoU score for step 9 is decreasing. This can be because the training with most of the potential was the previous phase.

4.3.5 Modified Reservoir

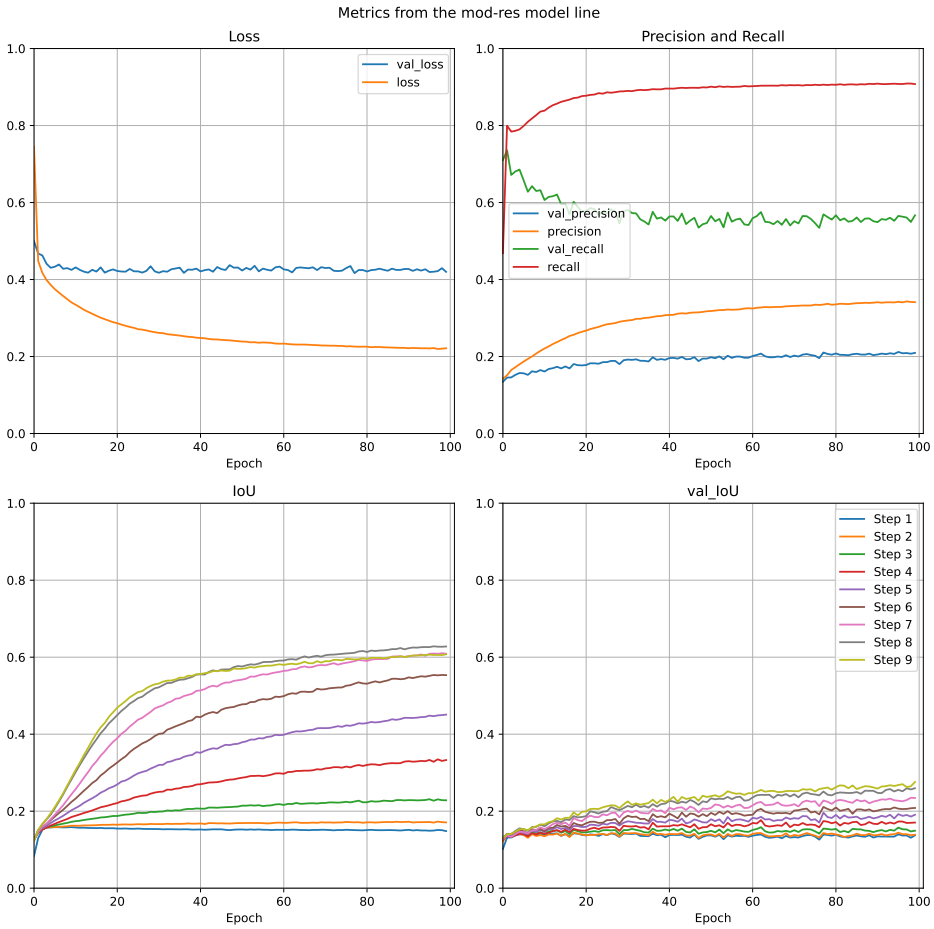

Figure 26: Plots of the modified reservoir model during training, with loss (top left) calculated with equation (2.24). With blue as out-of-sample validation (val) and orange as training data. Precision and recall are in the top right plot with val precision in blue and training in orange. Val recall is in green and training is in red. They are calculated with equation (2.26) and (2.27) respectively. The IoU score is per time-step (step 0 is excluded as it has no true positive) calculated with equation (2.25), the line starts the weakest at step 1 and is fully there at step 9. Training IoU in the bottom left plot and val IoU in the bottom right plot.

The modified reservoir in figure 26 results in a similar but even worse than the modified reservoir with the *box* data. The training results are like the other models better for the *line* data than the *box* data but not by much. The modified reservoir for the *box* datasets as seen in figure 15 ended with a training IoU step 9 score of about 0.55 while this modified reservoir score is just above 0.6. Some of this learning performance is transferred to the out-of-sample validation as the model reaches an IoU score of 0.3 and an out-of-sample validation loss just above 0.4. Beating all of the non-reservoir type models in this metric.

4.3.6 Deep Reservoir

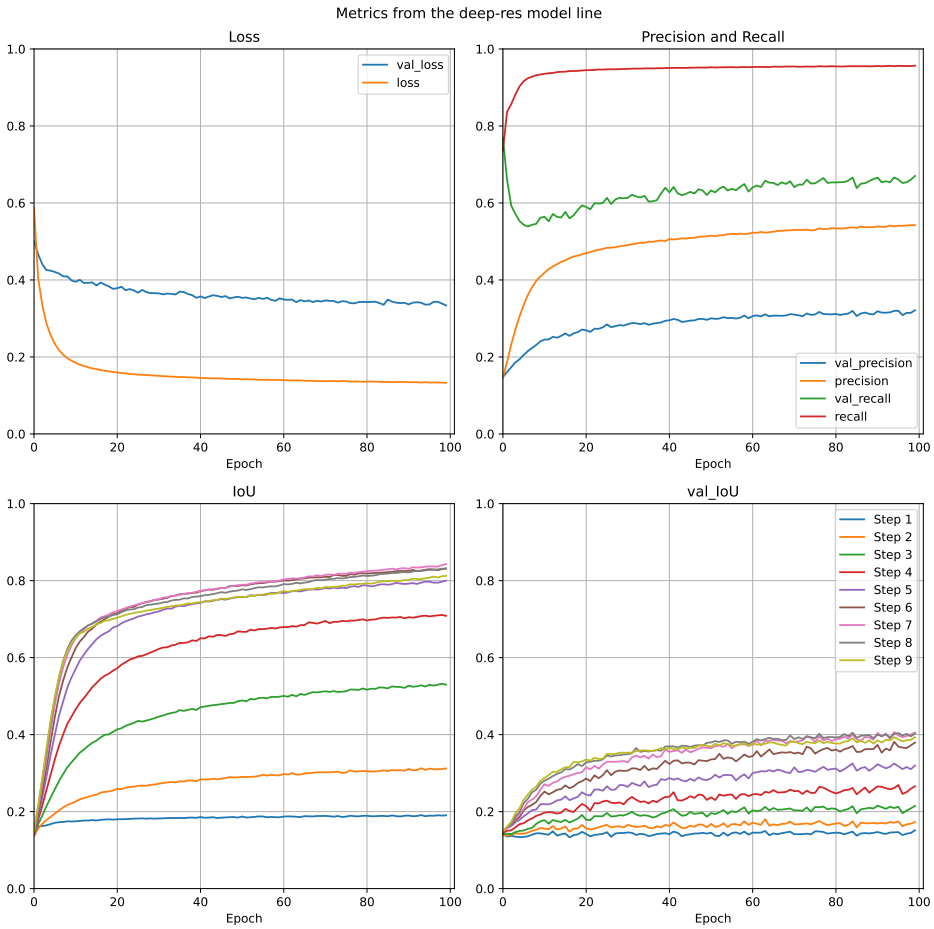

Figure 27: Plots of the reservoir model during training, with loss (top left) calculated with equation (2.24). With blue as out-of-sample validation (val) and orange as training data. Precision and recall are in the top right plot with val precision in blue and training in orange. Val recall is in green and training is in red. They are calculated with equation (2.26) and (2.27) respectively. The IoU score is per time-step (step 0 is excluded as it has no true positive) calculated with equation (2.25), the line starts the weakest at step 1 and is fully there at step 9. Training IoU in the bottom left plot and val IoU in the bottom right plot.

The deep reservoir shown in figure 27 performs again the same as the reservoir, and the extra trainable weights do not seem to help the network in any significant way, nor does the potential for deeper connections. A description of this figure will just repeat figure 24 for the reservoir on the *line* dataset. Deep reservoir 2 achieves again a different metric scores.

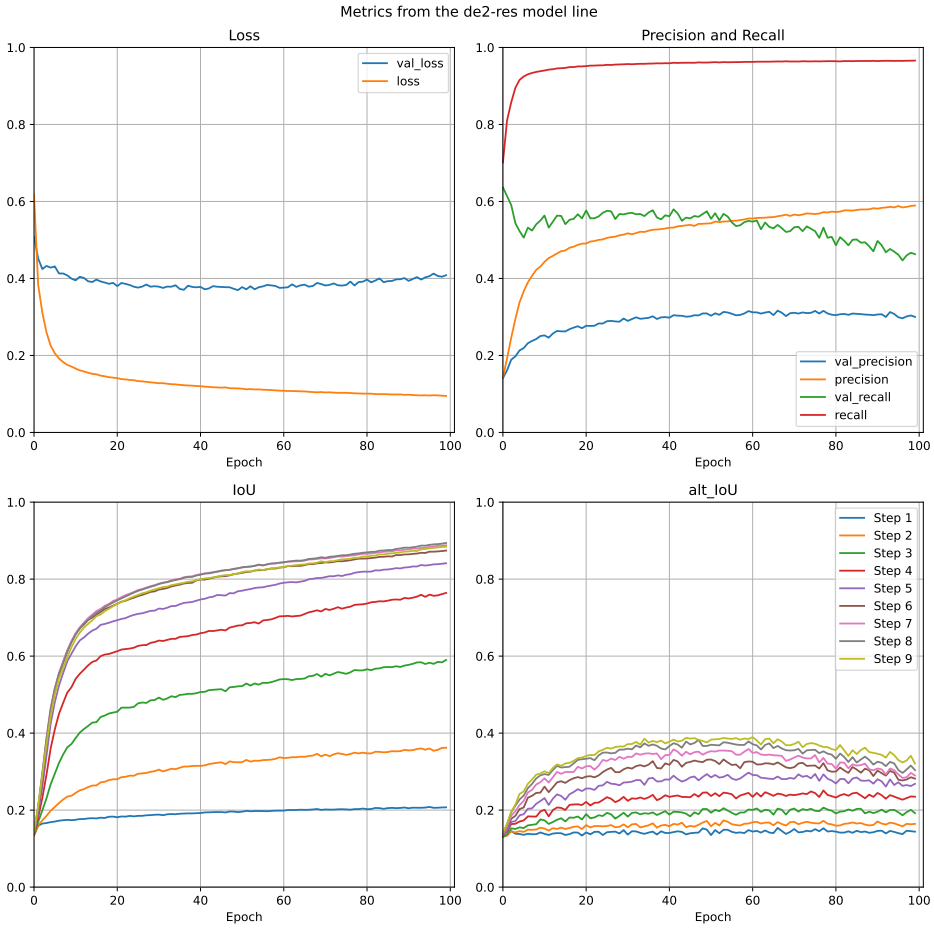

Figure 28: Plots of the reservoir model during training, with loss (top left) calculated with equation (2.24). With blue as out-of-sample validation (val) and orange as training data. Precision and recall are in the top right plot with val precision in blue and training in orange. Val recall is in green and training is in red. They are calculated with equation (2.26) and (2.27) respectively. The IoU score is per time-step (step 0 is excluded as it has no true positive) calculated with equation (2.25), the line starts the weakest at step 1 and is fully there at step 9. Training IoU in the bottom left plot and val IoU in the bottom right plot.

The deep reservoir 2 in figure 28, performs better on the training data than the reservoir with a peak IoU score of 0.05 points higher. This increase seems to come from the higher precision. The interesting result is in the out-of-sample validation, as the deep reservoir has a similar path in the first 60 epochs, reaching the same peak as the deep reservoir, but from epoch 60 to epoch 100 it starts to be over-specialised for the training data.

The loss ended at 0.4 somewhat higher than the normal reservoir. The loss comes from the decrease in the recall, meaning the model loses its ability to find the line with more training. This indicates that the more trainable network also allows the reservoir to over-specialise for only the training data orientation of the line.

4.3.7 CNN-Reservoir

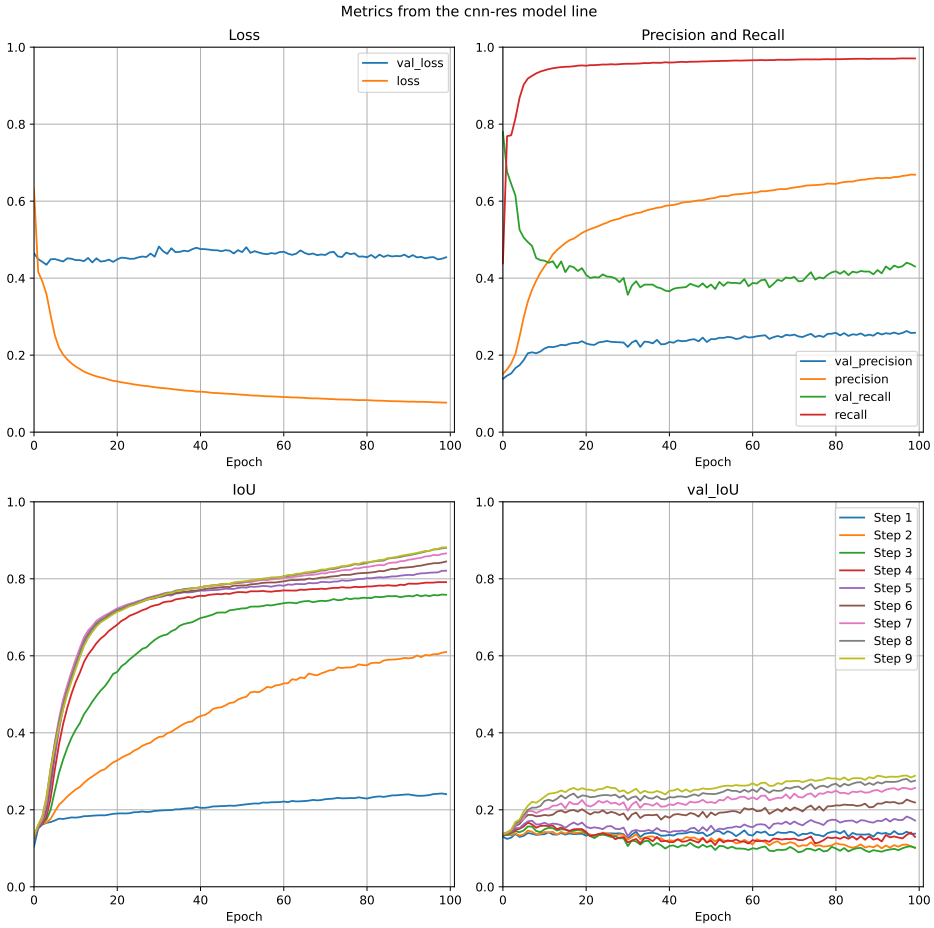

Figure 29: Plots of the CNN-Reservoir model during training, with loss (top left) calculated with equation (2.24). With blue as out-of-sample validation (val) and orange as training data. Precision and recall are in the top right plot with val precision in blue and training in orange. Val recall is in green and training is in red. They are calculated with equation (2.26) and (2.27) respectively. The IoU score is per time-step (step 0 is excluded as it has no true positive) calculated with equation (2.25), the line starts the weakest at step 1 and is fully there at step 9. Training IoU in the bottom left plot and val IoU in the bottom right plot.

The CNN reservoir has an interesting training curve shown in figure 29. This network seems to not be done with its training as the precision and IoU scores are at an increase at epoch 100. The training IoU score for steps 4 and up starts to increase more at epoch 60 after the first knee-point at epoch 15. This is unlike the CNN reservoir for the *box* dataset in figure 18. This can be because the CNN network needs to be of a certain size compared to the training data space to achieve this, or it can be that this would happen on the *box* dataset with more training.

The performance improvement in training data does not translate to the out-of-sample validation data, as the model performs worse in terms of the measured metrics than the reservoir. The improvement step over step is still there but the graph is about 0.1 points lower for all steps.

4.3.8 CNN-LSTM

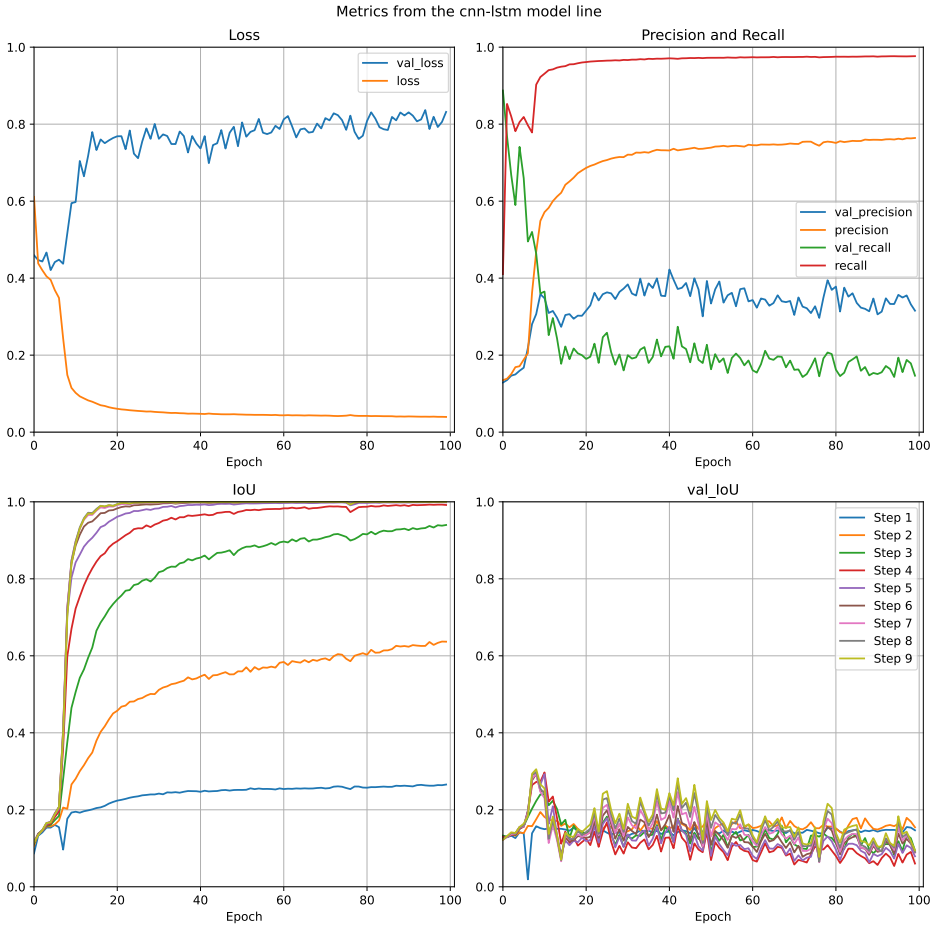

Figure 30: Plots of the CNN-LSTM model during training, with loss (top left) calculated with equation (2.24). With blue as out-of-sample validation (val) and orange as training data. Precision and recall are in the top right plot with val precision in blue and training in orange. Val recall is in green and training is in red. They are calculated with equation (2.26) and (2.27) respectively. The IoU score is per time-step (step 0 is excluded as it has no true positive) calculated with equation (2.25), the line starts the weakest at step 1 and is fully there at step 9. Training IoU in the bottom left plot and val IoU in the bottom right plot.

The CNN-LSTM network in figure 30 is training similar to the CNN-LSTM on the *box* dataset, although it has a five epoch period where it seemingly does not learn much compared to the CNN-LSTM on the *box* dataset, which uses about two epochs before starting the rapid learning spike. Going to the out-of-sample validation data the CNN-LSTM does have a short period where its loss is about similar to the modified reservoir, but the CNN-LSTM loss increases to around 0.8 after epoch five.

4.3.9 All models

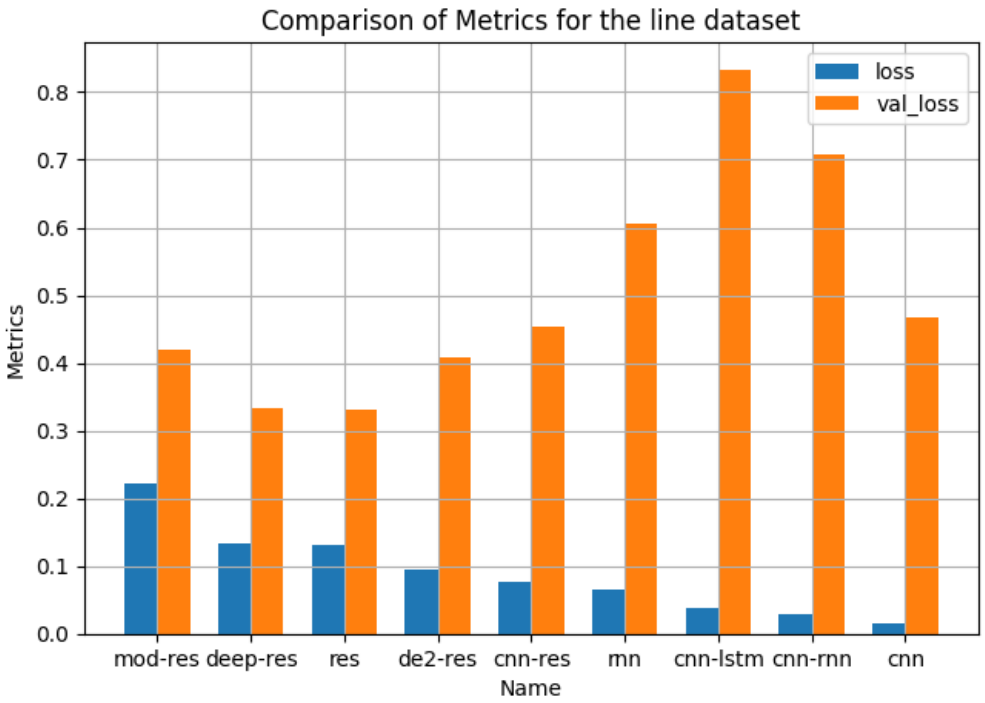

Figure 31: Comparing the loss at epoch 100 of all trained models, where blue is training data and val_loss is the out-of-sample validation. Here the out-of-sample validation is the rotating line

From figure 31 the similarities between most of the models that were seen in figure 21 with the *box* dataset are gone. The models have in general higher out-of-sample validation loss than the *box* dataset and there is not a level of loss that most models held.

The training loss is still in roughly the same order with only the CNN-LSTM and CNN-RNN changing place. The plot shows more of the benefits of the reservoir networks, as the normal reservoir has the best out-of-sample score with the deep reservoir right behind. All the reservoir-type models outperformed the non-reservoir-type models on out-of-sample validation. However, all of the models perform badly, especially the CNN-LSTM which is the worst. The deep reservoir 2, the best out-of-sample loss scorer performs nearly as badly as the modified reservoir with the *line* dataset. It would seem that a more trainable network improves loss performance only when the datasets are close like in the *box* dataset and decreases the loss performance with datasets further apart like the *line* dataset.

5 Discussion

In summary, nine different models have been tested as saliency detectors. This was done by making time series data with a noisy background and an object that fades in. The task of the models is then to predict where the object is. The models got two datasets each containing training and out-of-sample validation data, with the metrics loss, precision, recall and IoU measured under training and testing of the models. The metric results were then plotted. The research question is whether artificial neural networks, especially reservoir-type networks, can be saliency detectors like Li Zhaoping hypostasized the visual cortex is [2]. Without metrics from the visual cortex, the metrics measured could only be used to compare the different artificial neural networks with each other. The models got training data and test data that was outside of the training sample data to see how the models acted with input data they were not trained on. The models also got two different sets of training and out-of-sample data to tell if the results are dependent on the combination of training and out-of-sample data.

The basic CNN model is expected to perform well as the model is made for pictures and has spatial relations between the neurons as a standard. One of the main applications of CNNs is to segment images and to minimise the number of neurons needed to get a good segmentation [13]. The drawback of a basic CNN model is that it does not have any perspective on time. Therefore giving the model time series data where the only constant is the line makes it less likely that the model will excel. It is expected to perform worse than the models that have a combination of different neuron types, like CNN-RNN or CNN-LSTM.

This was not the case, the CNN has the lowest training loss on both datasets with an unprecedented step 1 IoU training score on both datasets. The mismatch in expectations was an overemphasis on the models needing consecutive timesteps to predict where the line is. This was not the case, the CNN network is good at image segmentation and this task seems to fit well inside that.

The RNN model lacks the kernel that organizes the spatial input information like the CNN has. The RNN's trainable network needs therefore to figure out the spatial relations but unlike the CNN, the RNN can remember the past to predict the spots that are getting consistently darker. It can therefore be expected that an RNN achieves a higher score than the reservoir but below the networks that are a combination of CNN and RNN.

The RNN performed as expected, it outperformed all reservoir-type models on training loss but was worse than all non-reservoir models.

The reservoir has connections through time and can therefore find connections with the change in the input, like the RNN network. But unlike the RNN, the reservoir can not update and specialise its network for the training data and is therefore expected to score lower than the other models with a fully trainable network. Especially if the specific configuration of the input dynamics is not captured in the reservoir. The limitation of the reservoir's trainability is supposed to make the reservoirs more resilient to changes in out-of-sample data because the reservoir has to learn more general correlations between input and the desired output in the training data.

The reservoir did perform worse than the other model on the training data as expected, suggesting that the full input dynamics were not represented in reservoir dynamics. And as expected the reservoir did outperform the other models with a lower out-of-sample valida-

tion loss on both datasets, suggesting it did learn something more general from the input dynamics.

Making the combination network CNN-RNN, it is expected that the drawback of CNN's inability to remember between timesteps and the RNN's lack of spatial information will be fixed. Therefore this model should outperform both the architectures it is based on. With the CNN encoding patterns and the RNN remembering the previous pattern and making calculations on the more abstract patterns instead of the pixel values it should be easier to find the part that is consistently getting darker. So the CNN part should allow the RNN part to discern the noisy background from the object quickly, making the CNN-RNN one of the best models for training data.

The CNN-RNN did outperform the RNN as expected but with the surprising performance of the CNN, the CNN-RNN did not manage to outperform the CNN. The CNN-RNN was mostly in the middle of their performance. Again this shows the lack of need for connections in time. The CNN-RNN is, however, outperforming both the CNN and the RNN on the *box* out-of-sample validation data but is worse than both on the *line* out-of-sample validation data.

Adding the structure from the example in figure 4 to the reservoir to make the modified reservoir. The modified reservoir is expected to improve the loss only if this structure benefits saliency detectors. However, from looking at the weights with a general lens this structure would be expected to perform worse since they are less stable, less connected and have a smaller output. Therefore it seems reasonable to expect a higher loss than the normal reservoir. The network as a whole also has fewer free weights to fit so it is only if this structure is really beneficial that any improvement will be shown.

The modified reservoir did not perform better than any of the models tested. So the more general view seems more correct than the structure benefiting saliency detectors. Of course, with the recurrent weight matrix less stable than the normal reservoir, and with the whole network having a smaller output it could be performing worse regardless of whether the structure is beneficial. But with the network being as it is the only positive that can be taken is that the network produces the most similar metrics on the training and out-of-sample validation data for both datasets.

Making two layers instead of splitting the weight matrix can give the deep reservoirs more of an opportunity to find more complex functions and therefore give a better prediction. The first deep reservoir is the two-layer deep reservoir where the output can read the whole reservoir so it also has more trainable weights and is therefore expected to have a better performance than the normal reservoir. However, it might be that smaller reservoir layers can not encapsulate the input dynamics and having two of them just introduces more noise in the reservoir state. If it is the case that reservoir states become too noisy having a trainable layer in the middle of the two layers should fix this issue. The deep reservoir 2 is expected to outperform the normal reservoir as it should be able to more efficiently utilize the two layers. To make the deep reservoir 2 more similar to the two layers of V1, layer 4 and layer 2/3, the output layer of the deep reservoir can only read the last reservoir layer. If there are useful dynamics in the first layer the trainable layer in between should get the second layer to have useful information for the output layer.

What the results show from the deep reservoir are the same as the reservoir. There is no benefit or drawback from this kind of reservoir configuration. The deeper reservoir did not

find any useful dynamics nor did the larger amount of trainable weights help the model. The deep reservoir 2 performance was closer to its expected performance as it outperformed the reservoir model. The deeper network did lower loss, suggesting that the task benefits from a deeper reservoir but the random weights from the deep reservoir were not able to capture this potential.

Given the similarities between the RNN and the reservoir, testing a CNN-Reservoir combination in a network seems relevant. Adding a CNN layer to the reservoir gives the reservoir more potential to process spatial data, it can therefore perform better. Building spatial information into the reservoir might change the reservoir neuron's dynamics as they would be calculating different inputs. This can lead to an earlier prediction. In this new network, the reservoir neuron does not need to encode the spatial data in the same way as the normal reservoir network thereby allowing the reservoir to find correlations that can lead to a lower loss. Adding the CNN layers also gives the network more trainable weights to optimize its behaviour for the specific task at hand giving it a chance for a higher IoU score in the training data, which can also decrease the out-of-sample validation scores. Therefore this network can become more over-specialized than the other reservoir-type models.

The CNN-Reservoir did improve the training performance and gave the lowest loss of all reservoir-type networks so the benefits from the CNN performance also transfer over to the CNN-Reservoir. The CNN-Reservoir did not gain a lot of performance from the CNN with the loss being more close to the other reservoirs than the other models and the CNN part also gave the CNN-Reservoir some higher loss on out-of-sample data

The CNN-LSTM model addresses the limitations of individual models by combining them. It solves the issue faced by CNN by using LSTM neurons in the convolutional layer that can remember past data. This gives the model an understanding of time and space, enabling it to identify important features accurately. Given the input as a time series of images, the CNN-LSTM seems like the model made for this task. The CNN-LSTM is similar to CNN-RNN but with the spatial aspects more integrated throughout the network, it should outcompete the CNN-RNN. The CNN-LSTM is expected to perform very well if not the best of all models tested.

The same problem arises for the CNN-LSTM as the other models. The CNN model performed better than expected and beat out all other models. The time connections and the model's ability to remember do not help the model enough. The increase in parameters from the convolutional LSTM lowers the neuron number of the network, and this could have cost the network an earlier prediction. Having the fully integrated time and spatial features in the network is then not as big of an advantage as expected. The CNN-LSTM trades second place with the CNN-RNN depending on the dataset so it is a well-performing network but not the best.

The *box* dataset, where the training data is a 2×6 rotating line and a 3×4 rotating box is out-of-sample validation data. Training the models on the *box* dataset shows that the reservoir is an inferior model in loss on the training data compared to the non-reservoir type networks. It also shows that with unknown data a reservoir-type network can perform better than non-reservoir models, at least all models tested here. The fact of the reservoir's higher loss score is expected but the results are promising for the reservoir as its predictions are mostly correct as can be seen in the reservoir plot of the time-series in figure 12. Changing the reservoir with the right modifications to make the deep reservoir 2 can improve loss

and IoU score performance on both training and off-sample validation. The results did not however show an improvement with the network mostly inspired by the visual cortex.

The training and the out-of-sample validation dataset in the *box* data combination were not too different so most models performed well. They have seen the line go both ways, so only the shape was changed for the out-of-sample validation. This penalizes the CNN model the most as the CNN's kernels are so big that one kernel can encapsulate the whole line, therefore, it might be that a kernel shape that can only find edges results in worse training loss but better out-of-sample validation loss. So changing the line to a box confuses only the CNN, as the other models are not looking for the specific line but seem to look for something like edges instead. Looking at just the loss on out-of-sample validation gives the deep reservoir 2 the best score, but looking at IoU scores instead gives the CNN-LSTM model the best score.

Going to the *line* dataset, with a non-rotating line of size 6×2 as training and a non-rotating line of size 2×6 as the out-of-sample validation data. All the models did better on training data than on the *box* training data as the input space is smaller in this dataset. Most of the models also failed to have any precision on the out-of-sample validation data, with only the reservoir-type models getting a stable IoU score over 0.3 though a score of 0.3 is not that good either. The best peak score came from the CNN model with a peak IoU score on out-of-sample validation data at about 0.5 but the score goes quickly down again. All models gained more loss going from the training data to the out-of-sample validation data on the *line* dataset than on the *box* dataset suggesting that the data spaces are further apart than in the *box* dataset.

The difference seems further apart in the *line* dataset, where the training and out-of-sample validation are perpendicular to each other. This generates higher losses for all models and this time the recurrent features of some networks did not stabilize the out-of-sample scores like they seem to have done in the *box* datasets. The reservoirs showed their value as they had the lowest out-of-sample loss of all the models.

The CNN and RNN networks are solving this problem very differently and therefore produce very different results. The small number of epochs for the CNN model to get its loss below 0.1, which is lower than all reservoir-type models, indicates that the problem is highly spatial and that a simple CNN network is enough to produce a sailiency detector. The large number of epochs needed for the RNN to get a similar result suggests the task is not as good of a fit for the RNN. It should also noted that the learning rate is not changed on any of the models. This is done so the number of epochs can be compared. But the CNN network is run every time step while the whole RNN network is run every time series, this is likely not the difference though as the CNN-LSTM has a similar learning spike.

Freezing big parts of the network to make a reservoir still resulted in good results and minimal increase in loss compared to other models even with big differences to the evaluation data. A fully trainable network can learn correlations in the training data that do not necessarily reflect the nature of all data given in this problem. Therefore this overspecialization will lead to an increase in loss when measured on the data in out-of-sample validation. This loss is also present in the reservoir as it is also trainable but the reservoir is forced to more general patterns in the training data. The reservoir benefits from this by decreasing the loss when tested on out-of-sample data.

The reservoir achieved scores of 0.8 and higher on the peak training IoU, what a score of

0.8 IoU roughly translates to can be seen in figure 12. How a visual cortex saliency map looks if it has run on this dataset is unknown, but the strongest output signal from the reservoir is at the line's location. This can be compared to the IoU score of one with for example the CNN-LSTM in figure 20. As can be seen in the comparison: the reservoir's calculations do not tend to produce sharp edges, this is true for even the final row. By comparison, consider the progression of the CNN-LSTM, where the output appears almost binary. This indicates that the input information to the reservoir is being smoothed out rather than highlighting contrasts. Alternatively, it seems the reservoir is introducing too much noise into the input signal to make a precise prediction. It can be that forcing the reservoir to have a binary prediction by adding a cutoff to the model would resolve the issue. But this is the prediction data classification metrics receive so the reservoir might have a similar response to the results gained from those. However, adding a cutoff will affect the loss and thereby affect training so the reservoir has the potential to perform better with the cutoff as a part of the network.

Imposing the more layered structure in the modified reservoir did not improve the learning rate or peek score of the model. This can be attributed to a crude method to generate the custom reservoir weights. This method resulted in an even stronger randomness as the absolute value of the eigenvalues was not equal to one. There might be a significant improvement in enhancing the stability of the recurrent weight matrix. Changing the distribution of the off-diagonal weights from a constant to having a distribution of weights could also improve the dynamics of the modified reservoir. However, it can also be a sign that the visual cortex structure is imposed more by biology and physics, not by the mathematical efficiency of having this kind of network. So finding this structure in biological neural networks is more about physical possibility and not the most efficient neuron configuration.

Increasing the number of free parameters or trainable weights in the reservoir to make the deep reservoir 2 led to the best out-of-sample validation data of the models on the *box* data. With the *line* dataset, the less trainable reservoir models show the benefit of a larger resistance to over-specialisation, showing that the behaviour of the networks can be tuned to improve its score or resistance to change in validation data. So the optimal amount of trainable weights in the network is dependent on the combination of training and the test data which in many cases are unknown and therefore hard to optimize for.

The time aspects of the model's task seem to have been overplayed, as there is not any strong correlation between the model that knows what time step the time series is in, as opposed to the CNN model that does not know. Adding time connections to CNN with networks such as CNN-RNN and CNN-LSTM did not improve the network performance. But at least for the *box* dataset adding the recurrent connections did stabilize its prediction results for the out-of-sample validation data. It might also be that the results from the models would change significantly if their datasets were changed. This means the results of the two datasets tested might not be the results received if other datasets were tested on the models.

The sensitivity of the results to the tested dataset becomes apparent when comparing the performance of the CNN and CNN-LSTM models on the two datasets. In the *box* dataset, the CNN exhibits the highest loss during out-of-sample validation, whereas the CNN-LSTM still demonstrates proficient prediction capabilities. The opposite is true in the *line* dataset. Here, the CNN-LSTM incurs a high loss and consistently fails to make accurate predictions, while the CNN performs comparatively better, resembling reservoir-like networks such as

the modified reservoir in the loss. The CNN also has the highest peak IoU score among all models. Therefore, the introduction of a new dataset could potentially alter some of the results. However, it's worth noting that across the two tested datasets, reservoir-type networks consistently outperform other models in terms of out-of-sample validation loss scores, indicating a more robust trend.

When developing a saliency detector, it might be more beneficial to have a good prediction on data similar to training data than having a more robust network capable of predicting data far outside of training data. Then using reservoir networks as saliency detectors might not be that beneficial. The reservoir models also seem to learn as fast as the RNN model in terms of epochs, so the benefit of quick learning is not shown either. Looking at the training time in Appendix C the training times on the hardware are almost the same on the reservoir type models as the non-reservoir model, except for the CNN-LSTM and CNN models. This is of course dependent on the nature of the input data that needs to be predicted. If the saliency detector needs to predict something crucial where a wrong prediction would be a disaster then a reservoir-based network as a saliency detector would be beneficial.

From the experiments on the visual cortex of cats in development from paper [9], it could be interpreted that the network would be unable to find the lines and therefore have an IoU score of around 0.12 or less, 0.12 since it is the relative size of the object to the total input so a random guess is expected to be somewhere close to that. An IoU score of 0.12 or lower can be seen in the non-reservoir models but the reservoirs are better than that, with the best model getting an IoU score of 0.4. Of course, the cats in the paper did not produce an IoU test so it does not make it clear how they compare. The same can be said about the training results, it can not be stated that a 0.8 IoU score is too bad to be a salience detector either. It can only be said that deep reservoir 2 is performing better on the training and worse on the off-sample validation data for the *line* dataset. This is more in line with the expectation from the research done on the early development of the visual cortex in cats in paper [11].

If one looks at the predictions from the input pictures and takes that as what is close to what the human brain can predict then the IoU score from step 1 should be should be very low as there is not enough contrast to see the step one performance. This is however not seen in the CNN model as it scores 0.6 on training data, with a linear increase. Then it could be argued that the CNN network performs too well to be a representation of its biological counterpart. With models like the CNN-RNN or CNN-LSTM having a more realistic prediction. They also result in perfect prediction of the IoU scores with a fine transition to the out-of-sample validation on the *box* dataset. This is also true for the RNN model. And if the saliency detector should mimic the behaviour of the cats from [11] then it is good none of the tree models did very well on the out-of-sample validation on the *line* dataset.

From the results found, there does not seem to be a definite reason why reservoirs can not be saliency detectors. Many of the models perform well as a saliency detector and give promising predictions. Meaning there are potentially many ways to solve this task. However, it might be that the result of the modified reservoir indicates that the neuron configuration is limited by physical or biological limitations.

The reservoirs have a clear advantage when it comes to out-of-sample data. Their resilience to very different environments compared to training data beats out all the models tested. Having artificial neural networks that have good predictions the first time seeing new data can be very useful. However, it does not seem to be one reservoir that fits all types of datasets but the best model has to be tuned to the data.

The results also imply that if the training data encapsulated most of the test data there is no need to use a reservoir-type network. The non-reservoir models outperformed all reservoir-type models on the training data so using them for prediction on data similar to training data would result in the best predictions. Therefore the best model is highly dependent on the environment. It might be that a precise network that knows only what it has been trained on is optimal for the visual cortex, as in the real world there are not that many shapes at this low of a level that are fundamentally new. This does not exclude the possibility that later in the visual system, there are networks that have features more like those of a reservoir.

5.1 Limitations

The exploration of hyperparameters in each model was not exhaustive, and overfitting mitigation techniques like dropout or regularization were not implemented, potentially impacting the thesis's results. Optimal hyperparameters could have sped up training, and changing the parameters like learning rate would have given a different progression for models, potentially increasing their peak performance. With robust overfitting mitigation, it could be that the overspecialization resistance found in the reservoirs is not that much better than what would be found in the improved non-reservoir type models. Additionally, there are additional ways for making training and out-of-sample data and combining them, implying that the testing conducted in this thesis may not fully capture the diverse range of the model's resistance to over-specialization. Time and calculation limitations meant only a selected part of the possible training and out-of-sample data could be used. There is also no biological counterpart to compare the data to, so concluding that a model is too bad or too good for a saliency detector is difficult.

5.2 Future Research

Further research is warranted to explore the potential of reservoir networks. Getting an IoU score above 0.8 with such a limited exploration of the reservoir and showing an advantage over non-res in both out-of-sample validation datasets, suggests that with more research both the training and the out-of-sample validation will perform better. There could also be benefits to adding a cutoff for the prediction on the reservoirs to make the prediction binary. There is also a more general need for more resilience in artificial neural networks doing tasks like driving cars, where there are huge possibilities for novel input data. Similar problems might be helped with a network where under normal conditions a traditional network is trained and in an unsure situation the reservoir can help with making the best guess. This could apply to every artificial neural network acting as an agent in the real world. A way to decrease the loss in both training data and off-sample data might be to make the network somewhat trainable. There might also be other initializers or rules for the choice of weights that perform better. The reservoirs did not generate sharp edges in their outputs so there is some loss in the high noise of the reservoir and finding a way to alleviate that might be beneficial.

6 Conclusion

This thesis has explored the potential of reservoir computing and other artificial neural network architectures as saliency detectors for the visual cortex, inspired by the work of Li Zhaoping [2]. To test the potential of the artificial neural network architectures different models were made and they were tested on two datasets. Each dataset contained training and out-of-sample validation data consisting of time series. Several key findings have emerged through a comparative analysis of reservoir-type models and other conventional artificial neural networks.

All of the artificial neural network architectures tested produced a saliency map. Even the worst model, the modified reservoir, got a training IoU score of just above 0.6, meaning all tested networks can somewhat function as a saliency detector. Using CNNs for this kind of task does make the most sense if all data is similar to the training data. However, if the nature of the data is not, then other alternatives might be highly relevant. The results indicate that reservoir-based models exhibit higher training loss compared to traditional networks, but they outperformed the non-reservoir models on new data. Modifying the reservoir to make a trainable layer between two reservoir layers decreased the loss on training data and on new data. The ability of reservoirs to achieve promising predictions without previous data can be invaluable in many artificial neural networks and shows their potential utility as saliency detectors.

In interpreting the findings, it is crucial to recognize that while demonstrating the effectiveness of a reservoir as a saliency detector, it does not inherently imply a direct resemblance to the architecture of the primary visual cortex (V1) as the architecture and functioning of V1 differ significantly. Furthermore, the results of out-of-sample validation on the *line* dataset indicate performance levels potentially surpassing expectations based on V1's characteristics from experiments on the visual cortex in development [11], hinting at the versatility and potential of reservoir networks are beyond their biological counterparts.

It should also be noted that limitations in hyperparameter search on all the models and the necessity for further exploration of reservoir weight choices must be acknowledged. There are also methods to combat overfitting like dropout or regularization have not been explored and can therefore affect the difference observed between reservoir-type models and the more conventional models.

The implications of this research extend beyond theoretical understanding of different network choices, with potential applications in computer vision and artificial intelligence. The resilience of reservoir networks to diverse environments suggests it would be beneficial for more research. Future research should focus on refining reservoir networks, exploring ensemble methods, and investigating alternative weight initialization strategies. Additionally, further examination of reservoir performance in real-world scenarios and the integration of reservoirs with traditional networks could provide insights into improving network robustness and precision.

The potential utility of reservoir networks in computer science is underscored by their inherent ability to score on out-of-sample data, as evidenced by their performance in the *box* dataset's out-of-sample validation. The adaptability and generalizability of reservoir learning mechanisms highlight their significance in addressing a wide array of computational tasks. However, while the results suggest promise, further research is warranted to fully ascertain the suitability of reservoir networks for the specific task at hand. Refinement and optimization of the current implementation are essential steps towards elucidating the role of reservoir networks in computational problem-solving.

References

[1] G. T. Einevoll, A. Destexhe, M. Diesmann, S. Grün, V. Jirsa, M. de Kamps, M. Migliore, T. V. Ness, H. E. Plesser, and F. Schürmann. The Scientific Case for Brain Simulations. *Neuron* **102**(4), 735–744, 2019. DOI: 10.1016/j.neuron.2019.03.027.

[2] Z. Li. A saliency map in primary visual cortex. *TRENDSin Cognitive Sciences* **6**(1), 2002.

[3] K. Han, A. Xiao, E. Wu, J. Guo, C. XU, and Y. Wang. Transformer in Transformer. *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan. Vol. 34. Curran Associates, Inc., 15908–15919, 2021. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/854d9fca60b4bd07f9bb215d59ef5561-Paper.pdf.

[4] H. Jaeger and H. Haas. Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication. *Science* **304**(5667), 78–80, 2004. DOI: 10.1126/science.1091277.

[5] J. Kim, O. Sangjun, Y. Kim, and M. Lee. Convolutional Neural Network with Biologically Inspired Retinal Structure. *Procedia Computer Science* **88**, 145–154, 2016. DOI: 10.1016/j.procs.2016.07.418.

[6] N. Wagatsuma, A. Hidaka, and H. Tamura. Correspondence between monkey visual cortices and layers of a saliency map model based on a deep convolutional neural network for representations of natural images. *eNeuro* **8**(1), 1–19, 2021. DOI: 10.1523/ENEURO.0200-20.2020.

[7] G. W. Lindsay. Convolutional neural networks as a model of the visual system: Past, present, and future. *Journal of Cognitive Neuroscience* **33**(10), 2017–2031, 2021. DOI: 10.1162/jocn_a_01544.

[8] D. J. Gauthier, E. Bollt, A. Griffith, and W. A. Barbosa. Next generation reservoir computing. *Nature Communications* **12**(1), 2021. DOI: 10.1038/s41467-021-25801-2.

[9] C. Blakemore and G. F. Cooper. Development of the Brain depends on the Visual Environment. *Nature* **228**, 477–478, 1970. DOI: 10.1038/228477a0.

[10] D. L. Yamins, H. Hong, C. F. Cadieu, E. A. Solomon, D. Seibert, and J. J. DiCarlo. Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proceedings of the National Academy of Sciences of the United States of America* **111**(23). the human brain can find an object in still high variance, 8619–8624, 2014. DOI: 10.1073/pnas.1403112111.

[11] R. J. Douglas and K. A. Martin. Mapping the matrix: the ways of neocortex. DOI: 10.1016/j.neuron.2007.10.017. 2007.

[12] D. Sterratt, B. Graham, A. Gillies, G. Einevoll, and D. Willshaw. Principles of Computational Modelling in Neuroscience. Cambridge University Press, 2024. ISBN: 9781108672955. DOI: 10.1017/9781108672955.

[13] S. Raschka and V. Mirjalili. Python Machine Learning. 3rd ed. Packt, 2019. ISBN: 978-1-78995-575-0.

[14] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics* **5**(4), 115–133, 1943. DOI: 10.1007/BF02478259.

[15] X. SHI, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. WOO. Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. *Advances in Neural Information Processing Systems*. Ed. by C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett. Vol. 28. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper_files/paper/2015/file/07563a3fe3bbe7e3ba84431ad9d055af-Paper.pdf.

[16] A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. arXiv: 1312.6120 [cs.NE]. 2014.

[17] D. C. Lay, S. R. Lay, and J. J. McDonald. Statistikk for universiteter og høgskoler. 5th. Pearson Education Limited, 2015. ISBN: 9781292092232.

[18] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Y. W. Teh and M. Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 249–256, 2010. URL: https://proceedings.mlr.press/v9/glorot10a.html.

[19] N. Ketkar. Introduction to Keras. Apress, 97–111, 2017. DOI: 10.1007/978-1-4842-2766-4_7.

[20] G. G. Løvås. Statistikk for universiteter og høgskoler. 4th. Universitetsforlaget, 2018. ISBN: 9788215031040.

[21] G. Du, X. Cao, J. Liang, X. Chen, and Y. Zhan. Medical Image Segmentation based on U-Net: A Review. *Journal of Imaging Science and Technology* **64**(2), 020508-1-020508–12, 2020. DOI: 10.2352/J.ImagingSci.Technol.2020.64.2.020508.

A Appendix A: Data Parameters

Variable name	Standard value	Explanation of the variable
matrix size	(10, 10)	the size of the input image
steps per time-series	10,	number of steps in each time series
subsets	False	if the placement is a subset of all possible
strength and kernel	(2, 3),	the sigma and size of the kernel
size	(6, 2),	the size of the object
rotate	True,	if the object can be rotated 90°
new background	True,	if the background changes every step
shape	line,	the type of object
val data	True,	if there is out-of-sample validation data
val strength kernel	(2, 3),	out-of-sample validation dataset versions
val size	(3, 4),	
val rotate	True,	
val new background	True,	
val shape	line,	

Table A.1: A table of the variables used in the code to make the training and out-of-sample validation data

B Appendix B: Hardware and Dependencies

The GitHub Code was run on a laptop with an Intel i7-12700H CPU with a base clock of 2.3 GHz with 16 GB of memory running at 4.8 GHz.

With the dependencies:

Package name	Version
keras	2.15.0
matplotlib	3.5.0
numpy	1.26.3
pandas	2.1.4
raster-geometry	0.1.4.2
scipy	1.11.4
tensorflow	2.15.0

Table B.2: The imported Python package names and versions used in the code to produce the results

C Appendix C: Training Times

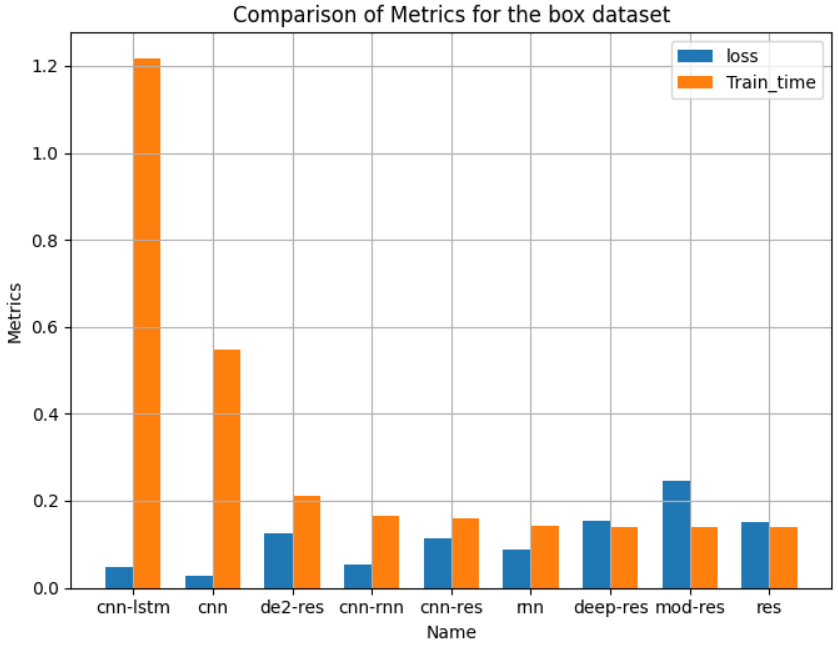

Figure C.1: Plots of the Time in hours and the resulting loss on the different models on the box dataset

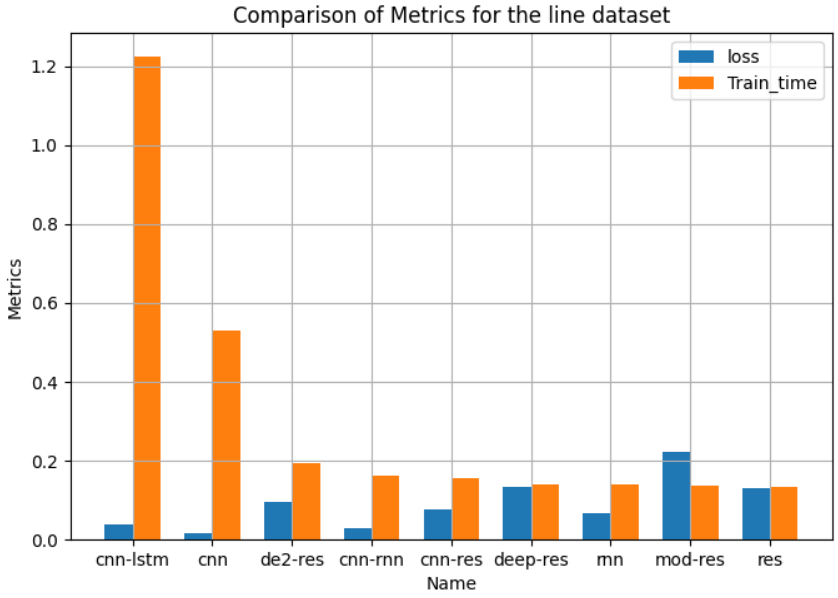

Figure C.2: Plots of the Time in hours and the resulting loss on the different models on the line dataset

Norges miljø- og biovitenskapelige universitet
Noregs miljø- og biovitskaplege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway