

https://dspace.org/

Learning level set method by Echo State Network for mage Segmentation

Mashinini, Thabang L

2022
https://hdl.handle.net/10539/33910

Downloaded from a DSpace repository. https://dspace.org/

University of the Witwatersrand

Johannesburg



WITS
UNIVERSITY

School of Computer Science And Applied Mathematics

A dissertation submitted to the Faculty of Science, University of the Witwatersrand,in fulfillment of the requirements for the degree of Master of Science.

Learning Level Set Method by Echo State Network for Image Segmentation

---

by

Thabang L. Mashinini

Supervisor: Prof. Joel Moitsheki

2022-02-06

Abstract

The back-propagation of Recurrent Neural Networks (RNNs) is computationally costly and unstable, resulting in vanishing and exploding gradients. Echo State Networks (ESNs) are a modern method of training RNNs that is computationally efficient. Unlike RNNs, the ESN has an untrained RNN called a reservoir, and the reservoir to output connection are the only components of the ESN that are trained using a linear approach (i.e. linear regression) without using back-propagation. However, setting up the parameters of the reservoir in order to achieve optimal results is a challenge.

This research aims to investigate the merits of using ESNs as an alternative approach to training RNNs, through a comparative study, applied to iterative segmentation problems. Under this iterative segmentation, we propose a novel approach to the current deep learning Variational Level Sets (VLS). The variational level set is formulated as a learnable spatiotemporal data-driven approach under ESNs. We investigate five approaches for learning the VLS Convolutional ESN (CESN), RNN (CRNN), Gated Recurrent Network (CGRU), Long Short-Term Memory (CLSTM), and a 3D Convolutional Neural Network (3DCNN).

The CGRU and CLSTM are shown to be the best architectures for learning data-driven spatiotemporal VLS on four different databases in our experiments. Significant improvements in the segmentation results are achieved with an increase in data size. The memory of the ESN is related to the leaking rate and spectral radius of the reservoir. These parameters are major contributors to the low performance of the CESN in learning the VLS.

Contributions of this research include a novel formulation to learnable deep learning VLS as a spatiotemporal data-driven method; the investigation of the ESNs hyper-parameters and how they relate to the memory and performance of ESNs.

Declaration of Authorship

I, THABANG LUKHETHO MASHININI, declare that this thesis titled, 'Learning Level Set Method by Echo State Network for Image Segmentation' and the work presented in it are my own. It is being submitted for the Degree of Master of Science at the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination at any other university.

Signed: 

Date: 2022-02-06

Acknowledgements

I'm truly grateful of Prof. Terence Van Zyl and Prof. Turgay Celik for introducing me to reservoir computing, and thanks to them I finally learned the functional of coding anything from scratch! I'd like to thank Prof. Moitsheki for the support and understanding, I'm beyond grateful. For proofreading my work, I'd like to thank Marike Kluyts, Milka Madahana, and John Ekoru. I thank Dr. Etienne Vos for his guidance and teaching for me the "short bursts of focus" principle.

I dedicate this research towards my mother, father, sister and brother. My parents Dumazile and Sello Sekgoto for trying by all means to provide me and my siblings a quality education, love, support, and a healthy loving home, I'm infinity grateful for everything you do for us. My sibling who have been my best friends all my life Lebogang Mashinini-Sekgoto and Palesa Mashinini-Sekgoto thank you for their continued support and tolerance to listening to my daily cries about my research, and most importantly for their encouraging words "Tlogela go stressa thata :-)". Ke ya leboga bo Sekhoto sa Makolobane!

I'd like to thank my friends Mpho Kgoadi and Sisipho Ntengo for their continued support and the "tatakai" spirit of going beyond plus ultra! And also my entire Shonen Jump family for their encouraging words, Naruto Uzumaki -"Believe it!", Roronoa Zoro -"You need to accept the fact that you're not the best and have all the will to strive to be better than anyone you face." and Yami Sukehiro -"Surpass your limit. Right here, right now."

Finally I give my thanks to Modimo le Badimo. Ke nna **Sekgoto**, Mokebe, Makolobane, Matlotlomela. Manelwa ke pula, monna oa batho. **Mashinini**, Limbembhe, Mwelase, Intozimandla. Izinhliziyo eziyishumi kwafa eyodwa. **Nhlapo**, Segede, Nogombile. Abangavali ngomvalo, abavala ngamakhanda amadoda.

In addition, the financial assistance of the South African Bank Seta towards this MSc, is hereby acknowledged and also I'd like to thank the Mathematical Sciences Support unit for the High Performance Computing infrastructure provided to computations for my experiments.

Some aspects and ideas of this research resulted in the following publications:

Mine Worker Threshold Shift Estimation via Optimization Algorithms for Deep Recurrent Neural Networks.

M.C.I. Madahana J.E.D. Ekoru T.L. Mashinini O.T.C. Nyandoro (IFAC 2019)

Noise Level Policy Advising System for Mine Workers

M.C.I. Madahana J.E.D. Ekoru T.L. Mashinini O.T.C. Nyandoro (IFAC 2019)

Long-Range Seasonal Forecasting of 2m-Temperature with Machine Learning

E. Vos, A. Gritzman, S. Makhanya, T. Mashinini, C. Watson (NeurIPS 2020)

Comparison of Deep Neural Network Based Methods for Estimation of Particulate Matter Concentration with application to Mining.

M.C.I. Madahana T.L. Mashinini J.E.D. Ekoru O.T.C. Nyandoro (Under Review)

Portions of this research have been presented at:

Third Deep Learning Indaba, Kenya, 2019

The 9th Cross-Faculty Postgraduate Symposium at the University of the Witwatersrand, 2018.

Contents

Declaration of Authorship i

Acknowledgements ii

Publications, Other work and Presentations iv

List of Figures x

List of Tables xiii

Abbreviations xiv

1 Introduction 1

1.1 Problem statement and motivation 1

1.2 Research aims, questions, objectives and hypothesis 6

1.2.1 Aims 6

1.2.2 Objectives 7

1.2.3 Research questions 7

1.2.4 Research hypothesis 8

1.3 Research significance and limitations 8

1.3.1 Research significance 8

1.3.2 Limitations 9

1.4 Thesis outline 10

2 Background and Literature Review 12

2.1 Recurrent Neural Networks (RNNs) 12

2.1.1 Introduction 12
2.1.2 Formalism 12
2.1.3 Long short-term memory (LSTM) 15
2.1.4 Gated recurrent unit (GRU) 18
2.1.5 Conclusion 20
2.2 Reservoir Computing (RC) 21
2.2.1 Introduction 21
2.2.2 Formalism 21
2.2.3 Echo state networks (ESNs) 23
2.2.4 Liquid state machines (LSMs) 28
2.2.5 Evolution of recurrent systems with linear output (Evolino) 31
2.2.6 Various types of reservoir 31
2.2.7 Conclusion 31
2.3 Learning from the reservoir 33
2.3.1 Introduction 33
2.3.2 Offline learning 33
2.3.2.1 Linear regression 33
2.3.2.2 Moore-Penrose pseudo-inverse 34
2.3.2.3 Tikhonov regularization 35
2.3.3 Online learning 35
2.3.3.1 Gradient descent 35
2.3.4 Conclusion 37
2.4 Global parameters of the reservoir 38

2.4.1 Introduction 38
2.4.2 Size of the reservoir 38
2.4.3 Sparsity of the reservoir 39
2.4.4 Spectral radius 39
2.4.5 Leaking rate 40
2.4.6 Input scaling 40
2.4.7 Conclusion 40
2.5 Reservoir dynamics and properties 41
2.5.1 Introduction 41
2.5.2 Echo state property and spectral radius 41
2.5.3 Conclusion 43
2.6 Image Segmentation 44
2.6.1 Introduction 44
2.6.2 Partial differential equations for image segmentation . 45
2.6.3 Variational levelset (VLS) 47
2.6.3.1 Chan-Vese image segmentation 49
2.6.4 Variational level set as a trainable deep learning methods 53
2.6.5 Conclusion 54

3 Method **55**

3.1 Introduction 55
3.2 Formulating the levelset as a ESN 56
3.2.1 Introduction 56
3.2.2 Formulation 56

3.2.3 Data generation 58
3.2.4 Data representation 59
3.2.5 Model architecture 62
3.3 Data acquisition 64
3.4 Model evaluation 65
3.4.1 Confusion matrices 65
3.4.2 Precision 66
3.4.3 Recall 66
3.4.4 F1 Score 67
3.4.5 Intersection over union (IoU) 67
3.4.6 Conclusion 68
3.5 Training and inference 68
3.6 Learning and optimization 69
3.6.1 Loss function 69
3.6.2 Optimization 70
3.6.3 Data augmentation and preprocessing 70

4 Results and Discussion **72**

4.1 Introduction 72
4.2 WSD 73
4.2.1 Introduction 73
4.2.2 Results and discussion 73
4.2.3 Conclusion 79
4.3 BSD 81

4.3.1 Introduction 81
4.3.2 Results and discussion 81
4.3.3 Conclusion 86
4.4 CIFAR-10 87
4.4.1 Introduction 87
4.4.2 Result and discussions 87
4.4.3 Conclusion 92
4.5 CIFAR-100 94
4.5.1 Introduction 94
4.5.2 Results and discussion 94
4.5.3 Conclusion 99
4.6 CESN global parameter optimization 101
4.6.1 Introduction 101
4.6.2 Results and discussion 101
4.6.3 Conclusion 103

5 Conclusions, Contribution and Future Work 105

5.1 Overview 105
5.2 Research achievements 105
5.3 Contributions and Future Work 108

Appendices 109

References 110

List of Figures

1 Schematic of a neural network. (left) Recurrent Neural Network (RNN). (right) Feed-forward neural network (FNN) . . 2

2 Schematic of an RNN. The hidden layer's recurrent connections enable information to be passed from one input to the next. The arrow indicate the information flow. 13

3 Schematic of an Long Short-Term Memory Cell. 16

4 Schematic of an Gated Recurrent Memory Cell. 20

5 Schematic of an Echo State Network. 24

6 Liquid State Machine. 28

7 Binary Image Segmentation. The task is to group the pixels into foreground (i.e object of interest Giraffe) and the background. 44

8 Representation of a implicit levelset. The image is defined on the image plane (red), the high dimensional levelset function (gray cylinder) and the zero levelset function (green) $\mathbf{C}$. . . 48

9 The image is defined on $\mathbf{\Lambda}$ (the big rectangle), the (arbitrary) red curve is $\mathbf{C}$. 50

10 Iterative segmentation architecture. 57

11 LS Generated training dataset. An input image $\mathbf{I}$ and initial binary mask $\mathbf{M}_0$, are feed into the VLS, and at each time step t a corresponding binary mask $\mathbf{M}_t$ is generated, where t is the number of iterations to the final optimal segmentation. . . . 58

12 Input image representation. 61

13 3DCNN architecture. 63

14 CRNN architecture. 64

15 WSD validation loss curves for the CESN, CLSTM, CGRU, CRNN and 3DCNN. 74

16 WSD validation performance curves for the CESN, CLSTM, CGRU, CRNN and 3DCNN. 76

17 WSD examples of iterative segmentation. 1st column input, 2nd Chan-Vese, 3rd column CESN, 4th CRNN, 5th column CLSTM , 6th column CGRU , 7th column 3DCNN. 79

18 BSD validation loss curves for the CESN, CLSTM, CGRU, CRNN and 3DCNN. 82

19 BSD validation performance curves for the CESN, CLSTM, CGRU, CRNN and 3DCNN. 83

20 BSD examples of iterative segmentation. 1st column input, 2nd Chan-Vese, 3rd column CESN, 4th CRNN, 5th column CLSTM , 6th column CGRU , 7th column 3DCNN. 85

21 CIFAR-10 validation loss curves for the CESN, CLSTM, CGRU, CRNN and 3DCNN. 88

22 CIFAR-10 validation performance curves for the CESN, CLSTM, CGRU, CRNN and 3DCNN 89

23 CIFAR-10 examples of iterative segmentation. 1st column input, 2nd Chan-Vese, 3rd column CESN, 4th CRNN, 5th column CLSTM , 6th column CGRU , 7th column 3DCNN. 91

24 CIFAR-100 validation learning curves for the CESN, CLSTM, CGRU, RNN and 3DCNN 95

25 CIFAR-100 validation performance curves for the CESN, CLSTM, CGRU, RNN and 3DCNN 96

26 CIFAR-100 examples of iterative segmentation. 1st column input, 2nd Chan-Vese, 3rd column CESN, 4th CRNN, 5th column CLSTM , 6th column CGRU , 7th column 3DCNN. . . . 98

27 CESN hyper-parameter optimization 102

List of Tables

1 Confusion matrix. 65

2 WSD dataset optimal parameters for the CESN, CRNN, CLSTM, CGRU and 3DCNN 73

3 The performance of the CESN, CLSTM, CGRU, CRNN and 3DCNN on the testing set. 77

4 BSD optimal parameters for the CESN, CRNN, CLSTM, CGRU and 3DCNN. 81

5 BSD testing results. Here we compare various models and noise is also included. 84

6 CIFAR-10 dataset optimal parameters for the CESN, CRNN, CLSTM, CGRU and 3DCNN 87

7 CIFAR-10 testing results. Here we compare various models and noise is also included. 90

8 CIFAR-100 dataset optimal parameters for the CESN, CRNN, CLSTM, CGRU and 3DCNN 94

9 CIFAR-100 testing results. Here we compare various models and noise is also included. 97

Abbreviations

Acronym	Meaning
FNN	Feed-Forward Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
GRU	Gate Recurrent Unit
RC	Reservoir Computing
ESN	Echo State Network
ESP	Echo State Property
LSM	Liquid State Machines
CESN	Convolutional Echo State Network
CLSTM	Convolutional Long Short-Term Memory
CGRU	Convolutional Gate Recurrent Unit
CRNN	Convolutional Recurrent Neural Network
CNN	Convolutional Neural Network
3DCNN	3D-Convolutional Neural Network
BPDC	BackPropagation-Decorrelation Learning Rule
BPTT	Back-Propagation Through Time
RTRL	Real-Time Recurrent Learning
FTP	Fast Forward Propagation
GT	Greens Function
BU	Block Update
PDE	Partial Differential Equations
RLS	Recurrent Levlset
AC	Active Contours
VLS	Variational Level Set
WSD	Weizmann Segmentation Dataset
BSD	Berkeley Segmentation Dataset
CIFAR	Canadian Institute For Advanced Research

1 Introduction

1.1 Problem statement and motivation

The modeling of temporal (i.e time dependent) tasks forms a vital part of many fields of research (e.g speech recognition, character recognition, dynamical system). The problem of modeling temporal tasks is referred to as *time series prediction* (i.e to forecast possible events of the time series based on previously observed events). Such estimates may be obtained from a mathematical model of the mechanisms which generate the time series.

In practice, insufficient or no physical knowledge of the processes generating the time series is often available, making the deductive approach to modeling impractical. In this case, the rules that govern these processes must be deduced from historical observations. These laws may be defined as an inferred functional relationship between the previous and future events of the time series.

Over the past decade, a successful implementation of these functional relationships which provide a mapping from previous observation onto estimates of future events, is artificial neural networks (ANNs), particularly feed forward networks (FNNs) as shown in Figure 1 (right). These networks have proven to be reliable tools in many problems. FNNs are stateless (i.e. have no memory) networks, they have been applied to time series problems, however due to the lack of memory they are not best suited for

time series problems. This issue was solved through the introduction of feedback connections (i.e. loops) from the output of the units back to their inputs in the connectivity of the FNNs, referred to as *recurrent-neural network (RNNs)*. The advantage of RNNs compared to FNNs is their ability to model temporal events (i.e. time dependent) due to the internal memory introduced by the feedback connections as shown in Figure 1 (left).

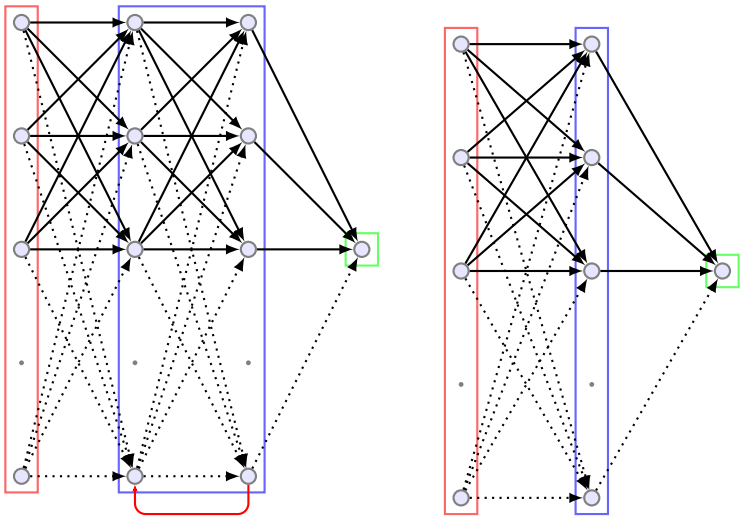

Figure 1: Schematic of a neural network. (left) Recurrent Neural Network (RNN). (right) Feed-forward neural network (FNN)

RNNs were proposed in the 80's for solving time series problems [20, 60, 78]. RNNs have a configuration similar to a multi-layer perception or feed-forward network, with the exception that the latter has cycles (loops) in the hidden units. [43], as shown in Figure 1. RNNs can retain temporal information between events that are separated from each other, because of these cyclic (i.e recurrent) connections, allowing them to preserve information about the past [55]. These cyclic connections are important due to the following:

- Even in the absence of input, RNNs may establish self-sustaining temporal activation along their recurrent connections [43]. These recurrent connections make RNNs dynamic systems [1], while feed-forward networks are non-dynamic system. Non-dynamic systems output is determined by the current input, while dynamical systems output is determined by both the current and previous inputs.

- RNNs can preserve a nonlinear transformation of their input history in their hidden state, giving them a dynamic (i.e. complex) *memory* and allowing them to process time-dependent information [43].

For two purposes, RNNs are a valuable method for working with nonlinear time series applications. RNNs have been found to be universal approximators [2] of dynamical systems, and the brain's neuronal structure exhibits recurrent pathways [21]. Both findings show that RNNs have the ability to be useful tools in engineering applications [43]. Gradient descent is a traditional approach to supervised learning in neural networks that involves recursively updating the input, internal, and output weights in accordance with their predicted gradients in order to decrease the output error. The following are few classic examples of such methods:

- Real-time recurrent learning (RTRL): The gradients with respect to the weights at any time t are computed with respect to weights from previous steps $t-1$ [5]. This strategy is appropriate for applications

[1] A dynamic system is one in which a mechanism explains how a point in a geometrical space changes over time.

[2] Universal approximations theorem states that a feedforward network with a single layer containing a finite number of neurons can approximate continuous functions on a compact set of recurrent neural network is neural network $\mathbb{R}^n$, under the condition that the activations function is assumed to be simple.

which require online learning. This methods of $O(N^4)$ computational complexity, which means computations are performed per data points where N denotes the number of units in the network [81].

- Backpropagation through time (BPTT): The fundamental idea is to convert the RNN to FNN by unfolding time iteration into layers, each layer layer have the same weights [79]. The equivalent feed-forward network is trained using the back-propagation algorithm. For the offline applications, the method requires $O(N^2)$ computations for each data point. Conversely, for online applications, the method is computationally expensive, since the gradient is propagated for each data point through time $t = 1$ [59].

- Fast forward propagation (FFP): The gradients are computed recursively, and is thus an online technique [69, 70]. The FFP method is based on the principle of recursively obtaining the boundary conditions of back-propagated gradients at time $t = 1$. Instead of using BPTT, this allows the gradient to be measured forward in time. This method has a computational complexity of $O(N^3)$.

- Green's function (GF): This approach extends upon FFP, by computing the output gradient recursively [66]. This method is an online approach, and the complexity is $O(N^3)$. The gradients are updated in a online manner.

- Block update (BU): This method updates the weights by combining FFT and BPTT, and it has a computational complexity of $O(N^3)$ per data point [80, 63, 82].

- A new method known as Atiya-Parlos recurrent learning (APRL) with a $O(N^2)$ complexity was created to unify the above-mentioned approaches [5]. The basic idea is to approximate the gradients in terms of the activations of the neurons and change the weights incrementally to converge toward the optimal activation [5]. This method converges faster than previous methods, and as a result, it could be able to close the gap between traditional gradient descent trained RNNs methods and reservoir computing methods.

Other than the approach mentioned above, there are other ways to train RNNs. This involves using algorithms like Expectation-Maximization or Extended Kalman filters [44, 57]. While some of the algorithms discussed above can effectively calculate the gradient using gradient descent, it is difficult to train RNNs using gradient descent-based methods in practice [43] due to the following drawback:

- An infinitesimal change of the parameters of the network during learning results in bifurcation [3] in the networks dynamics. This can often results in unstable gradient, thereby becoming non-convex in the weight space. As a consequence, gradient descent algorithms may not converge [18].
- The recurrent connections limits the size of RNNs since performing a single parameter update is computationally expensive, so training RNNs takes more time [43].

[3] Bifurcations result in a change in the topological structure of a network.

- Owing to the exponential loss of gradient information over time due to BPTT, learning long-term temporal dependencies becomes difficult [43].
- Advanced training algorithms can be computationally expensive and require a large number of parameters to be optimized [43].

RNNs remain promising approaches and their ability to approximate any dynamical system, due to the possessions of cycles, which enables them to retain history information. It is in this context of slow progress in the development of an effective way to train RNNs, that a new approach was discovered referred to as *reservoir computing* (RC) [30, 46].

This study is primarily concerned with a comparative analysis between classical RNNs and RC methods. We explore the advantage of using these methods in solving iterative image segmentation problems.

1.2 Research aims, questions, objectives and hypothesis

1.2.1 Aims

This research aims to comparatively investigate the performance of ESNs to that of state-of-the-art RNNs for learning the variational levelset. And how to formulate classical variational levelset problems under deep learning frameworks as a data-driven approach.

1.2.2 Objectives

The above aim of this research will be achieved through the following objectives:

- To generate a sequence of images using the variational levelset method, this will serve as a spatiotemporal dataset for learning the VLS segmentation problem.
- To learn the evolution of the variational levelset using deep learning methods.
- To compare the ESN's performance in learning the variational levelset to that of state-of-the-art RNNs on different databases.
- To extensively study and investigate how the ESN parameters are set up and what effect they have on performance.

1.2.3 Research questions

Here we present the research questions that will guide us through the research process. The questions we are aiming to answer are:

- If we consider the reservoir (i.e. recurrent layer) to have a similar updating mechanism as the variational levelset γ_t. How do we represent a single image $\mathbf{I}$ as spatiotemporal data X_t in ESNs?
- Is it possible for the ESN to learn the variational levelset? And even outperform the most advanced RNNs?

- Which of our proposed deep learning architectures is the most effective at learning the variational levelset? Is this deep learning architecture better than the current existing state-of-the-art deep learning variational levetset methods?
- Is it possible to empirically determine which set of optimal parameters satisfy the echo state property described in Section 2.5.2?

1.2.4 Research hypothesis

The hypothesis of this research is:

- The ESN can learn the evolution of the variational levelset from the generated data.
- The CESN is computationally less expensive to train than RNNs. However, finding a set of optimal hyper-parameters such that the echo state property is satisfied is nontrivial, resulting in sub-optimal performance.

1.3 Research significance and limitations

1.3.1 Research significance

Image segmentation is an essential preprocessing step for many computer vision problems such as object detection/classification. These problems find many applications in self-driving cars, medical imagining, and more.

However, in practice, particularly for supervised machine learning problems, we often have limited annotation (i.e. labeled) segmentation databases. As a result, iterative segmentation methods can be appropriate in such cases.

Several studies have proposed reformulating classical variational levelsets methods as learnable deep learning methods referred to as deep learning variational levelsets. Deep learning variational levelsets propose a combined hybrid architecture between deep learning and variational levelsets methods in an end-to-end trainable deep learning variational levelset. The variational levelset component is either used as a preprocessing or postprocessing step. However, there are few, if any, studies that propose learning the variational levelset independently as a fully data-driven approach. The computational cost of training a fully data-driven deep learning variational levelset reduces as a result.

1.3.2 Limitations

Our investigation of variational levetset reformulated as a deep learning framework is limited to the following problems: Can the future state of a variational levelset be predicted using deep learning given the historical states of the variational levelset?

1.4 Thesis outline

The following is how the content in this study is organized:

- In Chapter 1, I begin by stating the research problem, aims, objectives, hypothesis, research significance, limitation and the questions I plan to answer in this research.
- In Chapter 2 I provide detailed literature review:
  - In Section 2.1, I establish the theoretical basis of different RNNs. I also discuss the benefits and drawbacks of each method for modeling sequential problems.
  - In Section 2.2, I establish the theoretical basis for a full understanding of Reservoir Computing (RC). The fundamental aspects of pertaining to various RC techniques are developed and discussed in detail in this chapter, which are discussed in the following sections:
  - In Section 2.5, I provide a comprehensive discussion on the dynamic properties of the reservoir, and the best practice for selecting their parameters based on literature.
  - Section 2.3 consists of a detailed theoretical framework for learning the reservoir. I discuss the various forms of offline and online learning approaches, along with their advantages and disadvantages. In the online approach, I demonstrate how to derive the gradient descent approach in this research, while clearly stating the assumptions considered.

  - In Section 2.4, I address the relevance of the hyper-parameters on how they contribute to performance and memory of the reservoir.
  - In Section 2.6, For active contours applied to image segmentation, I present both a specific and a generic theoretical context.

- Chapter 3 I provide a detailed research design. I begin with the formulation of neural network as levelset, and also provided the data acquisition procedure and evaluation metrics to validate our experiments.

- Chapter 4 I present results of various experiments performed on different databases and compare the performances of the CESN to CLSTM, CGRU, CRNN and a 3DCNN. I also investigate the relationship between the parameters of the reservoir and it's ability to learn the LS segmentation process.

- Finally, in Chapter 5, I conclude with a discussion of the implications of the analyses presented in this research, and discuss how the work detailed in it can be developed developed in future studies.

2 Background and Literature Review

2.1 Recurrent Neural Networks (RNNs)

2.1.1 Introduction

RNNs (recurrent neural networks) are neural networks that are used to learn sequential data (e.g., speech, videos or text) [30]. Their internal memory (self-hidden state) retains the temporal (i.e. time varying) state of an input sequence, allowing them to predict the corresponding future observations $\bar{\mathbf{y}}_\mathbf{n}$ based on the previous states (i.e. history) of the sequence.

2.1.2 Formalism

Consider an RNN with a discrete time [4] input signal $\mathbf{u}_n$, as shown in Figure 2. The current hidden state is given by:

$$\mathbf{x}_n = f(\mathbf{\Theta}^{in}\mathbf{u}_n + \mathbf{\Theta}\mathbf{x}_{n-1} + b) \tag{1}$$

[4] Discrete time refers to the discrete nature of sequences (i.e. they are made of data indexed by integers).

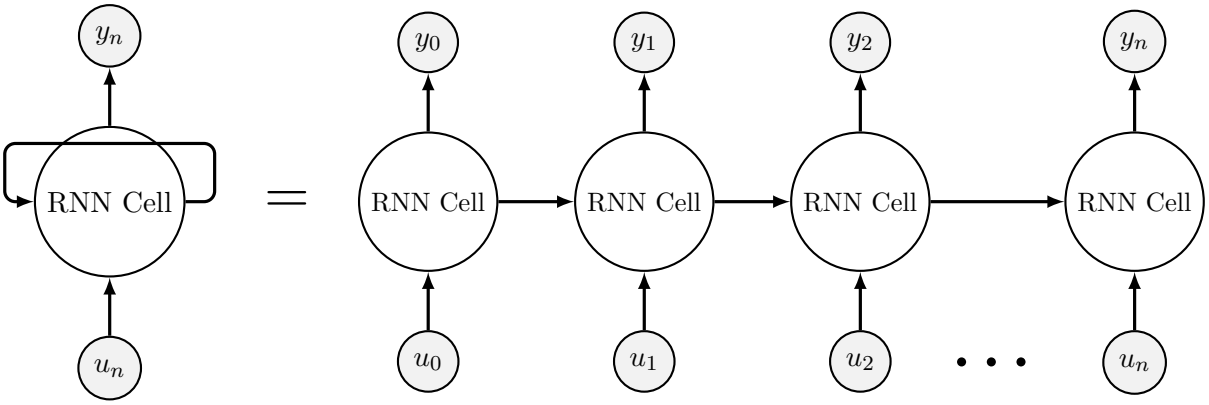

Figure 2: Schematic of an RNN. The hidden layer's recurrent connections enable information to be passed from one input to the next. The arrow indicate the information flow.

where $n = 1, ...N$, and N represents the number of instances in the training set $\{(\mathbf{u}_n, \bar{\mathbf{y}}_\mathbf{n})\}$, and x_{n-1} is the hidden recurrent state, which represents a state in the past (i.e. history). The input-hidden and hidden-hidden weight matrices, as well as a bias vector, are represented by $\mathbf{\Theta}^{in}$, $\mathbf{\Theta}$ and b, respectively. A sigmoid or hyperbolic tangent is a non-linear activation f. The experimenter chooses the initial hidden state x_1 , which is normally set to zero or random [56]. The output at any time n is computed as follows:

$$\mathbf{y}_n = f(\mathbf{\Theta}^{out} x_n) \tag{2}$$

where $\mathbf{\Theta}^{out}$ is the hidden-output weight matrices. One approach of training RNNs (i.e. learning optimal weights for a given task) is through back-propagation through time (BPTT) [59, 60], where the RNN is unfolded

through time and represented as a feed-forward network (with an unbounded number of layers) and back-propagation is applied on the unrolled RNN as shown in Figure 2.

However as discussed in Section 1.1, RNNs are unable to accurately capture long-term temporal dependencies (i.e. remembering state of longer sequences), and this is due to the gradient information either decaying or exploding exponentially during training. The vanishing and exploding gradients are two phenomena that have been well studied in RNN problems [56].

The exploding gradient problem refers to the significant increase in the norm (i.e magnitude) of the gradient during training, such that long-term temporal dependencies increase in magnitude exponentially than short-term temporal dependencies from one RNN cell state to another. In contrast to the vanishing gradient problem, where long-term temporal dependencies decrease exponentially to a magnitude close to zero, making it difficult to train RNNs to learn correlation between temporally distant events.

As a result, RNNs are unsuitable for problems requiring long-term memory (e.g., speech processing, text and video summarization), and thus suffer from short-term memory.

2.1.3 Long short-term memory (LSTM)

Long Short-Term Memory (LSTM) was introduced by Hochreiter et al. 1997 to overcome RNNs failure to handle with problems that involve long-term memory [63]. Because of the gating mechanisms implemented in the hidden state of LSTMs, they can preserve both short and long-term memory. These gating addresses the short-term memory dilemma in RNNs.

Consider the LSTM cell in Figure 3, at each time step n, three inputs are given to the LSTM cell, the current input signal $\mathbf{u}_n$, the short-term memory (*hidden state*) $\mathbf{x}_{n-1}$ and the long-term memory (*cell state*) $\mathbf{c}_{n-1}$.

The gates control the flow of information from one LSTM cell to another, ensuring that it is retained if it is significant and discarded if it is redundant at each time step n.

In this study, a traditional LSTM architecture is investigated [26]. Consider the following LSTM cell in Figure 3, which has three gates: an input gate $\mathbf{i}$, a forget gate $\mathbf{f}$, and an output gate $\mathbf{o}$ which are used to updates the memory $\mathbf{c}$ at any time n.

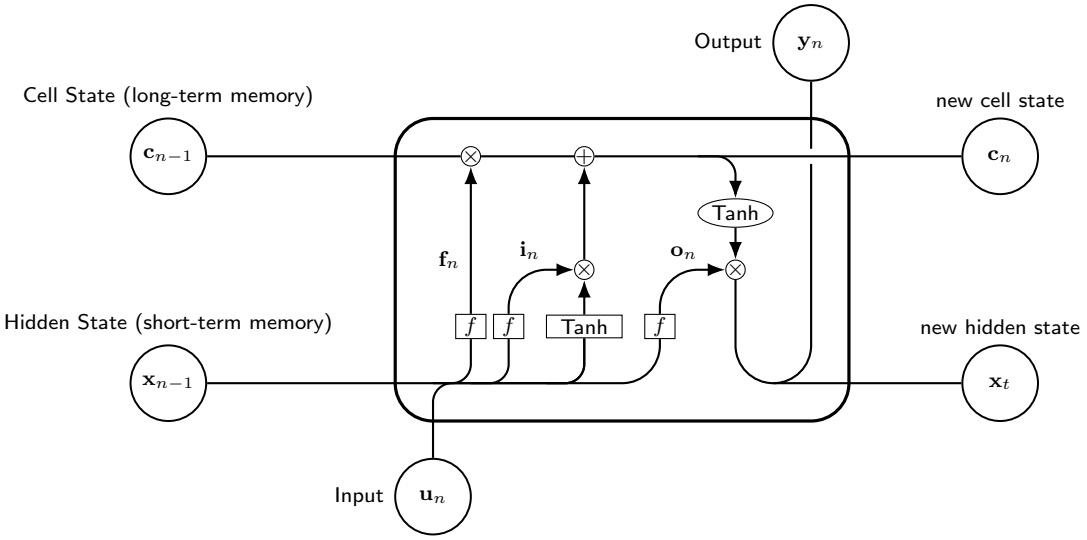

Figure 3: Schematic of an Long Short-Term Memory Cell.

The input gate determines which new state are preserved in the long-term memory $\mathbf{c}$. Given the most recent input input signal $\mathbf{u}_n$ and the previous hidden state $\mathbf{x}_{n-1}$, a sigmoid function f is applied to give values in the range [0,1], where values close to zero indicate discarded states, and one states that will be kept for the next step. Back-propagation is applied in this layer in order to learn which states to keep or remove. The states are computed as follows,

$$i_1 = f(\mathbf{\Theta}^{in}_{i_1}(\mathbf{x}_{n-1}, \mathbf{u}_n) + b_{i_1}) \tag{3}$$

In the second layer a *tanh* function is applied to $\mathbf{x}_{n-1}$ and $\mathbf{u}_n$.

$$i_2 = tanh(\mathbf{\Theta}^{in}_{i_2}(\mathbf{x}_{n-1}, \mathbf{u}_n) + b_{i_2}) \tag{4}$$

Multiplying the outputs of the preceding two layers decides which states should be preserved in the long-term memory, as follows:

$$i_{input} = i_1 \times i_2 \tag{5}$$

The forget gate determines whether long-term memory information ought to be retained or discarded, and it is calculated as follows:

$$i_{forget} = f(\mathbf{\Theta}^{in}_{i_{forget}}(\mathbf{x}_{n-1}, \mathbf{u}_n) + b_{i_{forget}}) \tag{6}$$

This new long-term memory is computed as follows,

$$\mathbf{c}_n = \mathbf{c}_{n-1} \times i_{forget} + i_{input} \tag{7}$$

Then the new short-term memory is generated by the output gate, and is computed as follows,

$$y^1 = f(\mathbf{\Theta}^{in}_{o_1}(\mathbf{x}_{n-1}, \mathbf{u}_n) + b_{o_1}) \tag{8}$$

$$y^2 = tanh(\mathbf{\Theta}^{in}_{o_2}\mathbf{c}_n + b_{o_2}) \tag{9}$$

$$y_n = y^1 \times y^2 \tag{10}$$

The recently computed long-term and short-term memory are fed from one LSTM cell to another. The short-term is exposed for each time step. However, these gating mechanism increases the number of parameters to be learning, therefore, training LSTMs is both computationally and time intensive.

2.1.4 Gated recurrent unit (GRU)

Gated Recurrent Units (GRUs) is another variate RNN, introduced by Kyunghyun et al. 2014 [6] to solve the vanishing and exploding gradients during back-propagation through time. GRUs, like LSTMs, use a gating mechanism to control the flow of information across cells [5]. But, unlike LSTMs, which have two states passed across cells, the cell and hidden state, GRUs only have one hidden state, resulting in fewer parameters to learn. As a result, they are faster to train than LSTMs and need less computing memory. The architecture in the gates in GRUs enable the hidden state to capture both long-term and short-term memory.

Consider the GRU cell in Figure 4, at each time step n, the cell is given two inputs, the current input signal $\mathbf{u}_n$ and the previous hidden state $\mathbf{x}_{n-1}$. The GRUs consist of two fundamental gates, the update z gate and the reset gate r. The update gate functions similar to the forget and input gate in LSTMs, it decide what information to discard and what new information to keep. The update is computed as follows,

[5] Cells are objects with a single scalar output.

$$\mathbf{z}_n = \sigma(\mathbf{\Theta}_z^{in}\mathbf{u}_n + \mathbf{\Theta}_z\mathbf{x}_{n-1}) \tag{11}$$

where $\mathbf{\Theta}_z^{in}, \mathbf{\Theta}_z$ are the input and internal weights respectively in the updated gate. The reset gate determines how much past information to forget and it is updated as follows,

$$\mathbf{r}(n) = \sigma(\mathbf{\Theta}_r^{in}\mathbf{u}_n + \mathbf{\Theta}_r\mathbf{x}_{n-1}) \tag{12}$$

where $\mathbf{\Theta}_r^{in}, \mathbf{\Theta}_r$ are the input and internal weights respectively in the reset gate. And lastly, *tanh* is applied to results in Equation 12 such that,

$$\tilde{\mathbf{r}}(n) = tanh(\mathbf{\Theta}^{in}\mathbf{u}_n + \mathbf{\Theta}(\mathbf{r}_{n-1}\mathbf{x}_{n-1}) \tag{13}$$

The linear combination between the previous activation $\mathbf{x}_{n-1}$ and $\tilde{\mathbf{r}}_n$ is the new hidden state, and it is computed as follows,

$$\mathbf{x}_n = (1 - \mathbf{z}_n\mathbf{x}_{n-1})) + \mathbf{z}_n)\tilde{\mathbf{r}}_n \tag{14}$$

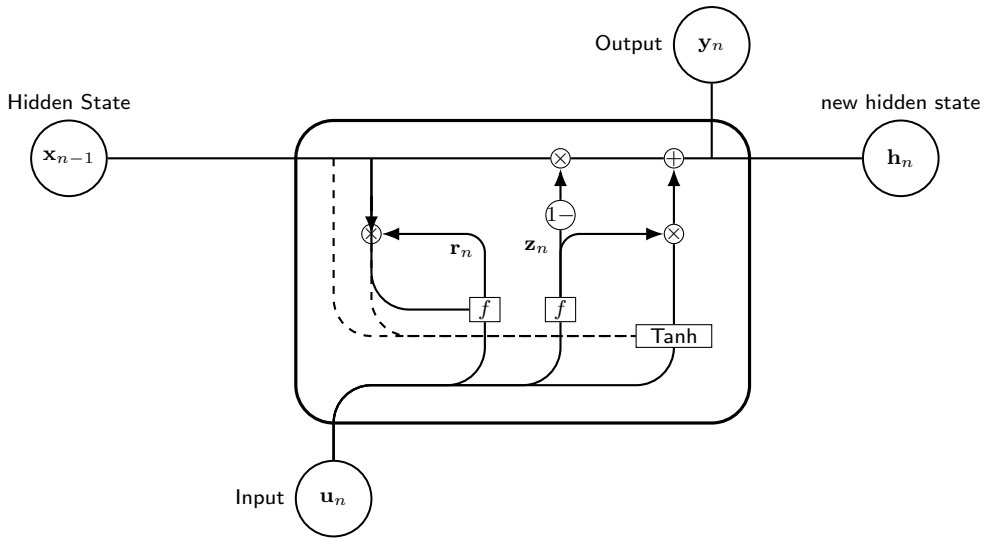

Figure 4: Schematic of an Gated Recurrent Memory Cell.

Compared to LSTMs, GRUs are faster and need less computing memory to train. LSTMs still remain state-of-the-art in long-term memory problems, especially in cases where the performance outweighs computational expense.

2.1.5 Conclusion

The fundamental distinction between regular RNNs and LSTMs/GRUs is that the former uses gating of mechanism, this gates determine when and how the hidden state is updated. Standard RNNs and LSTMs/GRUs still remain powerful tools for modeling sequential data, however training these models is computationally expensive and unstable during the optimization of larger hidden weights through back-propagation [32]. Therefore, a cheaper and efficient alternative is required for training RNNs.

2.2 Reservoir Computing (RC)

2.2.1 Introduction

Reservoir computing (RC) is a computing paradigm based on neural networks that allows for the efficient processing of temporal problems [19]. RC introduces a dynamic *reservoir* having a short-term memory [6] to transform features from the temporal inputs into a high-dimensional feature space. To learn the projected features for any temporal problems (i.e. classification and regression) a *readout* layer is used. Since only the *readout* layer is trained using a linear approach, RC systems can effectively capture auto-correlated events within the data, and training such systems is computationally cheap compared to RNNs which use back-propagation [25].

2.2.2 Formalism

Consider a temporal task [7] of learning a functional relationship f between a discrete time input signal $\mathbf{u}_n$ and a discrete time target output $\bar{\mathbf{y}}_\mathbf{n}$ such that the error $E(\mathbf{y}, \bar{\mathbf{y}})$ is minimized. Then the learned output $\mathbf{y}$ will be given by,

$$\mathbf{y}_n = f(..., \mathbf{u}_{n-1}, \mathbf{u}_n) \tag{15}$$

[6] Short-term memory is when the current outputs y_n depend on earlier values of the input u_{n-k} and/or earlier values from the outputs y_{n-k}.

[7] A temporal problem is one in which $\mathbf{u}_n$ and $\bar{\mathbf{y}}_\mathbf{n}$ are discrete time domain signals $n = 1, ...N$, and N, and the aim is to learn a functional $\mathbf{y} = y(..., \mathbf{u}_{n-1}, \mathbf{u}_n)$ that minimizes the error $E(\mathbf{y}, \bar{\mathbf{y}})$.

Most problems are non-linear, and solving them with a linear relationship between the input $\mathbf{u}_n$ and output signal $\mathbf{y}_n$ is difficult.

$$\mathbf{y}_n = \mathbf{\Theta}^{\mathbf{out}} \mathbf{x}_n \tag{16}$$

where $\mathbf{\Theta}^{\mathbf{out}}$ and $\mathbf{x}_n$ are the output (i.e learned) weights and the reservoirs activation's, respectively. This is because the error $E(\mathbf{y}, \bar{\mathbf{y}})$ is big regardless of $\mathbf{\Theta}^{\mathbf{out}}$.

Using the reservoir $\mathbf{x}_n$ to spatially transform the history input signal $\mathbf{u}_n$, one can resort in the transformation of the input u_n into a high-dimensional feature vector $\mathbf{u}_n \in R^{N_u}$, and this feature vector serves a transformed (i.e new) input that is used by a linear method. This approach is referred to as a *kernel* method with a called kernel function K, then $\mathbf{x}_n$ is in the form,

$$\mathbf{x}_n = K(\mathbf{u}_n) \tag{17}$$

The function to be learned in any temporal task depends on the past states of the input signal,

$$\mathbf{x}_n = K(..., \mathbf{u}_{n-1}, \mathbf{u}_n) \tag{18}$$

and returns the high dimensional current input signal and its potentially infinite previous states [43]. K has the potential of having an infinite number of parameters, practical implementations normally require the function to be defined recursively as,

$$\mathbf{x}_n = K(\mathbf{x}_{n-1}, \mathbf{u}_n) \tag{19}$$

We can then use $\mathbf{x}_n$ and a linear method in Equation 16 to approximate $\mathbf{y}_n$, as follows,

$$\begin{aligned} \mathbf{y}_n &= \mathbf{\Theta}^{\mathbf{out}}\, \mathbf{x}_n \\ \mathbf{y}_n &= \mathbf{\Theta}^{\mathbf{out}}\, K(\mathbf{x}_{n-1}, \mathbf{u}_n) \end{aligned} \tag{20}$$

The purpose of learning $\mathbf{y}_n$ is then to determine the output weights $\mathbf{\Theta}^{\mathbf{out}}$ such that the error $E(\mathbf{y}, \bar{\mathbf{y}})$ is minimized . It is important to state that the K function of Equations 18 and 19 represents a dynamic system driven by the input signal $\mathbf{u}_n$.

This method, along with several main streams of RC are described in more details below.

2.2.3 Echo state networks (ESNs)

Jaeger et al. 2001 suggested a radically new approach to RNN architecture and training called Echo State Networks (ESN) in 2001 [30]. The echo state approach is based on the observation that if the RNN is initialized at

randomly, then the training of the linear readouts is adequate to achieve optimal performance in practical applications with a simple model [43]. The untrained RNN is called a dynamic *reservoir* [8] and the resulting activated reservoir states $\mathbf{x}_n$ are referred to as the *echoes* of the input history as shown in Figure 5.

Definition 1. A Echo State Network *is a recurrent and random sigmoidal neural network which is excited by an input signal in the discrete time, and the activations of the neurons are treated with a linear method. The reservoir is a nonlinear dynamic filter that uses a high-dimensional temporal map to transform input signals. [30].*

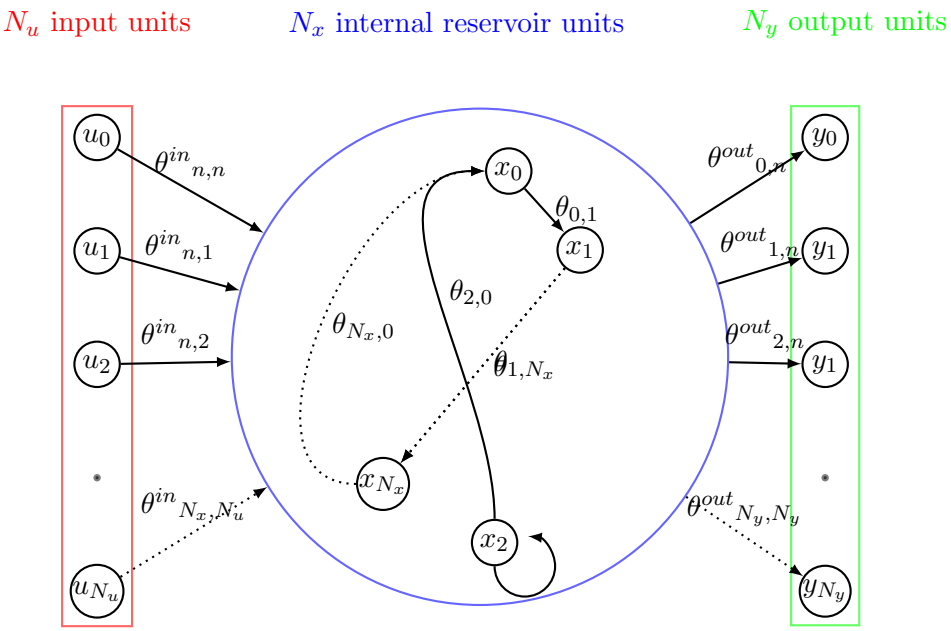

Figure 5: Schematic of an Echo State Network.

This new approach aims to overcome the shortcomings of RNN training mentioned in Section 1, by setting up RNNs in the as follows:

[8] The reservoir is similar to the hidden layer in RNNs.

- The RNN is created with random dynamic parameters and connections.
- The reservoir acts as a complex, high-dimensional, non-linear dynamical system.
- The input excites the reservoir, which preserves a non-linear transformation of the entire input history in its state.
- The reservoir has to be exponentially stable (i.e. fading memory over time).
- A linear combination of the input-reservoir signals are used to compute the target output signal.
- Optimal output weights are learned through linear methods (e.g., regression) as shown in Figure 5.

Consider the ESN shown in Figure 5, which is derived directly from Equation 19. The reservoir of the ESN is exited by input signal $\mathbf{u}_n$ with N_u input neurons, and the corresponding activations of N_x neurons in the reservoir at any time n are given by,

$$\mathbf{x}_n = (x_{1,n}, ..., x_{N_x,n})^T \tag{21}$$

similarly, the output activations or readouts $\mathbf{y}_n$ of N_y neurons in the output layer at any time n are given as,

$$\mathbf{y}_n = (y_{1,n}, ..., y_{N_y,n})^T \tag{22}$$

where $\mathbf{\Theta}^{in}$ are the input-reservoir connections, $\mathbf{\Theta}$ recurrent connections in the reservoir and $\mathbf{\Theta}^{out}$ are the reservoir-output connections , respectively defined as follows,

$$\mathbf{\Theta}^{in} = (\theta_1^{in}, ..., \theta_{N_u}^{in})^T \tag{23}$$

$$\mathbf{\Theta} = (\theta_1, ..., \theta_{N_x})^T \tag{24}$$

$$\mathbf{\Theta}^{out} = (\theta_1^{out}, ..., \theta_{N_y}^{out})^T \tag{25}$$

When the output to reservoir feedback is not taken into account. The following is a description of the reservoir's temporal neural state,

$$\mathbf{x}_n = f((1-\tau)\mathbf{x}_{n-1} + \tau(\mathbf{\Theta}^{in}\mathbf{u}_n + \alpha\mathbf{\Theta}\mathbf{x}_{n-1} + \lambda_n) \tag{26}$$

where n denote the discrete time, τ is the leaking rate, α is the spectral radius and λ_n is a regularization parameter (noise) and f is a nonlinear activation function, typically the positive sigmoid or the symmetric *tanh* [43]. Equation 26 represents a dependent dynamical system (i.e time-dependent dynamical systems) excitable by the external input signal $\mathbf{u}_n$.

The reservoir then functions as a non-linear time expansion mechanism with a memoryless output, allowing it to learn the output states $\mathbf{y}_n$ [61]. This implies that RC methods are able to understand,

- $\mathbf{x}_n$: as a vector expanding the history of the input signal $(..., \mathbf{u}_{n-1}, \mathbf{u}_n)$.
- $\bar{\mathbf{y}}_\mathbf{n}$: as a linear combination of reservoir neural state activation $\mathbf{x}_n$ and the approximated signal to be learned $\mathbf{y}_n$.

The readouts from the reservoir are linear, where the reservoirs state $\mathbf{x}_n$ is included as part of the input states $\mathbf{u}_n$. The outputs are given as linear combination of the neural states of the reservoir, input signal and the output weights expressed as,

$$\mathbf{y}_n = g(\mathbf{\Theta}^{out}[\mathbf{x}_n, \mathbf{u}_n]) \tag{27}$$

where g is the activation function, usually taken to be the unity and [ ] represents the vertical concatenation of vectors.

In supervised learning, the output weights $\mathbf{\Theta}^{out}$ are learned such that the error $E(\mathbf{y}, \bar{\mathbf{y}})$ is minimized at a given time [43]. The ESN's performance is determined by the reservoir's configuration. The ability of the ESN to approximate any temporal input, the reservoir should satisfy the *echo state property* stated in Definition 3.

A detailed discussions on the spectral radius is covered in Sections 2.5

2.2.4 Liquid state machines (LSMs)

Wolfang Mass et al. 2002 proposed a reservoir methods called *Liquid State Machines* (LSMs) independently of ESN [46]. LSMs origins are found in computational neuroscience , with the main focus of understanding computational properties of neural microcircuits [45, 46, 47, 51]. The LSM model is founded on the idea of multiple cycles that form a large complex network making up more than 80% of all synapses in a neocortical column and it demonstrates how multiple outputs can be learned to perform various tasks on the same series of states as shown in Figure 6 [61].

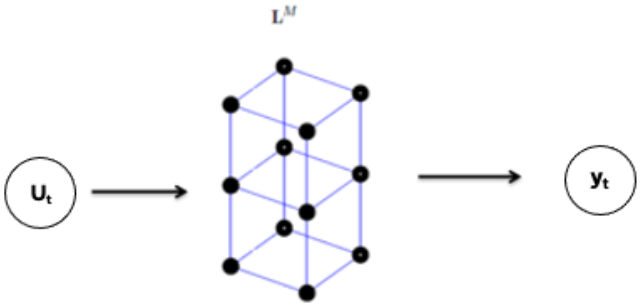

Figure 6: Liquid State Machine.

Definition 2. A Liquid State Machine *is a three-dimensional, locally connected network that is generated at random using biologically-based parameters influenced and excited by sequences of external inputs. Both neuron responses are projected to the next cortical layer, where learning takes place [46].*

In the LSMs, the reservoir is referred to as a *Liquid*, based on the intuitive comparison of excited states as ripples on the surface of a pool of water [43]. Formally, an LSM is characterized by the state of the liquid $\mathbf{u}_t^M$ at continuous time t, where M denotes relation to the liquid state machine. The Liquid activation are in the form,

$$\mathbf{u}_t^M = (\mathbf{L}^M \mathbf{x})_t) \tag{28}$$

where $\mathbf{L}^M$ denotes a liquid (e.g. dynamical system), $\mathbf{x}_s$ is the input signal at time $s \leq t$. The output $\mathbf{y}_t$ of the network can then be calculated using an output function f^M for the target function, given by,

$$\mathbf{y}(t) = f^M(\mathbf{u}(t)) \tag{29}$$

In literature the terms *reservoir* in ESNs and *liquid* in LSMs are used interchangeably in the context of reservoir computing. Although, several differences between ESNs and LSMs exist:

- The use of LSMs make a model based on spiking neurons, these neurons use a threshold that once exceeded by the accumulation of inputs produces a pulse at the output of the neuron. The outputs are pulse sequences based on a Poisson process. The ESNs primarily use sigmoid neurons, but they can also use other types of neurons.

- LSMs use a broader set of output type functions. A linear combination is widely used by ESNs, LSMs generally admit outputs in the form of layer networks [61].

- The internal structure of ESNs is mainly based on a random architecture, while that of LSMs is based on the observation of micro-neocortical columns and has the following structure of biological constraints [61]. The network nodes are placed on a 3D grid based on microcolumns located in the cortex as shown in Figure 6.

As previously mentioned, the reservoir in LSMs employs the Leaky Integrate Fire neuron model, which is a relatively basic spiking neuron model. The term 'leaky' is a threshold for which the liquid has a fading memory of the driving input.Spiking neurons in LSMs are more complex to implement, set up, and tune than ESNs, and they are often more costly to simulate on modern computers, so they are used less commonly in practice. [43].

2.2.5 Evolution of recurrent systems with linear output (Evolino)

Evolino fundamentally extends the idea of short-term memory ESN from a RNN to a LSTM. Evolino combines neuroevolution (i.e. the evolution of neural networks) and linear methods (e.g., linear regression) to solve time-dependent problems [50]. The readouts in Evolino is learned using evolutionary techniques [43].

2.2.6 Various types of reservoir

The reservoir can comprise of different neurons structures. Furthermore, different reservoirs exist due to the different topologies, and activation functions used to construct the reservoir. This includes, Quantized-ESNs which are ESNs whose node states take discrete units, where the number of possible states of a single unit is controlled by a parameter called quantization level [64]. SHESN (Scale-Free clustered ESN) are ESNs whose degree of connection follow a power law [61]. The NWESN, BAESN and MESN are respectively are those ESNs using the Newman-Watts' small-world model (small-world), BarabasiAlbert's scale-free model, and a mixed topology of these two models [17, 61].

2.2.7 Conclusion

Reservoir computing methods offer a cheaper alternative for training of RNNs, firstly by only training the *readout* layer which is computational inexpensive. Secondly, the reservoir does not use back-propagation to

learn the weights, therefore does not suffer from vanishing and exploding gradients. And also due to it's simplistic nature, it is easy to implement in hardware and prototyping.

2.3 Learning from the reservoir

2.3.1 Introduction

Learning weights associated with reservoir readouts can be done in two ways, through an online and offline approach. The offline method, or batch mode, consists in collecting all the states $\mathbf{u}(t)$ of the reservoir during the learning phase in a matrix $\mathbf{U} \in R^{N \times T}$, and all the target $\mathbf{y}(t)$ results in a matrix $\mathbf{Y} \in R^{M \times T}$, to compute the output weights $\mathbf{\Theta}^{out}$.

The online mode consists of calculating the weights at each new state of the reservoir and to update the output weights $\mathbf{\Theta}^{out}$. This section presents a brief introduction to three offline methods: linear regression, Moore-Penrose pseudo-inverse, and regularization of Tikhonov, followed by a brief overview of online methods.

2.3.2 Offline learning

In offline (i.e. batch) learning the weights associated with the reservoir readouts are updated after passing the entire training set.

2.3.2.1 Linear regression

Learning the output weight $\mathbf{\Theta}^{out}$ can be coined as solving a system of linear equations such that the error $E(\mathbf{Y}, \mathbf{\Theta}^{out}\mathbf{U})$ is minimized.

$$\mathbf{\Theta}^{out}\mathbf{U} = \mathbf{Y} \tag{30}$$

representing the above equation as a normal equation, we obtain

$$\mathbf{\Theta}^{out}\mathbf{U}\mathbf{U}^T = \mathbf{Y}\mathbf{U}^T \tag{31}$$

A naive solution to compute $\mathbf{\Theta}^{out}$ would be

$$\mathbf{\Theta}^{out} = \mathbf{Y}\mathbf{U}^T(\mathbf{U}\mathbf{U}^T)^{-1} \tag{32}$$

Since $\mathbf{Y}\mathbf{U}^T \in R^{M\times N}$ and $\mathbf{U}\mathbf{U}^T \in R^{N\times N}$ are independent to the number of training examples T, they can be computed iteratively as the training data passes through the reservoir [43]. As a result, the complexity of Equation 32 solution is independent of the number of training examples in space and time.

2.3.2.2 Moore-Penrose pseudo-inverse

The Moore-Penrose pseudo-inverse is an alternative method for solving 30 $\mathbf{U}^+$ of $\mathbf{U}$, and $\mathbf{\Theta}^{out}$ as

$$\mathbf{\Theta}^{out} = \mathbf{Y}\mathbf{U}^+ \tag{33}$$

This approach offers numerical stability compared to Equation 32, but is more expensive for large state collecting matrix $\mathbf{U} \in R^{N \times T}$, As a result, this technique is limited by the size of the reservoir N or the number of training examples.

2.3.2.3 Tikhonov regularization

The numerical stability of Equation 19 can be improved by using Moore-Penrose pseudo-inverse $(\mathbf{U}^T\mathbf{U})^+$ instead of the real inverse $(\mathbf{U}^T\mathbf{U})^{-1}$. In addition, this approach uses ridge, or Tikhonov as regularization, defined as follows,

$$\mathbf{\Theta}^{out} = \mathbf{Y}\mathbf{U}^T(\mathbf{U}\mathbf{U}^T + \alpha^2\mathbf{I})^{-1} \tag{34}$$

where $\mathbf{I} \in R^{N \times N}$ is the identity matrix and α is a regularization parameter. The regularization reduces the over-fitting during training.

2.3.3 Online learning

In online learning the weights associated with the reservoir readouts are updated iteratively (i.e per training example).

2.3.3.1 Gradient descent

Online learning aims to update $\mathbf{\Theta}^{out}$ over time rather than only once in the learning phase. The easiest method compute $\mathbf{\Theta}^{out}$ is to use stochastic

gradient descent, but this does not ensure convergence. The advantage of gradient descent algorithm is the low computationally complexity [37]. Consider the cost function (i.e. error function),

$$E = \frac{1}{2}\mathbf{e}_n^T\mathbf{e}_n \tag{35}$$

where $\mathbf{e}_n = \mathbf{y}_{target,n} - \mathbf{y}_n$ is the error. In the classical gradient descent algorithm, the weights are updated as follows,

$$\mathbf{\Theta}_{n+1}^{out} = \mathbf{\Theta}_n^{out} - \alpha\nabla_{\mathbf{\Theta}_n^{out}}\mathbf{E}_n \tag{36}$$

where α is a small positive constant, called the learning rate. The partial derivative of $\mathbf{E}_n$ with the output weight matrix $\mathbf{\Theta}_n^{out}$ must be derived, that is,

$$\begin{aligned}
\nabla_{\mathbf{\Theta}_n^{out}}\mathbf{E}_n =& \mathbf{e}_n\frac{\partial\mathbf{e}_n}{\partial\mathbf{\Theta}_n^{out}} \\
=& \mathbf{e}_n\frac{\partial(\mathbf{y}_{target,n} - \mathbf{y}_n)}{\partial\mathbf{\Theta}_n^{out}} \\
=& -\mathbf{e}_n\frac{\partial\mathbf{y}_n}{\partial\mathbf{\Theta}_n^{out}} \\
=& -\mathbf{e}_n\frac{\partial(\mathbf{\Theta}_n^{out}\mathbf{U}_n)}{\partial\mathbf{\Theta}_n^{out}} \\
=& -\mathbf{e}_n[\frac{\partial\mathbf{\Theta}_n^{out}}{\partial\mathbf{\Theta}_n^{out}}\mathbf{U}_n + \frac{\partial\mathbf{U}_n}{\partial\mathbf{\Theta}_n^{out}}\mathbf{\Theta}_n^{out}]
\end{aligned} \tag{37}$$

Where

$$\frac{\partial \mathbf{U}_n}{\partial \mathbf{\Theta}_n^{out}} = \begin{bmatrix} \frac{\partial \mathbf{u}_n}{\partial \mathbf{\Theta}_n^{out}} \\ \frac{\partial \mathbf{u}_n}{\partial \mathbf{\Theta}_n^{out}} \\ \frac{\partial \mathbf{y}_{n-1}}{\partial \mathbf{\Theta}_n^{out}} \end{bmatrix} \tag{38}$$

Let

Let δ denote the second term in Equation 37. Since the network's weights are selected from distribution in the range $[-1, 1]$ and $\mathbf{\Theta}$ is a sparse matrix with many zero elements, consequently $|\delta| << |e_n U_n|$ and thus it can be ignored $\delta \approx 0$. Thus, the output weights can be learned as follows:

$$\mathbf{\Theta}_{n+1}^{out} = \mathbf{\Theta}_n^{out} + \alpha \mathbf{e}_n \mathbf{U}_n \tag{39}$$

2.3.4 Conclusion

Learning the output weights is computationally cheaper both through the offline and online approach, compared to RNNs. However, the online approach has an advantage over the offline approach when working with larger reservoir sizes, since it is computationally cheaper [9].

[9] It should be noted that the computational expense in RNNs refers to the optimization of the weights during back-propagation while in reservoir computing it refers to the size of the network

2.4 Global parameters of the reservoir

2.4.1 Introduction

The construction of the reservoir depends on the parameters, and an optimal reservoir depends on the tuning of these parameters. The defining global parameters of the reservoir are the size of the reservoir (i.e. N), spectral radius, the sparsity of the reservoir, connectivity of the reservoir (i.e distribution of nonzero elements), input scaling (i.e. $\mathbf{\Theta}^{out}$) and the leaking rate.

2.4.2 Size of the reservoir

The size of the reservoir refers the number of hidden units in the reservoir. The larger the reservoir space $\mathbf{x}_n$, the simpler it becomes to find a linear combination of the predictions $\mathbf{y}_n$. The computational expense associated in training ESNs compared to RNNs techniques, reservoir sized of order 10^4 are common. In literature, larger reservoirs are mostly employed for demanding problems.

Nonetheless, computational trade-offs are important. Starting with a large reservoir from the beginning in order to achieve better performance is cumbersome since the optimization of reservoir parameters vary only slightly with the size, it is advisable to optimize them in a small reservoir and then to create a larger reservoir with the same parameters [64]. The number of independent real values that the reservoir must remember from the

feedback to effectively learn the task can be used to approximate a lower bound for the reservoir size N. Finally, in ESNs, the maximum number of storage values, known as *memory capacity*, is limited to the number of neurons N [61, 42].

2.4.3 Sparsity of the reservoir

The sparsity of $\mathbf{\Theta}$ ensures loosely coupled reservoir activation signals and reduces the cost of using large reservoirs. Jaeger et el. (2001) recommends making the reservoir connection sparse (i.e making most of the elements in $\mathbf{\Theta}$ zero). The sparsity of $\mathbf{\Theta}$ results in fast reservoir updates, irrespective of the reservoir size the computational cost of the network state update increases linearly with the size of the network instead of quadratically [42, 43].

A uniform distribution centered around zero is used to generate a random reservoir weight $\mathbf{\Theta}$ and input $\mathbf{\Theta}^{in}$ matrices, the former being more dense [30, 42]. Thus, with a representation of sparse matrices, a randomly connected reservoir result in faster reservoir update states.

2.4.4 Spectral radius

The spectral radius impacts the ESNs short-term memory, spectral radii close one increases the ESN's long-term memory [42]. A more detailed explanation is presented in Section 2.5.2.

2.4.5 Leaking rate

The leaking rate determines the rate at which reservoir dynamics are updated (i.e how slowly or how fast the reservoir neurons react to the incoming inputs signal $\mathbf{u}_t$) [42, 33]. For difficult problems which require more memory, a lower leaking rate is suitable and a high leaking rate is suitable for problems that require less memory [61].

2.4.6 Input scaling

The amount of non-linearity in the reservoir is regulated by input scaling [42]. Higher input scaling results in a reservoir based on the hyperbolic tangent *tanh* activation function's zero point. This indicates that the reservoir units will have been running in the *tanh* curve's linear regions. As a consequence, it is critical that the data be normalized; otherwise, memory loss can occur.

2.4.7 Conclusion

Setting the reservoir's hyper-parameter is critical for achieving achieve optimal performance, but it also requires domain expertise. These parameters are crucial for determining the echo state property, which is linked to the reservoir's chaotic nature and, as a result, its performance.

2.5 Reservoir dynamics and properties

2.5.1 Introduction

The generic approach to generate a reservoir in ESNs consist of:

1. Generating a random reservoir $(\boldsymbol{\Theta}^{in}, \boldsymbol{\Theta})$.
2. Computing the state of reservoir with input signal $\mathbf{u}_n$ and store the corresponding states and $\mathbf{x}_n$ in some reservoir state matrix $\boldsymbol{\Theta}$.
3. Computing the output weights $\boldsymbol{\Theta}^{out}$ from the collected states in (step 2), using a linear model (e.g., offline or online approach), while minimizing the error $E(\mathbf{y}, \bar{\mathbf{y}})$.

A significant amount of consideration is taken when developing the reservoir due to the optimization of many parameters that is required. Although, perhaps the most fundamental principal of a optimal reservoir in ESNs is the satisfying the *Echo state property* [30].

2.5.2 Echo state property and spectral radius

The reservoir is naturally a dynamic system, due to the recurrent connections in the reservoir, however chaotic properties are inherently adopted by the reservoir. Consequently, initial conditions tend to drive the reservoir instead of the input signal $\mathbf{x}_n$, this is often not desired since it results in sub-optimal performance.

Definition 3. (Echo State Property) *A network $F : X \times U \to X$ (with the compactness condition) has the echo state property with respect to U: if and only if for any left infinite input sequence $u^{-\infty} \in U^{-\infty}$ and any two state vector sequence $x^{-\infty}, y^{-\infty} \in X^{-\infty}$ compatible with $U^{-\infty}$, it hold that is $x_0 = y_0$ [30]*

Intuitively, the echo state say; if the ESN has been operating for a while the current reservoir state $\mathbf{x}_n$ is uniquely determined by the previous state of the input $\mathbf{x}_{n-1}$ and output $\mathbf{y}_{n-1}$. The existence of the *echo state property* is due to the fading effect of initial conditions on the network over time state in Theorem 1.

Theorem 1. *(State Contraction) A network $F : X \times U \to X$ (with the compactness condition) satisfies the echo state property with respect to U if and only if it has the uniform state contracting property,.i.e if there exist a null space $(\delta_L)_L \geq 0$ such that $\vee u^{-\infty} \in U^{-\infty}$, $\vee x^{-\infty}, y^{-\infty} \in X^{-\infty}$ compatible with $U^{-\infty}$, it holds [43].*

In order to create a non-chaotic reservoir, it is necessary to define conditions for this property in order to ensure its existence. This argument requires careful consideration because it has been the source of much misunderstanding in many journals, resulting in sub-optimal reservoir. We firstly look at the spectral radius as one of the parameters to establish the echo state property.

Definition 4. (Spectral Radius) *The spectral radius of the weight vector $\mathbf{\Theta}$, note $\rho(\mathbf{\Theta})$, is its largest eigenvalue in absolute terms.*

The *echo state property* is satisfied if the reservoir weight matrix $\mathbf{\Theta}$ is scaled such that its spectral radius $\rho(\mathbf{\Theta})$ satisfy $\rho(\mathbf{\Theta}) < 1$ [30]. The mathematical connection between the spectral radius and *echo state property* is that reservoirs using the *tanh* as neuron activation function and for zero input $\mathbf{u}_t = 0$ the condition $\rho(\mathbf{\Theta}) < 1$ is violated. The optimal values of $\rho(\mathbf{\Theta})$ is usually set depending on the amount of memory [43, 30].

2.5.3 Conclusion

The echo state property is a fundamental and necessary prerequisite for reservoirs to function optimally. However, there is some debate in the literature about the required and proper criteria for determining the reservoir's echo state based on the parameters mentioned in Section 2.4 [23]. While this property has led to a broader acceptance of ESNs due to their efficiency in training and stable gradients, the amount of parameters that must be optimized in order to satisfy the echo state property is more likely to result in sub-optimal performance, hence failing to outperform the existing state-of-the-art RNNs listed in Section 2.1 that have less parameters to optimize.

2.6 Image Segmentation

2.6.1 Introduction

Dividing an image into a set of pixels that represents meaningful regions that are simpler to analyze is referred to as *image segmentation*, as shown in Figure 7. In computer vision problems like object tracking and recognition, image segmentation is a critical processing phase. Over the years, there has been significant advances in developing image segmentation algorithms, however achieving accurate segmentation is still a challenge.

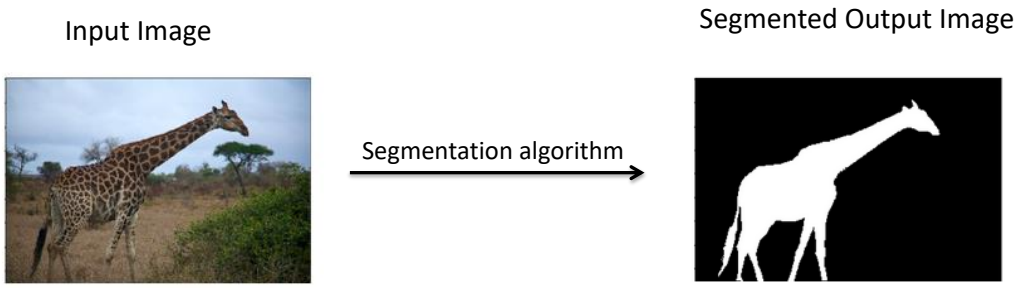

Figure 7: Binary Image Segmentation. The task is to group the pixels into foreground (i.e object of interest Giraffe) and the background.

There are five major categories of image segmentation: thresholding, edge, clustering, region pixel classification, and partial differential equation based segmentation methods.

Thresholding segmentation changes an image into a binary image using a threshold value [7, 36]. Thresholding is sensitive to noise and is unable to detect edges. Region-based segmentation groups pixels into homogeneous regions, this methods is both computationally and compute (i.e. slow) expensive. Clustering based segmentation groups similar pixels into clusters, however finding the optimal number of cluster is difficult and also it is computationally expensive. Edge-based segmentation, groups pixels based on discontinuities such as texture or intensity in the image, this method is computationally expensive and finding robust edges is non-trivial [74]. PDE based segmentation methods are considered in this study, due to their sequential nature in performing image segmentation and therefore can be framed into a sequential problem, such that a RNNs based methods may be used to learn the PDE process for performing image segmentation. PDE based segmentation are also computationally expensive.

2.6.2 Partial differential equations for image segmentation

Several partial differential equations (PDEs) image segmentation based methods have been implemented. This includes Geometric Active Contour Model (GACM) which are widely adopted for image segmentation [15, 85, 73, 2, 12, 14] and are categorized into region-based and edge-based models [8, 9, 54]. An energy functional of the image is formulated by these

models, thereby obtaining an active contour using variational principles [10] and gradient flow methods (i.e. gradient descent). By using difference methods an iterative formulation is derived, the choice in using gradient information to represent the levelset is what differentiate region-based and edge-based levelset formulation. The gradient in formation is used to control the evolution of the curve in edge-based levelset formulations, the boundaries are detected using a positive and decreasing function called Edge-Stopping Function (ESF) such that the boundaries of an object are detected when it equal zero, thereby stopping evolution of the curve [16].

However, in practice the ESP may never be zero of edges therefore unable to contours of the object. This is because digital images imposed as discrete gradient. In contrast to edge-based, region-based models take advantage of the statistical information, this make them less sensitive to noise especially for images with weak or without edges [16]. And also are not dependent on the location of the initial curve shown in red in Figure 9. Even though the region-based GACMs provide better segmentation results, they cannot effectively segment images with intensity inhomogeneity (i.e smooth intensity change inside originally homogeneous regions) [16].

[10] Variational principles allows the problems to be solved using calculus of variations.

2.6.3 Variational levelset (VLS)

The implicit implementation of active contours (AC) is called a variational levelset (VLS). The fundamental principles of using AC for image segmentation is to begin with an arbitrary initial closed form curve $\mathbf{C}$ as shown in Figure 9 in red. Therefore images-driven forces are used to expand or shrink the curve until the curves extends to the boundaries of the object and this procedure called *curve evolution* or *contour evolution*, denoted as $\frac{\partial \mathbf{C}}{\partial t}$.

Active contours are categorized into levelsets and snakes. Levelset methods the curve is defined as higher dimensional function than the image $\mathbf{I}$ and the curves are evolved implicitly, while snakes methods evolve predefined snake points based on an energy minimization scheme. Levelset based curve evolution models are a more flexible and convenient implementation of active contours, therefore, have a wider adoption in image processing [41].

Consider Figure 8, let $\mathbf{C}$ denote an initial curve that implicitly represent an active contour, within a higher dimensional function called a levelset function $\gamma(x, y) : \mathbf{\Lambda} \rightarrow \mathbb{R}$, such as $\mathbf{C} = (x, y) : \gamma(x, y) = 0, \forall (x, y) \in \mathbf{\Lambda}$, where $\mathbf{\Lambda}$ denotes the whole image plane.

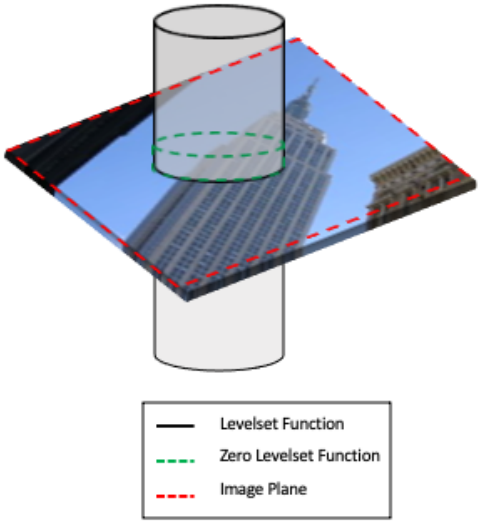

Figure 8: Representation of a implicit levelset. The image is defined on the image plane (red), the high dimensional levelset function (gray cylinder) and the zero levelset function (green) $\mathbf{C}$.

The intersection of the levelset function and the image plane is called a zero levelset function and it is used to represent the contour, the contour evolution is equal to the evolution of the level set function i.e., $\frac{\partial \mathbf{C}}{\partial t} = \frac{\partial \gamma(\mathbf{x},\mathbf{y})}{\partial t}$ as shown in Figure 8. The contour can be defined as the boundary between a positive and a negative field, which is why the zero levelset is used. As a result, everything on the zero area belongs to the contour, which is defined as follows using the oriented distance function:

$$\gamma(x) = \begin{cases} D(x, \mathbf{C}) & x \ inside \ \mathbf{C} \\ 0 & x \ on \ \mathbf{C} \\ -D(x, \mathbf{C}) & x \ outside \ \mathbf{C} \end{cases}$$

where $D(x, C)$ is the distance between any point and the curve. Chan-Vese proposed a widely adopted region-based active contour segmentation method [12].

2.6.3.1 Chan-Vese image segmentation

The Chan-Vese algorithm is a region-based segmentation algorithm that incorporates contour evolution, the levlset of Osterand Sethian's and the Mumford-Shan functional [12, 67, 9]. The gradient-dependent information is supplemented by a parameter based on region homogeneity, which provides the following benefits.

- it is possible to distinguish between contour regions with and without strong gradient detail.
- interior contours can be detected.
- the initial curved can be initialized anywhere on the image.
- the images is divided into two regions, object of interest as foreground and everything else as the background.
- Noise removal and other preprocessing steps are not necessary in advance.

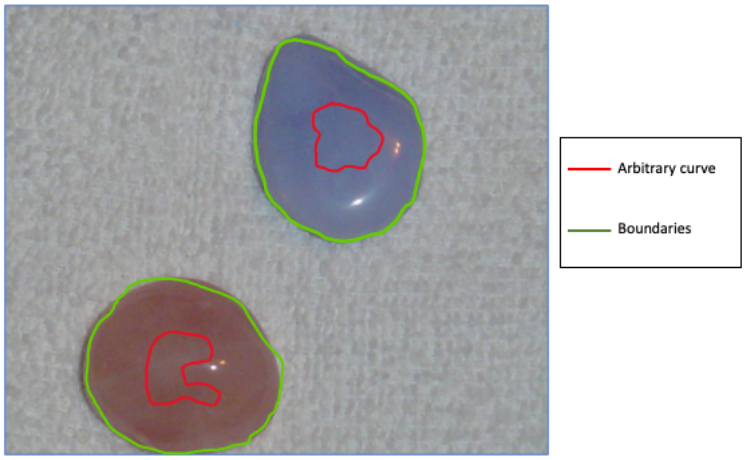

Figure 9: The image is defined on $\mathbf{\Lambda}$ (the big rectangle), the (arbitrary) red curve is $\mathbf{C}$.

Let $\mathbf{\Lambda} \in \mathbb{R}^2$ be a open and bounded set and $\mathbf{I} : \mathbf{\Lambda} \to \mathbb{R}$ be a difference image consisting of two homogeneous regions $\mathbf{\Lambda}_1$ and $\mathbf{\Lambda}_2$ ($\mathbf{\Lambda} = \mathbf{\Lambda}_1 \bigcup \mathbf{\Lambda}_2$). The Chan-Vese algorithm detects a contour that divides the difference image into two sub-regions $\mathbf{\Lambda}_1$ and $\mathbf{\Lambda}_2$, that defines the image's optimal piece-wise [11] constant approximation. By minimizing energy of the segmentation, the contour $\mathbf{C}$ is calculated[12],

$$\begin{aligned} E(\mathbf{C}, \epsilon_1, \epsilon_2) =& \lambda_1 \int_{\mathbf{\Lambda}_1} (\mathbf{I}(x,y) - \epsilon_1)^2 dxdy + \\ & \lambda_2 \int_{\mathbf{\Lambda}_2} (\mathbf{I}(x,y) - \epsilon_2)^2 dxdy + \mu Length(\mathbf{C}) \end{aligned} \tag{40}$$

where ϵ_1 and ϵ_2 represent the weighted intensities in the regions inside $\mathbf{\Lambda}_1$ and outside $\mathbf{\Lambda}_2$ $\mathbf{C}$ respectively, and λ_1, λ_2 and μ are the averaging parameters for the fitting terms and regularization, respectively. When μ

[11] A piecewise-defined function is a function that is defined by several sub-functions, each of which applies to a sub-domain of the main function's domain.

is small, objects of smaller size are also detected, when μ is large, larger objects are detected,

The Chan-Vese algorithm uses the level sets of Osher and Sethian not arbitrary point to control the contour $\mathbf{C}$ [9]. The contour $\mathbf{C}$ is a zero level set function γ over the domain of $\mathbf{\Lambda}$ that satisfy the following conditions:

$$\forall(x,y) \in \Lambda \begin{cases} \gamma(x,y) < 0 & in \quad \mathbf{\Lambda}_1 \\ \gamma(x,y) = 0 & on \quad \mathbf{C} \\ \gamma(x,y) > 0 & in \quad \mathbf{\Lambda}_2 \end{cases}$$

Using the Heaviside function Π, is defined by

$$\Pi(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$$

and it's distributive derivative $\sigma = \partial\Pi/\partial z$, thus the following is a representation of Equation 40,

$$\begin{aligned} E(\gamma, \epsilon_1, \epsilon_2) = \int_{\mathbf{\Lambda}} \Big(& \lambda_1 \mathbf{I}(x,y) - \epsilon_1)^2 (1 - \Pi(\gamma(x,y)) + \\ & \lambda_2 (\mathbf{I}(x,y) - \epsilon_2)^2 \Pi(\gamma(x,y))\Big) dxdy \end{aligned} \tag{41}$$

The energy functional $E(\gamma, \epsilon_1, \epsilon_2)$ can be minimized with respect to the constants ϵ_1 and ϵ_2 for a fixed γ, according to Osher and Sethian [9].

$$\begin{cases} \epsilon_1 = \Psi(\mathbf{I}) & \gamma \geq 0 \\ \epsilon_2 = \Psi(\mathbf{I}) & \gamma < 0 \end{cases}$$

The weighted pixel pixel intensity of image $\mathbf{I}$ is calculate by $\Psi(\mathbf{I})$ for a specific region. The Euler-Lagrange equation for the levelset γ can derived be derived by minimizing the energy functional $E(\gamma, \epsilon_1, \epsilon_2)$ in relation to to γ. The mean curvature and error for a constant ϵ_1 and ϵ_2 [8, 10] defined as,

$$\frac{\partial \gamma(x,y)}{\partial t} = \delta\gamma(x,y)[k\gamma(x,y) - \mu - \lambda_1(\mathbf{I}(x,y) - \epsilon_1)^2 + \lambda_2(\mathbf{I}(x,y) - \epsilon_2)^2] \quad (42)$$

where k, δ is the curvature and Dirac delta function respectively. The curvature k is given by,

$$k(\gamma(x,y)) = -div(\frac{\Delta\gamma}{|\Delta\gamma|}) = -\frac{\gamma_{xx}{\gamma_y}^2 - 2\gamma_x\gamma_y\gamma_{xy} + \gamma_{yy}{\gamma_x}^2}{({\gamma_x}^2 + {\gamma_y}^2)^{1.5}} \quad (43)$$

where $\partial_x\gamma_t$ and $\partial xx\gamma_t, \partial yy\gamma_t$ are the first and second derivatives of γ_t with respect to x and y directions. Intuitively the curvature k tells us by how much the tanget is changing along the curve. We use γ_t to denote γ at the tth iteration. Now, intuitively the curve evolution can be represented as time series process, such that γ_t is updates as,

$$\gamma_{t+1} = \gamma_t + \alpha \frac{\partial \gamma_t}{\partial t} \tag{44}$$

The VLS at time $t+1$ determined by the previous VLS at time t, as well as the curve evolution $\frac{\partial \gamma_t}{\partial t}$ with a learning rate α. The VLS is computationally costly and takes time to converge, this is because the it computed on the space as the image plane $\mathbf{\Lambda}$.

2.6.4 Variational level set as a trainable deep learning methods

The variational levelset segmentation are known to be sensitive to initial settings and highly depend on the number of iterations. Recent studies, have proposed reformulating the variational Levelset as an end-to-end trainable deep learning architecture applied to binary, instance and semantics segmentation [77, 86, 68, 28, 29, 40].

However, the end-to-end trainable deep learning VLS are a hybrid architectures that embed the levelset component within these deep learning architectures. And within these hybrid architectures the levelset is proposed as a preprocessing or postprocessing step under these formulation. These deep learning VLS architectures are computationally expensive and require longer training time, this is because to the number of of parameters introduced by the embedded levelset. Therefore, the formulation of the VLS as a trainable deep learning method is considered state-of-the-art in this study.

2.6.5 Conclusion

The levelset methods are computationally expensive and also the convergence rate is quite slow, this is due to the computation being performed on the same dimension as the image plane $\mathbf{\Lambda}$. Levelset methods are also highly dependent on the initialization of the velocities for propagating the levelset function, if the velocity flow is not constant the levelset function becomes strongly distorted, which mean that its gradient may become very large or very small around the edges [53].

Although there has been recent proposals of trainable deep learning Levelset approaches, these methods are extremely computationally expensive, because they incur the both the computational cost of the classical levelsets and that of deep learning methods under the formulation of the trainable deep learning levelset methods. Le et al. 2018 [39] suggest reformulating the levelset as a trainable recurrent architecture under GRUs called Recurrent Level Set (RLS), which is used as a benchmark in this study.

3 Method

3.1 Introduction

The methods and experimental setup are described in detail in this chapter. In Section 3.2, we begin by reformulating the variational levelset as a fully data-driven [12] learnable approach under deep learning frameworks.

We then provide details on the databases used for our various experiments in Section 3.3. We present and derive different metrics used in the iterative segmentation problem, as well as a comprehensive interpretation of each metric, in order to be able to determine the effectiveness of our approaches in Section 3.4. Finally, in Sections 3.5 and 3.6, we present and explain the formulation of learning the variational under a data-driven deep learning framework, as well as the optimization and data preprocessing techniques that were employed.

As a result, the two important components of this research are *data generation* and *image segmentation*, which will enable us to determine which of the proposed deep learning architectures are best suited to learning the variational levelset.

[12] Fully data-driven in this study, means that the variational levelset is learned from data that is generated by the variational levelset instead of an end-to-end deep learning VLS. End-to-end deep learning VLS combines the VLS and deep learning model as a hybrid model.

3.2 Formulating the levelset as a ESN

3.2.1 Introduction

The variational levelset introduced by Chan et al. 2000 [11] is used to demonstrate how to formulate the variational levelset as a data-driven learnable neural network (i.e ESN). As a result, one of our research questions is answered:

- If we consider the reservoir (i.e. recurrent layer) to have a similar updating mechanism as the variational levelset γ_t. How do we represent a single image $\mathbf{I}$ as spatiotemporal data X_t in ESNs?

We begin by formulating the overall learning procedure in Section 3.2.2, then explain how the data is generated in Section 3.2.3, finally answering the question in Section 3.2.4 by showing how to represent the data in our new formulation for learnable ESNs. Finally, the different architectures used to learn the variational level are presented in Section 3.2.5.

3.2.2 Formulation

Consider the following variational levelset (VLS) formulation under ESNs [13] shown in Figure 10:

[13] The ESN can be replaced with any deep learning method.

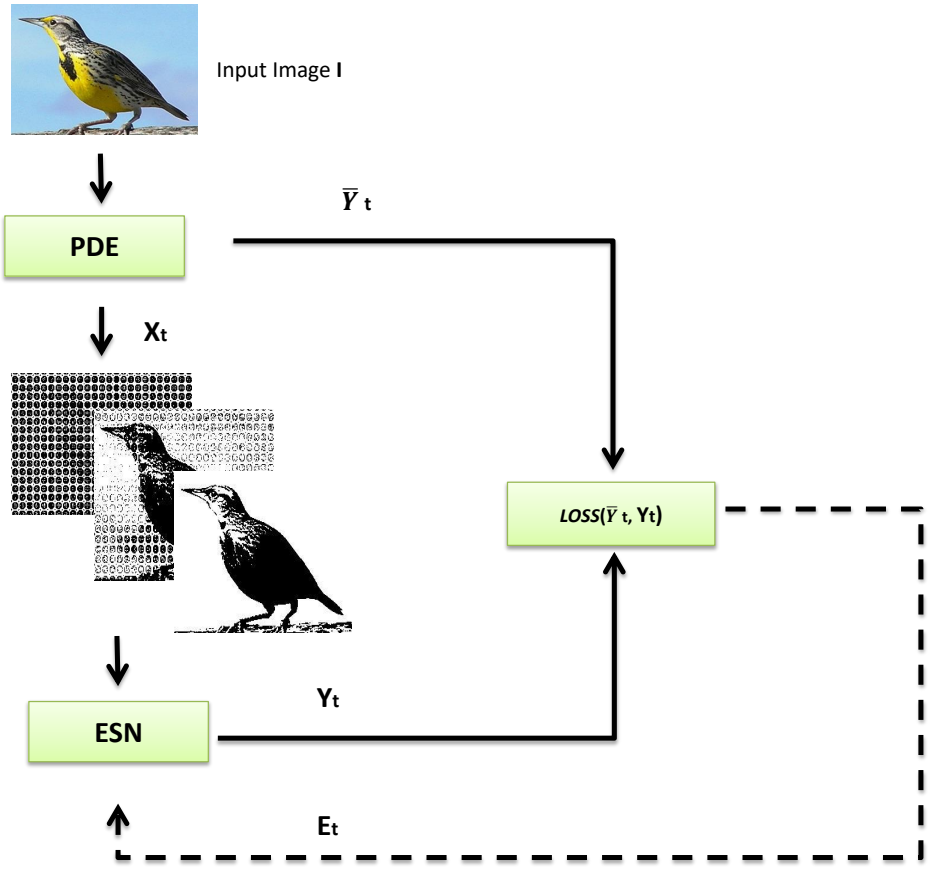

Figure 10: Iterative segmentation architecture.

- A single input image $\mathbf{I}$ is fed into the VLS (e.g., PDE in Figure 10), which produces a series of segmented images (i.e. spatiotemporal dataset) $\mathbf{X}_t$ of length t steps.

- The VLS generated images $\mathbf{X}_t$ serve as ground-truth samples $\mathbf{Y}_t$ for a deep learning model (such as ESN, as shown in Figure 10) to learn.

- The VLS segmentation mechanism is learned by using a stochastic gradient descent method to minimize the binary-cross entropy loss function (e.g., error) $Loss(\bar{\mathbf{Y}}_t, \mathbf{Y}_t)$, where $\bar{\mathbf{Y}}_t$ represents the predictions.

- The learnable parameters (e.g., weights) of the ESN are adjusted using the error/loss $\mathbf{E}_t$.

3.2.3 Data generation

Consider a grayscale 2D input image $\mathbf{I}$ and an initial binary mask $\mathbf{M}_0$. The binary mask $\mathbf{M}_0$ is used to initialize the VLS function γ_0 such that the outer boundary approximates the object of interest in the image $\mathbf{I}$ to be segmented. $\mathbf{M}_0$ is defined as a checkerboard with the same dimension as the input image $\mathbf{I}$ as shown in Figure 11. The initial binary mask is created using a binary step function because it is easier to generate new contours and faster to compute the gradients as compared to a signed distance map.

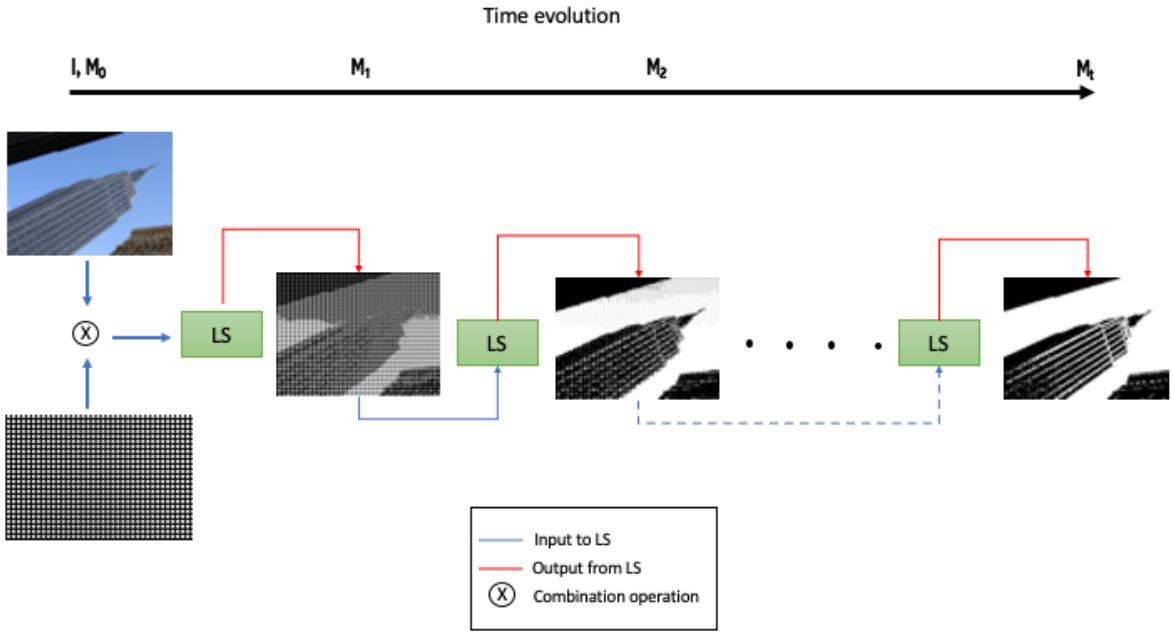

Figure 11: LS Generated training dataset. An input image $\mathbf{I}$ and initial binary mask $\mathbf{M}_0$, are feed into the VLS, and at each time step t a corresponding binary mask $\mathbf{M}_t$ is generated, where t is the number of iterations to the final optimal segmentation.

The inner $\mathbf{\Lambda}_1$ and $\mathbf{\Lambda}_2$ pixels, are computed as $\mathbf{\Lambda}_1$ being composed of $\mathbf{\Lambda}_1 = \mathbf{M}$, and $\mathbf{\Lambda}_2$ as the set $\mathbf{\Lambda}_2 = \mathbf{\Lambda} \setminus \mathbf{M}$. $\mathbf{\Lambda}_1$ is assumed to be a connected component [24]. The evolution of the curve is based on the gradient homogeneity and heterogeneity of the pixels to detect continuous objects and edges on the image space, by optimization of gradient descent to minimize the energy of the curve, such that $\mathbf{\Lambda}_1$ and $\mathbf{\Lambda}_2$ are computed for each step of the gradient descent, as mentioned in Section 2.6.2

In this experiment, the VLS parameters are initialized as follows: $\lambda_1 = \lambda_2 = 1$, a 2D convolution with a 3×3 kernel, stride of $\Delta x = \Delta y = 1$ is used to compute the gradients and $\Delta t = 0.5$ as the time step used for the gradient descent optimization of the energy functional. The weights of the length term $\mu = 0.2$. The number of iteration is 100, which means for each input image we generated 100 segmentation images [1]. These generated images form an iterative sequence for image segmentation using the VLS. The implementation of the levelset is adapted from [11, 75] and it is implemented in MATLAB [49]. Figure 11 shows the sample VLS generated dataset that is used to learn the VLS. As shown in the figure the segmentation has the same dimension as the input image, and it is initialized as a cheakerbox at iteration X_0. The evolution of the PDE evolved the curve to regions of uniform gradient.

3.2.4 Data representation

In Le et al. 2018 [39] the input image $\mathbf{I}$ is passed from the VLS module to the GRU when reformulating the VLS as Recurrent Level Set (RLS),

such that the VLS module pre-processes the input image and transforms it into a levelset image γ_t. When the 2-D levelset image γ_t is fed to the GRU module, we believe it is collapsed into a 1-D vector and represented as such.

However, since our approach is not an end-to-end RLS and the variational levelset is not embedded into deep learning architectures. Therefore, data representation forms an critical component for reformuating the levelset as a fully data-driven approach. We propose representing the input image $\mathbf{I}$ and levelset image γ_t as image channels, as shown in Figure 12, where $\mathbf{I}$ is the input image and $\mathbf{M}$ is the VLS generated segmentation image masks, such that $\mathbf{X}_t \in \{\mathbf{x}_0, \mathbf{x}_1, .., \mathbf{x}_t\}$, where $\mathbf{x}_t = [\mathbf{I}, \mathbf{M}_t]$ is the concatenation of the input image $\mathbf{I}$ and mask $\mathbf{M}$ at any time step t.

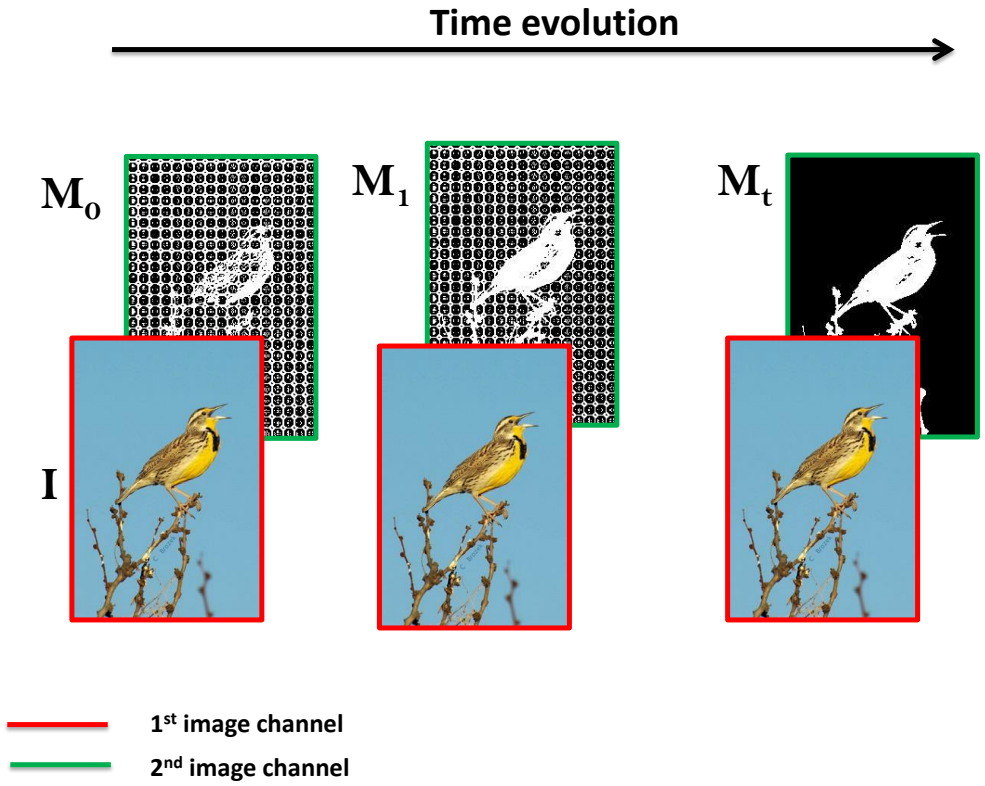

Figure 12: Input image representation.

In our representation, a single example is a 2-channel image. The first channel is a grayscale input image $\mathbf{I}$, and the second channel is the segmentation mask $\mathbf{M_t}$, as shown in Figure 12. The input image $\mathbf{I}$ and $\mathbf{M_t}$ implicitly represent the evolution of the variational levelset at any time step t, and can be considered as an alternative to embedding the variational levelset component in deep learning frameworks under deep variational levelsets methods.

Both the input and target images are resized to 64×64. The input images are passed into the model as a 2D vector of $64 \times 64 \times 2$ pixels, which then

are mapped to target pixels of 64 × 64. Convolutions are used to embed spatial features mentioned in Section 3.2.5.

3.2.5 Model architecture

We use two types of architecture specialized for spatiotemporal problems, a convolutional recurrent neural network (CRNN) and a 3D-convolutional neural network (3DCNN) since the VLS is formulated as a spatiotemporal problem. The CNN component in CRNNs is used to learn the spatial information of the VLS and the RNN component is used to learn the temporal evolution of the VLS [34, 76, 13]. 3DCNNs applies the convolution and pooling operations on volumetric spatiotemporal data [71, 34, 35]. Both these methods have are known for state-of-the-art results in spatiotemporal problems and thus are considered in the study.

The 3DCNN architecture consist of 2 3D-convolutional blocks (Conv3D → BatchNorm3D → ReLU → MaxPool3D →Dropout3D), followed by a 4096-units fully-connected layers and a 4096-units output layer, as shown in Figure 13. The input is a t×64×64, where t is the number of time steps. Both 3D-conv consist of 32 filters each with a kernel 3×7×7, stride 2×2×2, padding 0×0×0 and dilation rate 2×2×2. The conv1 and conv2 inputs channels are 2 and 32, respectively.

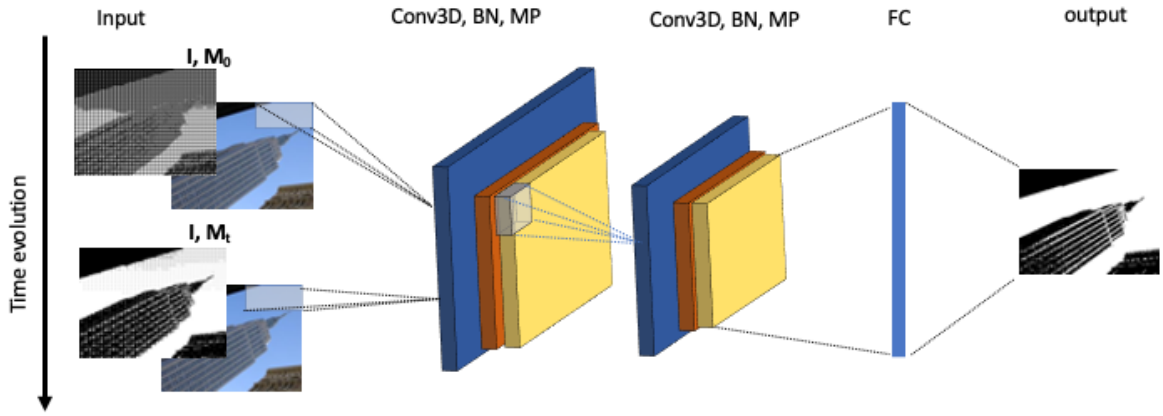

Figure 13: 3DCNN architecture.

The CRNN architecture consist of 4 2D-convolutional block (Conv2D → BatchNorm → ReLU → Dropout → Conv2D → MaxPool), an RNN layer with 512 ESN/GRU/LSTM units, followed by a fully-connected layer with 4096 units, and a 4096-units output layer, as shown in Figure 14. The input is a t×64×64, where t is the number of time steps. The first 2D-conv block consists of 2 inputs channels 32 filters each with a kernel 7×7, stride 2×2, padding 0×0, and dilation rate 2×2. The second 2D-conv block consist of 64 inputs channels 64 filters each with a kernel 1×1, stride 1×1 , padding 0×0 and dilation rate 1×1, The third 2D-conv block consist of 64 inputs channels 128 filters each with a kernel 7×7, stride 2×2 , padding 0×0 and dilation rate 1×1. The fourth 2D-conv block consist of 128 inputs channels 128 filters each with a kernel 1×1, stride 1×1 , padding 0×0 and dilation rate 1×1.

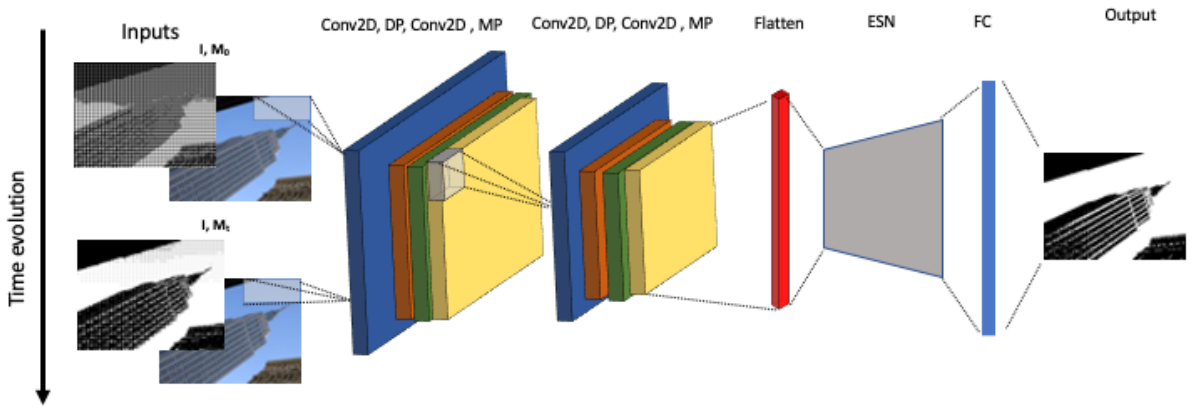

Figure 14: CRNN architecture.

3.3 Data acquisition

The Weizmann Segmentation Database (WSD), Berkeley Segmentation Dataset and Benchmarks (BSD), CIFAR-10, and CIFAR-100 databases were used in this study for model training, validation, and testing [3, 4, 38]. The CIFAR-10 consists of 60000 *32* × *32* RGB images divided into ten classes, each with 6000 images. The CIFAR-100 database is identical to the CIFAR-10 database, except it contains 100 classes, each with 600 images.

The WSD database consists 200 *300* × *200* RGB images that specifically represent one or two foreground object/s that are distinguished from their surroundings by intensity, texture, or other low-level cues. The BSD database has 500 *481* × *321* RGB images. These databases consist of natural images.

3.4 Model evaluation

To have a guided analysis for the segmentation task, the following metrics are defined in the context of learning the evolution of the variational levelset. Consider a segmentation task such that the positive class is the foreground pixels (1's) and the negative class is the background pixels (0's). The following metrics are defined as follows:

3.4.1 Confusion matrices

Consider the following confusion matrix between the predicted and the actual classified pixels. It describes the performance of a model on the validation and testing sets, where the actual true pixels are given. Table 1 shows a generic confusion matrix diagram such that:

		Predicted		
		Negative: Background pixels	Positive: Foreground pixels	Total
Actual	Negative: Background pixels	a	b	$a+b$
	Positive: Foreground pixels	c	d	$c+d$
	Total	$a+c$	$b+d$	N

Table 1: Confusion matrix.

- a is the number of correct predictions of a foreground pixel called true positive (TP).
- b is the number of incorrect predictions of a foreground pixel called false positive (FP).

- c is the number of incorrect predictions of a background pixel called false positive (FN).
- d is the number of correct predictions of a background pixel called true positive (TN)

3.4.2 Precision

Precision is the proportion of predicted foreground pixels in the segmentation results that match with the ground truth. Precision (P) is calculated as follows,

$$P = \frac{correctly\ Predicted\ Foreground\ Pixels}{total\ number\ of\ Predicted\ Foreground\ Pixels} = \frac{d}{b+d} \tag{45}$$

The precision is a function of FP, which means it measures the rate of falsely predicting foreground pixels as background, and therefore segmentation performance is sensitive to over-segmentation [48].

3.4.3 Recall

Recall is the proportion of ground-truth foreground pixels in the segmentation results that were correctly identified. Recall (R) is calculated as follows,

$$R = \frac{correctly\ Predicted\ Foreground\ Pixels}{total\ number\ of\ Actual\ Foreground\ Pixels} = \frac{d}{c+d} \tag{46}$$

The recall is a function of FN, which means it measures the rate of falsely predicting background pixels as foreground, and therefore the segmentation performance is sensitive to under-segmentation [48].

3.4.4 F1 Score

The trade-off between precision and recall is summarised using the F_1Score which is a value that can be described as the harmonic mean of precision and recall, it is calculated as follows,

$$F_1Score = 2 * \frac{P * R}{P + R} = \frac{2a}{2a + b + c} \tag{47}$$

The F_1Score takes FP and FN into account and is always between precision and recall, and is useful for contour or boundary matching between predicted segmentation and ground-truth segmentation.

3.4.5 Intersection over union (IoU)

For image segmentation problems, the IoU score is a widely used metric. The IoU is a metric that measures the similarities between ground-truth and predicted segmentation. It is defined by the equation below [58].

$$IoU = \frac{a}{a + b + c} \tag{48}$$

For object detection problems, the IoU can be used to calculate precision and recall for a given threshold. Such that, if the IoU for a predicted

segmentation is above the threshold, we can classify the prediction as True Positive (TP) and as a False negative if the IoU is below the threshold.

3.4.6 Conclusion

The ability of a model to solve a task is positively correlated to the score of the metric. The metrics used to evaluate the final results on the validation and testing data are precision, recall, F_1Score, and IoU. Recall and precision are related to under-segmentation and over-segmentation of the images. Over-segmentation results in low precision, while under-segmentation results in low recall [48].

IoU is used to measures the overall quality of boundary or contour segmentation, while F_1 is used to measures the trade-off between over-segmentation and under-segmentation. As seen in Equation 48 and 47, the distinction between IoU and F_1Score is that IoU penalizes inaccurate predicted segmentation classes more than. All metrics can be monotonic or positively correlated to one another.

3.5 Training and inference

Learning the variational levelset as a data-driven approach is formulated as follows: Given a sequence of input frames $\mathbf{x}_0, ...\mathbf{x}_t$ and target frames $\mathbf{y}_0, ...\mathbf{y}_t$ with $t = 100$ steps (i.e. samples), we treat the VLS evolution as a spatiotemporal problem such that given n previous frames we want to predict the $t + n$ frame, where $n = 1$ [52].

During training and inference, the input sequence is $\mathbf{x}_0, ...\mathbf{x}_t$ and the task is to predict $\mathbf{y}_{t+n}$, the model has to see the entire input history of length n steps as shown in Figure 12. The total number of time steps used is $t = 100$ and future steps $n = 1$.

3.6 Learning and optimization

3.6.1 Loss function

The binary-cross entropy is used as a loss function, defined as follows,

$$L = \sum_{i=1}^{2} [\bar{Y}_i log(Y_i) + (1 - \bar{Y}_i) log(1 - Y_i)], \tag{49}$$

where $\bar{Y}_i$ and Y_i represent the ground-truth and predicted class i. Intuitively the cross-entropy measures the amount of information (i.e.bits) required to represent a given pixel (i.e. background or foreground) from a probability distribution $\bar{Y}_i$ as that of probability distribution Y_i. And thus a higher cross-entropy value results in a model that is unable to distinguish between the background and foreground pixels, in contrast to a lower cross-entropy value.

To ensure that the model learns, we need to minimize the loss function and stochastic gradient descent with momentum is used as the optimization functions as discussed in Section 2.3. .

3.6.2 Optimization

To apply the previously mentioned techniques, we used a parallel hyper-parameter search optimization on a cluster of compute nodes to choose the optimal parameters of the CESN in a search space given by a leaking rate $lk \in \{0.0078125, 0.078125, 0.78125, 1\}$, sparsity $s \in \{0.2, 0.4, 0.6, 0.8\}$, spectral radius $sp \in \{0.09, 0.9, 1\}$ and input scaling $is \in \{0.5, 1\}$. The total number of hidden (reservoir) neurons is 4096. We use Stochastic Gradient Descent (SGD) to optimize our models on a computing cluster. We use mini-batches of 32 examples, momentum of 0.9, and weight decay of 0.1.

The models are initialized with learning rates of 0.1 this value is further reduced for $\frac{N_{epoch}}{5}$ steps during training, where N_{epoch} is the number of epochs. Parallel runs for each parameter search, are monitored using Tensorboard, and the optimal parameters are selected based on the minimization of the validation loss and maximization of the IoU, for each of the databases used presented in 6, 8, 4 and 2

3.6.3 Data augmentation and preprocessing

The total number of generated VLS examples for each database WSD, BSD, CIFAR-10, and CIFAR-100 is 20 000, 50 000, 5 000 000, and 6 000 000, respectively. However, due to the computational resources constrained in terms of computing time, the CIFAR-10 and CIFAR-100 datasets

were limited to only 200 000. Each database was randomly split into the training, validation, and testing set of 70%, 10%, and 20%, respectively.

CIFAR-10 and CIFAR-100 training, validation, and testing set sizes are 140 000, 20 000, and 40 000, examples respectively. And the sizes of the WSD training, validation, and testing sets are 14 000, 2 000, and 4 000, examples respectively. The sizes of the BSD training, validation, and testing sets are 35 000, 5000, and 10,000, examples respectively.

To reduce the effect of over-fitting we applied data augmentation. Before presenting an example to a model, we pre-process all the images by resizing them to 64×64 pixels, randomly flipping the images horizontally, vertically with probability 50%, adding Gaussian noise with a variance of 0.01, and randomly rotating the images for different angles $a \in \{60°, 90°, 180°, 270°, 360°\}$. These preprocessing and data augmentation procedures are done such are they are similar to Le et al. 2018 [39].

And lastly the pixels of the input images $\mathbf{I}$ as shown in Figure 12 are normalized between [-1, 1], as follows,

$$x = \frac{x - X_{mean}}{X_{std}} \tag{50}$$

where, x, X_{mean} and X_{std} is the input image, mean and standard deviation images of the entire training set. A total number of 500 epochs is used, with early stopping, to prevent over-fitting.

4 Results and Discussion

4.1 Introduction

In this chapter, we present the experimental results and thorough analysis, which addresses a research question posed in Section 1.2, namely:

- Which of our proposed deep learning architectures is the most effective at learning the variational levelset? Is this deep learning architecture better than the current existing state-of-the-art deep learning variational levelset methods?

This is achieved by comparing the performance of three recurrent neural networks (RNN, LSTM, and GRU) described in Chapter 2.1 with that of an ESN described in Chapter 2.2. However, as discussed in Section 3.2.5, we compare architectures specialized for learning the spatiotemporal evolution of the VLS, such as the convolutional RNN (CRNN), convolutional LSTM (CLSTM), convolutional GRU (CGRU), and 3DCNN, to a convolutional ESN (CESN). This investigation is conducted on four databases, WSD and BSD, which are small segmentation databases, and CIFAR-100 and CIFAR-10, which are larger databases.

We also investigate the configuration of the reservoir's global parameters mentioned in Sections 2.4 and how they impact the performance of the ESN introduced, as well as the echo state property mentioned in Sections 2.5.2,

and how they affect reservoir performance, as well as short-term and long-term memory. As a consequence, we can answer the following research question:

- Is it possible to empirically determine which set of optimal parameters satisfy the echo state property described in Section 2.5.2?

4.2 WSD

4.2.1 Introduction

In this experiment, we consider the WSD database, where there is a clear distinction between the foreground and background, the images consist of one or two objects. The database consists of 200 images in total and therefore the smallest database in our experiments.

4.2.2 Results and discussion

Table 2 shows the optimal hyper-parameters for each model selected based on maximizing the IoU on the validation set.

Table 2: WSD dataset optimal parameters for the CESN, CRNN, CLSTM, CGRU and 3DCNN

Model	Hidden size	Leaking rate	Sparsity	Spectral radius	Input scaling
CESN	4096	0.0713	0.8	1	1
CRNN	4096	-	-	-	-
CLSTM	4096		-	-	-
CGRU	4096	-	-	-	-
3DCNN	4096	-	-	-	-

In Table 2, the number of fully-connected hidden units for the CESN, CRNN, CLSTM, CGRU, and 3DCNN are **4096** as stated in Section 3.6.2, and the optimal leaking rate, spectral radius, sparsity, and input scaling for CESN are **0.0173**, **1**, **0.8**, and **1**, respectively.

The training and validation loss curves for the CESN, CRNN, CLSTM, CGRU, and 3DCNN are shown in Figure 15 as a function of the number of epochs [14].

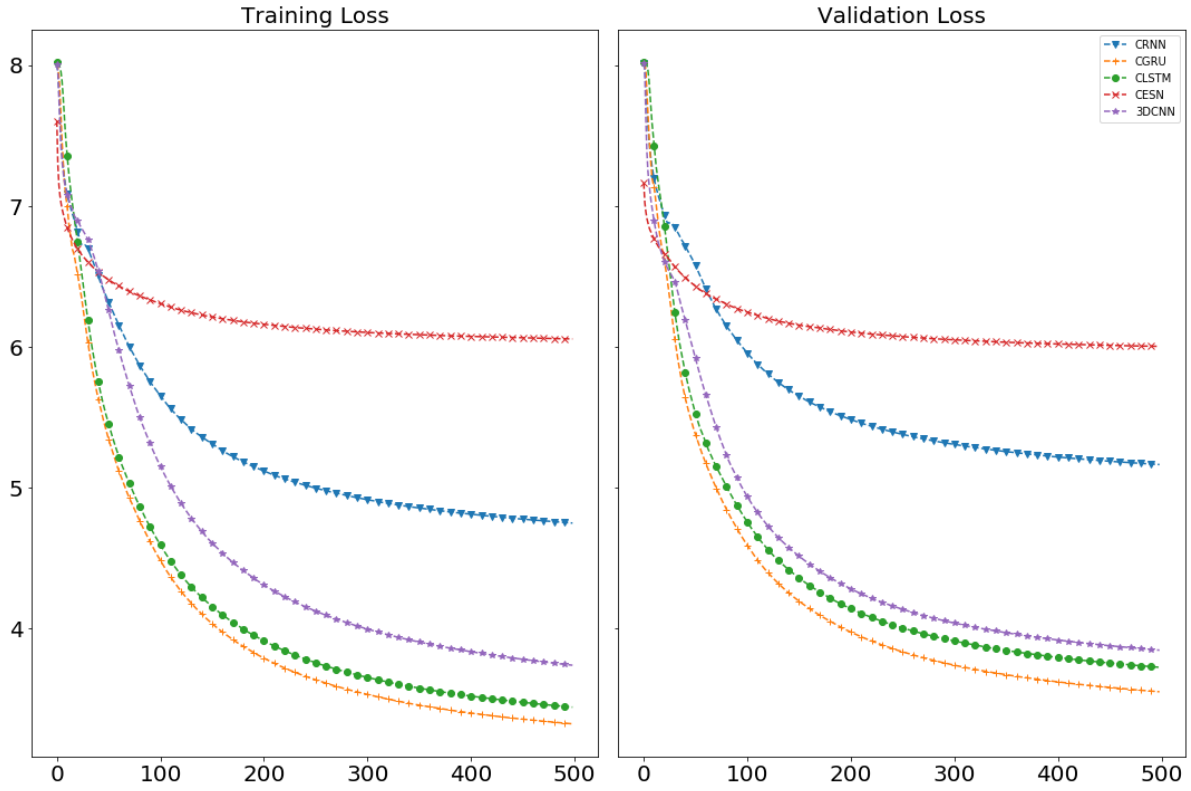

Figure 15: WSD validation loss curves for the CESN, CLSTM, CGRU, CRNN and 3DCNN.

As shown in Figure 15, in the early training stages, the models [15] poorly define the relationship between training inputs to their corresponding

[14] Epochs is the number of data passes during training.
[15] The CESN, CRNN, CLSTM, CGRU, and 3DCNN are referred to as **models** for brevity.

ground-truth, this is observed with the high error rate in the learning process.

As the networks learn to generalize better parameters and describe the relationship between the inputs and the corresponding ground-truth segmentations, the error begins to decrease, this can be attributed to 1) a decaying learning rate that aids in the convergence of models to global minima and the avoidance of oscillation [84] 2) dropout connections with a probability of **50%** between the convolutional layers help with overfitting, especially when working deep learning models that have a large number of parameters as the one considered in this study [27].

Both the validation and training losses decrease for all the models, this indicates their ability to learn the evolution of the VLS. The CESN has the lowest starting loss, but a low convergent rate in comparison to CRNN, CLSTM, CGRU, and 3DCNN.

Figure 16 shows the precision, recall, F_1Score and IoU for the CESN, CRNN, CLSTM, CGRU and 3DCNN on the validation set.

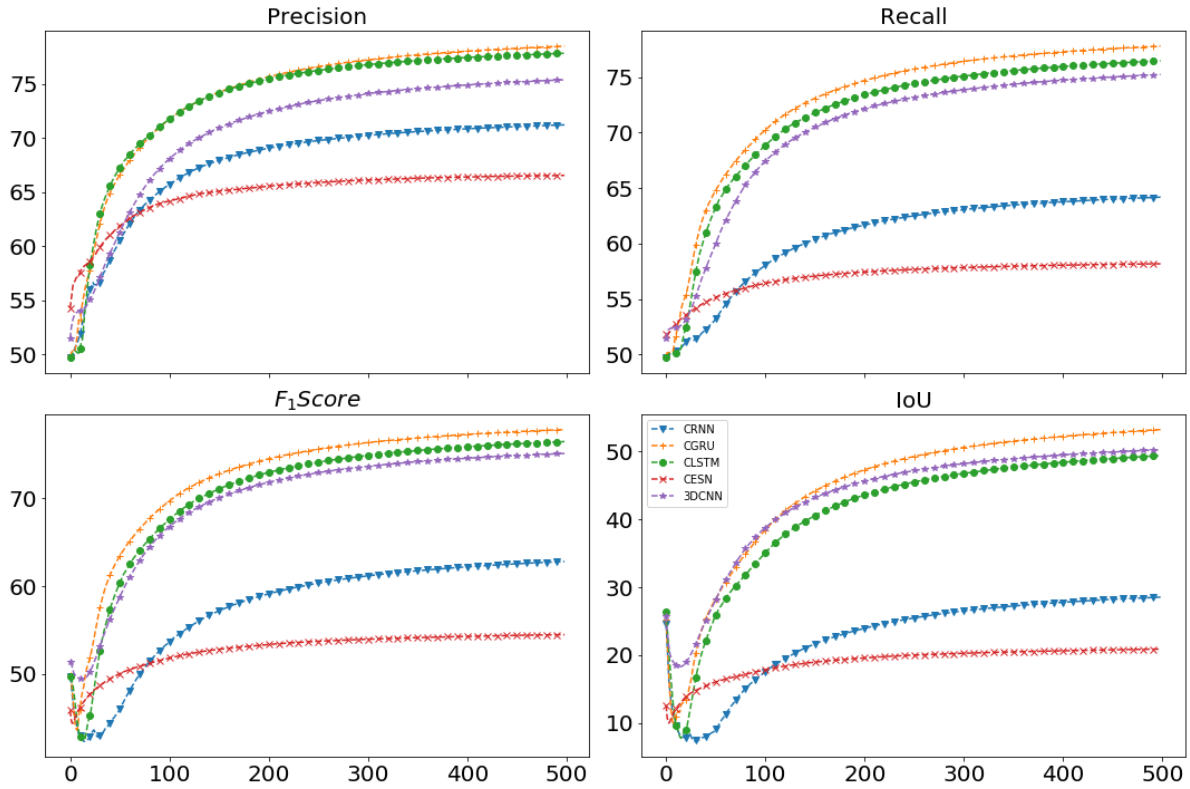

Figure 16: WSD validation performance curves for the CESN, CLSTM, CGRU, CRNN and 3DCNN.

In Figure 16, we observe that with a single pass of the data, the models have a lower performance, but with further training, their performance improves until it reaches a point where further training has no effect on the performance.

The CGRU has the highest IoU, F_1Score, recall, and precision, followed by the CLSTM, 3DCNN, CRNN, and CESN. The precision is higher than the recall for all the models, which implies that the models are more sensitive to under-segmentation than over-segmentation [65, 72] as previously discussed in Section 3.4. The F_1Score is higher than IoU for all the models,

this makes sense since there is a heavy penalty for incorrectly classifying foreground pixels as background pixels by the IoU.

We used a threshold [16] of **50%**, such that if the probability of the predicted segmentation classes is above the threshold the pixels are considered foreground (i.e object of interest) and background if less than the threshold. The average IoU for all the models during the entire training procedure is less than the threshold, this implies that the majority of predicted pixels are false negatives, i.e., background pixels are incorrectly predicted as foreground, resulting in under-segmentation.

Table 3 shows the performance of the CESN, CRNN, CLSTM, CGRU and 3DCNN on the testing set.

Table 3: The performance of the CESN, CLSTM, CGRU, CRNN and 3DCNN on the testing set.

Methods	**IoU**	**Precision**	**Recall**	$F_1 Score$
CGRU	**64.3%**	**76.8%**	**79.3**	**76.8%**
3DCNN	61.9%	75.5%	77.0%	76.0%
CLSTM	61.6%	75.5%	79.2%	75.7%
CRNN	37.4%	60.6%	66.5%	63.7%
CESN	21.9%	53.0%	63.2%	58.0%
White Noise	35.4%	50.0%	50.0%	48.2%

[16] The threshold value is used a cut-off between foreground and background pixels.

White noise [17] is measured on the test set as shown in Table 3, and to measure the predictability of the evolution of VLS. If the IoU of a model is less than that of white noise, then temporal states [18] of the VLS cannot be predicted by the model.

With an IoU of **64.3%** the CGRU is the best performing model, followed by the 3DCNN, CLSTM, CRNN, and CESN, with an IoU of **61.9%**, **61.6%**, **37.4%** and **21.1%**, respectively. The CESN is unable to learn the temporal states of the VLS, this is seen with an IoU less than white noise. The CGRU, CLSTM, 3DCNN, and CRNN are able to capture temporal states, however, since the recall is higher precision, the models are prone to over-segmenting. These observations can be due to the threshold value used and most definitely the small data size.

Figure 17 shows sample segmentations, the 1st, 2nd, 3rd, 4th, 5th, 6th and 7th columns are the input images, ground-truth (i.e. VLS), CESN, CRNN, CLSTM, CGRU, and 3DCNN, respectively.

[17] If a time series is white noise, it is a sequence of random numbers and cannot be predicted.

[18] Temporal states of the VLS refers how the initial conditions of the levelset γ affect the trajectory of the segmentation process.

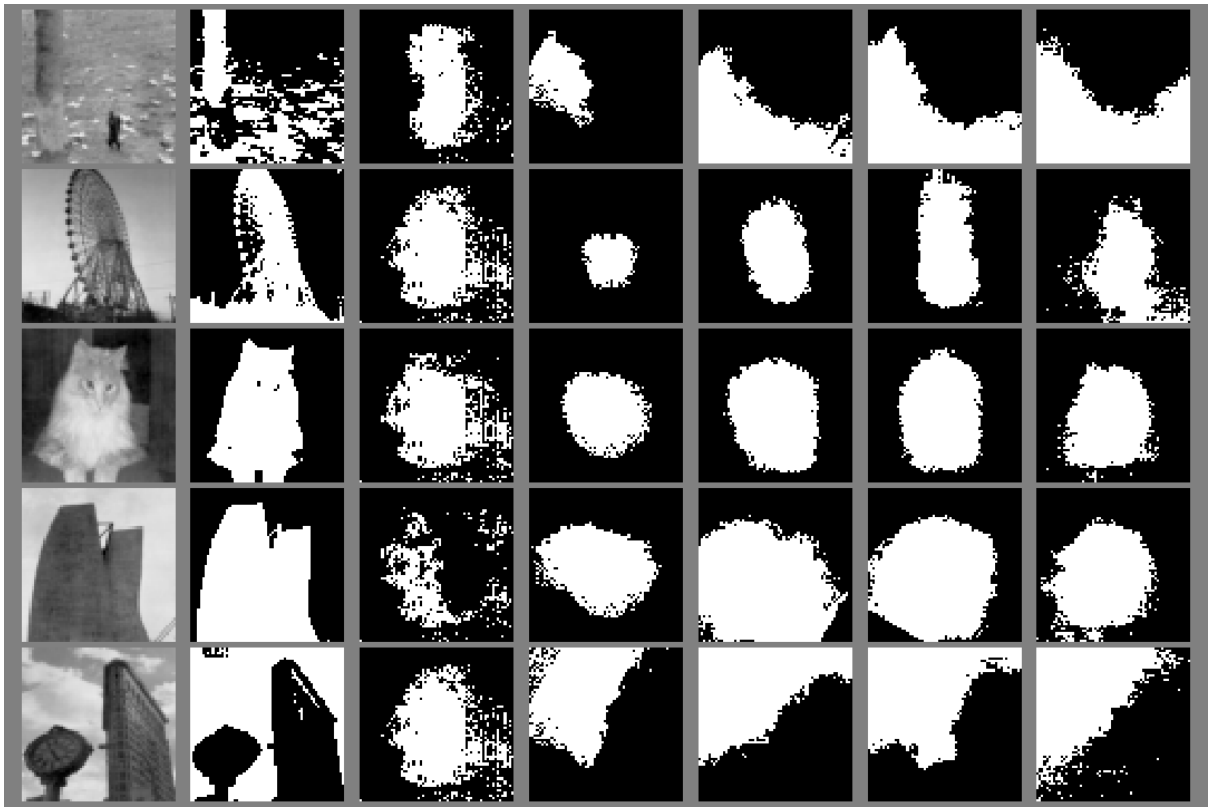

Figure 17: WSD examples of iterative segmentation. 1st column input, 2nd Chan-Vese, 3rd column CESN, 4th CRNN, 5th column CLSTM , 6th column CGRU , 7th column 3DCNN.

In 1st, 2nd, 3rd and 4th row, the models are able to localise the object in the image, but due to over-segmentation fine structures of the segmentation are lost. The image on 5th row is inverted and incorrectly segmented by the VLS, the CRGU can differentiate the two objects, this is not surprising since the majority of the pixels are backgrounds pixels due to class imbalance.

4.2.3 Conclusion

Considering that the WSD database consists of only 200 images, this means that 200 different sequences each with a length of 100 steps of

the VLS segmentation, and including the augmented images. Since only a fraction of these samples, **80%** were used to train the models, their generalization capability is limited by the size of the training data. Since the IoU of white noise is **35.4%** and less than our threshold of **50%**, this means the noise of the segmentation images is insignificant. Therefore, the observed CESN poor performance is due to the optimal hyper-parameters in Table 2, do not satisfy the ESP presented in Section 2.5.2, even though the spectral radius which is less than one as recommended in the literature, although this is not a necessary and sufficient condition [30, 31]. We will look at it in more detail in Section 2.4. The observed CRNN's poor performance is attributed to the exploding and vanishing gradients [56].

Therefore, CGRU is the best architecture for learning VLS under the conditions of the experiment. The CGRU does not outperform the state-of-the-art deep recurrent levelsets (RLS) proposed in Le et al. 2018 [39], which achieves an $F_1 Score$ of **99%** on the WSD for binary image segmentation.

4.3 BSD

4.3.1 Introduction

In this experiment, we use the BSD database images and which consist of 500 images. The images in this database consist of images of more than one overlapping object. Therefore, the BSD database is more complex and has more samples compared to the WSD database.

4.3.2 Results and discussion

Table 4 shows the optimal hyper-parameters for each model selected based on maximizing the IoU on the validation set.

Table 4: BSD optimal parameters for the CESN, CRNN, CLSTM, CGRU and 3DCNN.

Model	Hidden size	Leaking rate	Sparsity	Spectral radius	Input scaling
CESN	4096	0.0713	0.8	0.9	0.5
CRNN	4096	-	-	-	-
CLSTM	4096		-	-	-
CGRU	4096	-	-	-	-
CGRU	4096	-	-	-	-

In Table 4, **0.9** and **0.5** are the CESN's optimum spectral radius and input scaling, respectively. The other parameters are the same as the previous experiment in Section 4.2.

The training and validation loss curves for the CESN, CRNN, CLSTM, CGRU, and 3DCNN are shown in Figure 18 as a function of the number of epochs.

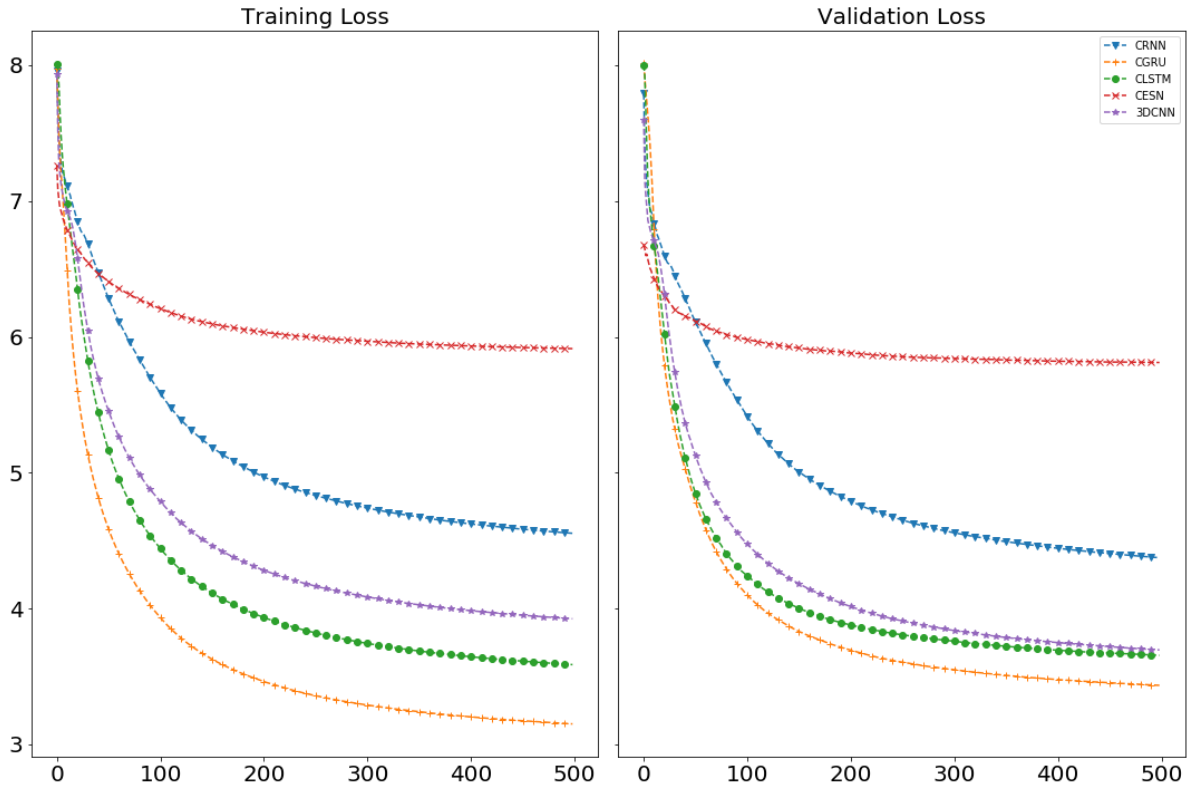

Figure 18: BSD validation loss curves for the CESN, CLSTM, CGRU, CRNN and 3DCNN.

As shown in Figure 18, both training and validation loss decreases with further training as in the WSD experiment mentioned in Section 4.2. However, the CLSTM and 3DCNN converge towards similar validation loss.

Figure 19 shows the precision, recall, F_1Score and IoU for the CESN, CRNN, CLSTM, CGRU and 3DCNN on the validation set.

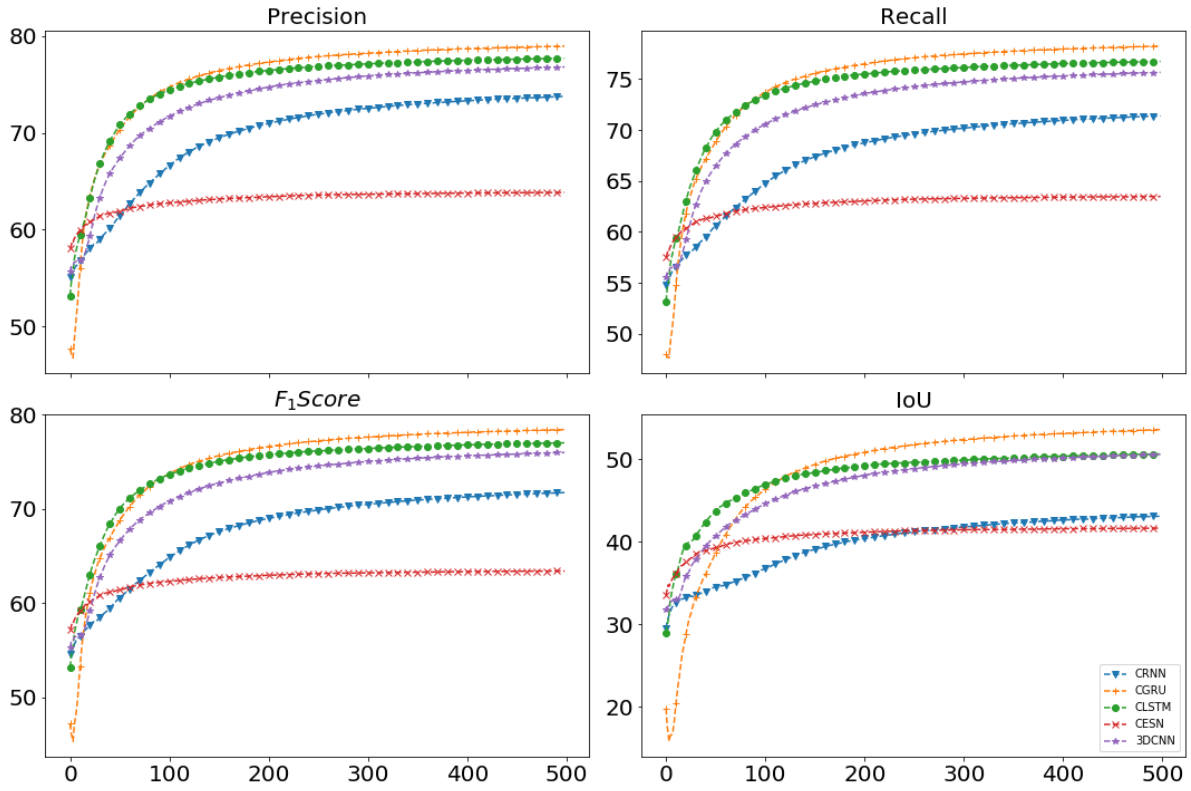

Figure 19: BSD validation performance curves for the CESN, CLSTM, CGRU, CRNN and 3DCNN.

In Figure 19, the CGRU still has highest IoU, F_1Score, recall, and precision, followed by the CLSTM, 3DCNN, CRNN, and CESN. The models are still sensitive to under-segmentation seen with a low recall. We see an improvement in performance for the CESN and CRNN, this can be due to the increased data size.

Table 5 shows the performance of the CESN, CRNN, CLSTM, CGRU, and 3DCNN on the testing set. The presence of white noise in the data is taken into consideration.

Table 5: BSD testing results. Here we compare various models and noise is also included.

Method	**IoU**	**Precision**	**Recall**	$F_1 Score$
CGRU	**56.7%**	**73.4%**	**76.1%**	**73.6%**
CLSTM	54.8%	71.1%	73.4%	71.4%
3DCNN	53.5%	70.7%	73.1%	71.0%
CRNN	49.7%	69.4%	71.8%	70.1%
CESN	37.5%	60.0%	61.1%	62.0%
White Noise	44.9%	50.0%	50.0%	43.4%

In Table 5, we see that the CESN and CRNN improved in performance compared to the WSD experiment, while the CRGU, CLSTM, and 3DCNN decreased in performance. And since the IoU of the CESN is less than the white noise, it's unable to learn the temporal states of the VLS. We also can see that the CGRU with IoU of **56.7%** outperforms, the CLSTM, 3DCNN, CRNN and CESN with IoU of **54.8%**, **53.7%**, **49.7%** and **37.5%**, respectively. The recall is still higher than precision, indicating that the models are prone to over-segmenting.

Figure 20 shows sample segmentations, the 1st, 2nd, 3rd, 4th, 5th, 6th and 7th columns are the input images, ground-truth (i.e. VLS), CESN, CRNN, CLSTM, CGRU and 3DCNN, respectively.

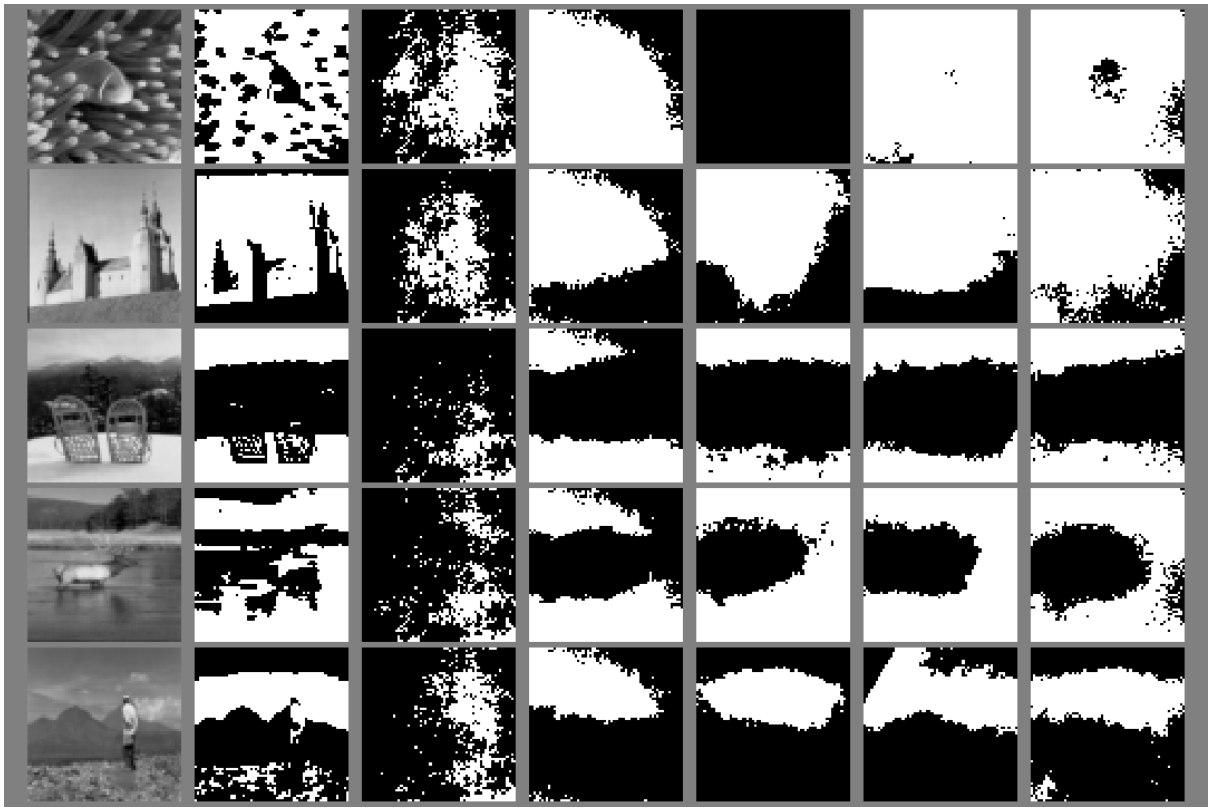

Figure 20: BSD examples of iterative segmentation. 1st column input, 2nd Chan-Vese, 3rd column CESN, 4th CRNN, 5th column CLSTM , 6th column CGRU , 7th column 3DCNN.

For overlapping objects like the image on the 5th row, the human and the ground are not correctly segmented. The CLSTM, 3DCNN, and CGRU can distinguish between the sky and the snow on the 3rd row. And for the other images, the model fails to localize and segment the images. The localization of objects in the images is an issue for the models and the segmentation as well. This means that images having more than one overlapping object are challenging to localize and segment.

4.3.3 Conclusion

We still observe similar characteristics to the WSD experiment. Considering that the BSD database has 300 more images than the WSD experiment and the added complexity of the overlapping objects. It is not surprising that the performance of the CGRU, CLSTM, and 3DCNN decreased since these models would require a large number of training examples for a challenging dataset.

The poor performance of the CESN is attributed to the optimal hyperparameters in presented Table 4 do not satisfy the echo state property. While the CRNN's poor performance is attributed to the exploding and vanishing gradients [56].

Under the conditions of this experiment, the CGRU is the most effective architecture for learning the evolution of the VLS. However, it does not outperform the state-of-the-art deep recurrent levelset (RLS) proposed in Le et al. 2018 [39], which achieved a $F_1 Score$ score of **99%** on the WSD for binary image segmentation.

4.4 CIFAR-10

4.4.1 Introduction

In this experiment, we used the CIFAR-10 database, this database is significantly larger than BSD and WSD.

4.4.2 Result and discussions

Table 6 shows the optimal hyper-parameters for each model selected based on maximizing the IoU on the validation set.

Table 6: CIFAR-10 dataset optimal parameters for the CESN, CRNN, CLSTM, CGRU and 3DCNN

Model	**Hidden size**	**Leaking rate**	**Sparsity**	**Spectral radius**	**Input scaling**
CESN	4096	0.713	0.2	0.9	1.0
CRNN	4096	-	-	-	-
CLSTM	4096		-	-	-
CGRU	4096	-	-	-	-
3DCNN	4096	-	-	-	-

In Table 6, we see that with an significant increase of data the optimum leaking rate, sparsity and input scaling of the CESN are **0.713**, **0.2**, **1**. Therefore, the leaking rate increase and sparsity decrease compared to the two previous experiments

The training and validation loss curves for the CESN, CRNN, CLSTM, CGRU, and 3DCNN are shown in Figure 21 as a function of the number of epochs.

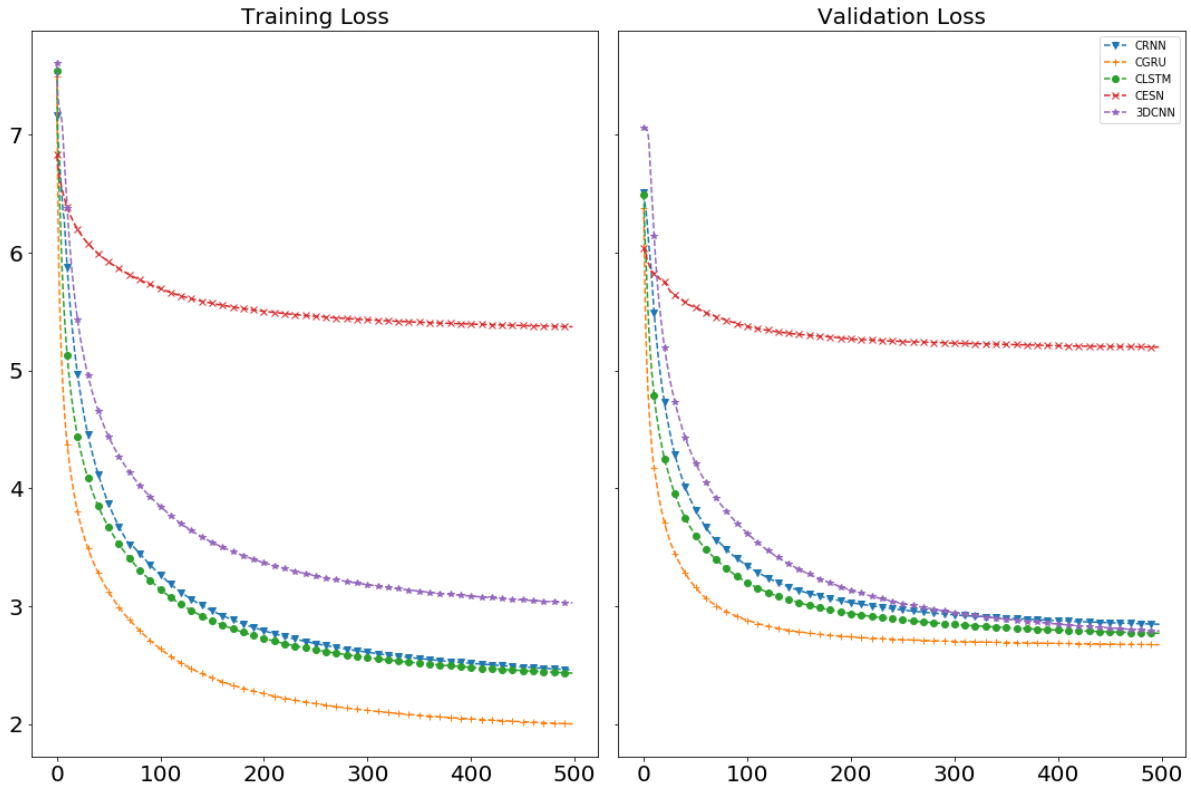

Figure 21: CIFAR-10 validation loss curves for the CESN, CLSTM, CGRU, CRNN and 3DCNN.

As shown in Figure 21, training on a larger dataset resulted in a decrease in the loss of all the models, more especially the CRNN compared to the WSD and BSD experiments. The CGRU has the lowest validation loss, followed by CLSTM, 3DCNN, CRNN, CESN. And all the models except the CESN converge towards the same loss, indicating that they are more likely to have comparable performance.

Figure 22 shows the precision, recall, F_1Score and IoU of the CESN, CRNN, CLSTM, CGRU and 3DCNN on the validation set.

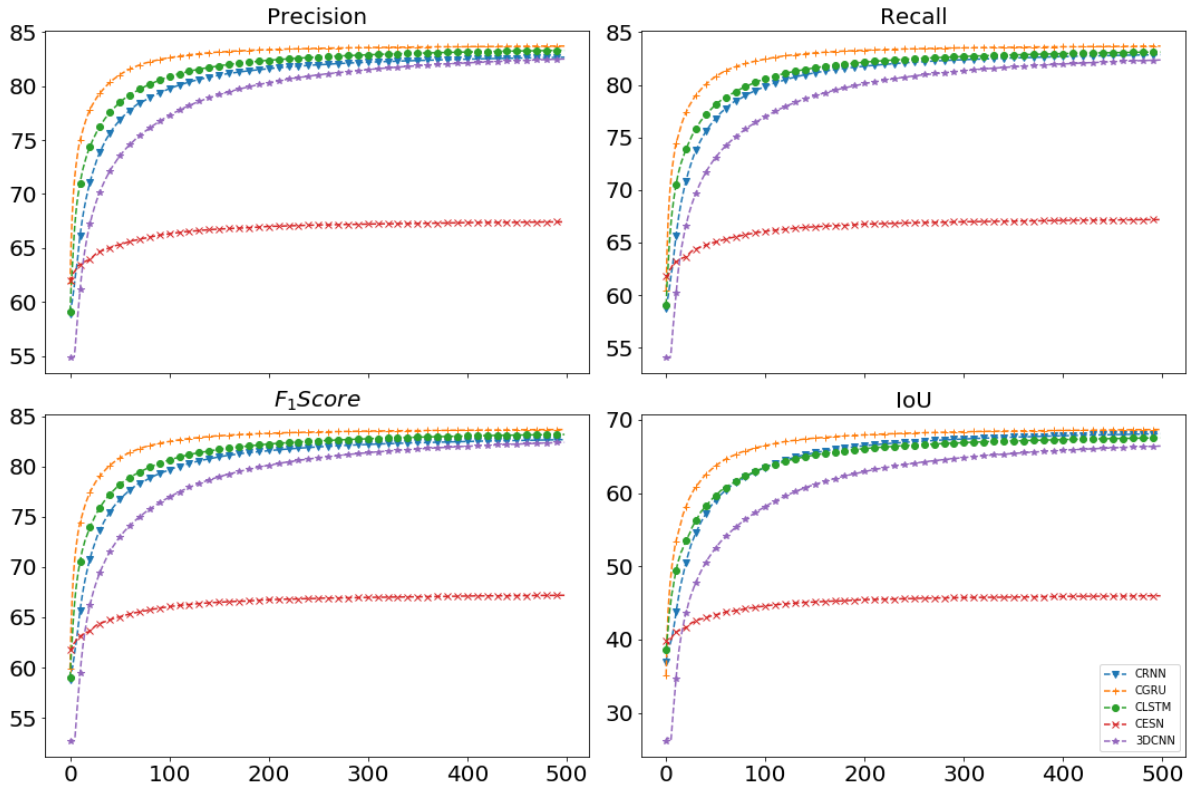

Figure 22: CIFAR-10 validation performance curves for the CESN, CLSTM, CGRU, CRNN and 3DCNN

The CLSTM has the highest F_1Score, IoU, recall, and precision, followed by CGRU, 3DCNN, CRNN, and the CESN. The precision and recall curves are equivalent to each other, this means we should observe a balance between under-segmentation and over-segmentation [65, 72]. The most interesting observation is that the CGRU, CLSTM, CRNN, and 3DCNN converge towards a single recall and precision as compared to the previous two experiments, this validates that these models require large amounts of data to achieve state-of-the-art results and better generalization capabilities to unseen observations. The CGRU, CLSTM, CRNN, and 3DCNN have IoU larger than the threshold of **50%**, which means we have a large

number of true positive predictions, however, the CESN IoU is still below the threshold.

Table 7 shows the performance of the CESN, CRNN, CLSTM, CGRU and 3DCNN on the testing set.

Table 7: CIFAR-10 testing results. Here we compare various models and noise is also included.

Method	**IoU**	**Precision**	**Recall**	$F_1 Score$
CLSTM	**68.8%**	**82.8%**	**83.8%**	**82.5%**
3DCNN	68.7%	82.5%	83.4%	82.2%
CGRU	68.3%	82.4%	83.0%	82.2%
CRNN	68.0%	81.9%	82.9%	81.9%
CESN	44.9%	65.1%	65.7%	66.4%
White Noise	30.3%	50.0%	50.0%	49.5%

In Table 7, with a IoU of **68.8%** the CLSTM outperforms, the 3DCNN, CGRU, CRNN and CESN with IoI of **68.7%**, **68.3%**, **68.0%** and **44.7%**, respectively. All the models can learn the temporal states of the VLS, this seen with IoU higher than white noise. The recall is higher than precision, indicating that the models are prone to over-segmenting.

Figure 23 shows sample segmentations, the 1st, 2nd, 3rd, 4th, 5th, 6th and 7th columns are the input images, ground-truth (i.e. VLS), CESN, CRNN, CLSTM, CGRU and 3DCNN, respectively.

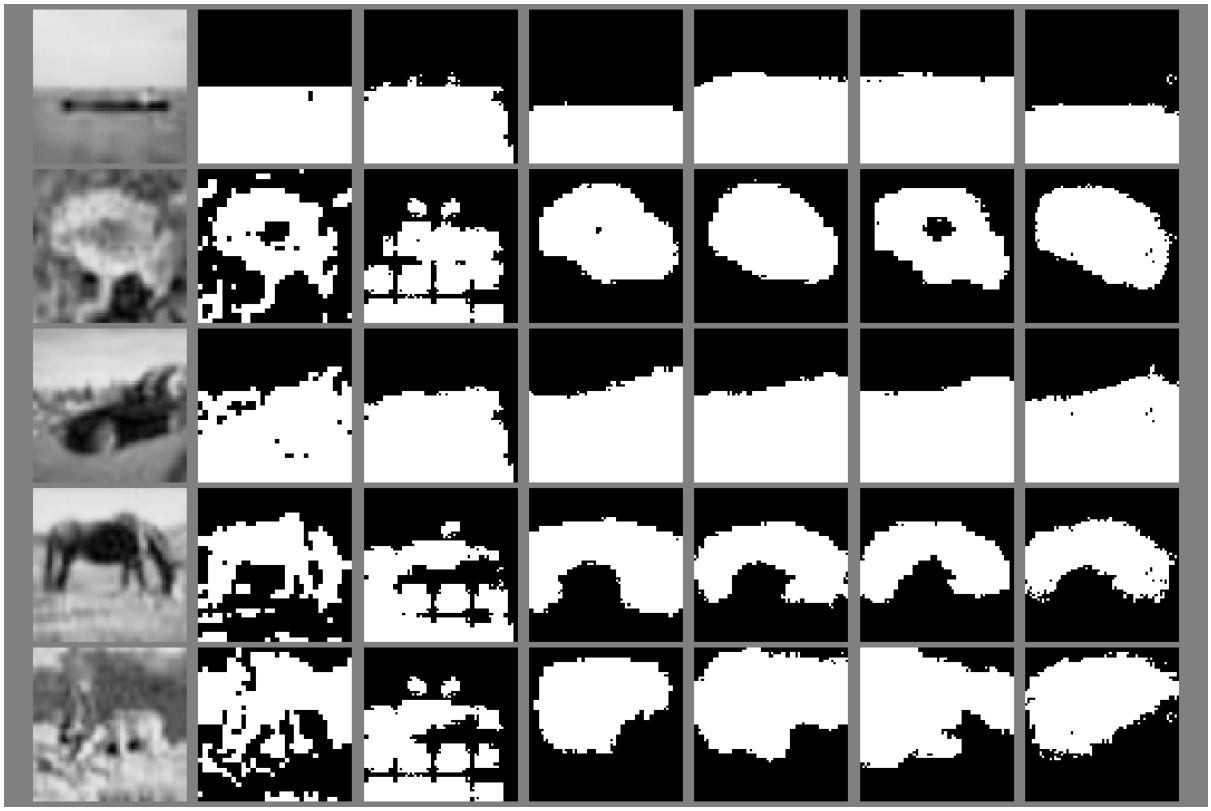

Figure 23: CIFAR-10 examples of iterative segmentation. 1st column input, 2nd Chan-Vese, 3rd column CESN, 4th CRNN, 5th column CLSTM , 6th column CGRU , 7th column 3DCNN.

Even though the CESN can localize the object in the images. The reservoir seems to be highly driven by the initial state of the VLS and struggles to segment the objects in the images. This is seen by the checkerbox structures in Figure 23 since the VLS is initialized as checkerbox, and these structures are common in all the samples in the data and the models are more likely to memorize these type of structures.

The CRNN, CLSTM, CGRU, and 3DCNN can localize and segment the objects in the images, but due to over-segmentation as seen from a higher recall than precision in Table 7, the segmentation boundaries are not well-defined and the fine details of the segmentation are lost. By up-scaling the

image from 32×32 to 64×64, the quality of the input images is reduced, this is seen by the poor contrast between inhomogeneous objects for all in the 1st column, especially the image in the 5th row, it is difficult to separate the bear from the surrounding. The ground-truth images are very noisy compared to the WSD and BSD databases, artifacts were introduced when up-scaling the images.

4.4.3 Conclusion

The data size is increased to 2000 images, thus, we have 2000 different sequences each with a length of 100 steps of the VLS segmentation, and including the augmented images. This a significant increase in the training data than the experiments conducted on the WSD and BSD databases. This validated our claim that CLSTM, CGRU, 3DCNN, and CRNN require large training examples to learn the VLS. The significant increase in the data resulted in an exponential improvement in the performance of the CRNN. The CESN still is unable to learn VLS.

We still observe over-segmentation because of the high false positive rate. This is because we have more background than foreground pixels.

Under the conditions of this experiment, CLSTM is the most effective architecture for learning VLS; however, it does not outperform the state-of-the-art deep recurrent levelsets (RLS) proposed in Le et al. 2018 [39], which achieved a $F_1 Score$ score of **99%** on the WSD for binary image

segmentation. These results can be improved with further training up 5000 epochs similar to Le et al. 2018 [39]

Although our proposed approach is benchmarked on the CIFAR-10 database not the WSD database in this experiment, it is still important to compare our proposed approach to RLS. As suggested in the previous experiments in Section 4.2 and 4.3.

4.5 CIFAR-100

4.5.1 Introduction

In this experiment we used the CIFAR-100 database, this database has ten times more classes than the CIFAR-10 and more data samples than the BSD and WSD.

4.5.2 Results and discussion

Table 8 shows the optimal hyper-parameters for each selected based on maximizing the IoU on the validation set.

Table 8: CIFAR-100 dataset optimal parameters for the CESN, CRNN, CLSTM, CGRU and 3DCNN

Model	Hidden size	Leaking rate	Sparsity	Spectral radius	Input scaling
CESN	4096	0.0713	0.4	0.9	1.0
CRNN	4096	-	-	-	-
CLSTM	4096		-	-	-
CGRU	4096	-	-	-	-
3DCNN	4096	-	-	-	-

Table 8, the optimum leaking rate and sparsity of the CESN are **0.0713**, **0.4**, respectively. The leaking rate decreased, while the sparsity of the reservoir increased compared to the CIFAR-10 experiment.

The training and validation loss curves for the CESN, CRNN, CLSTM, CGRU, and 3DCNN are shown in Figure 24 as a function of the number of epochs.

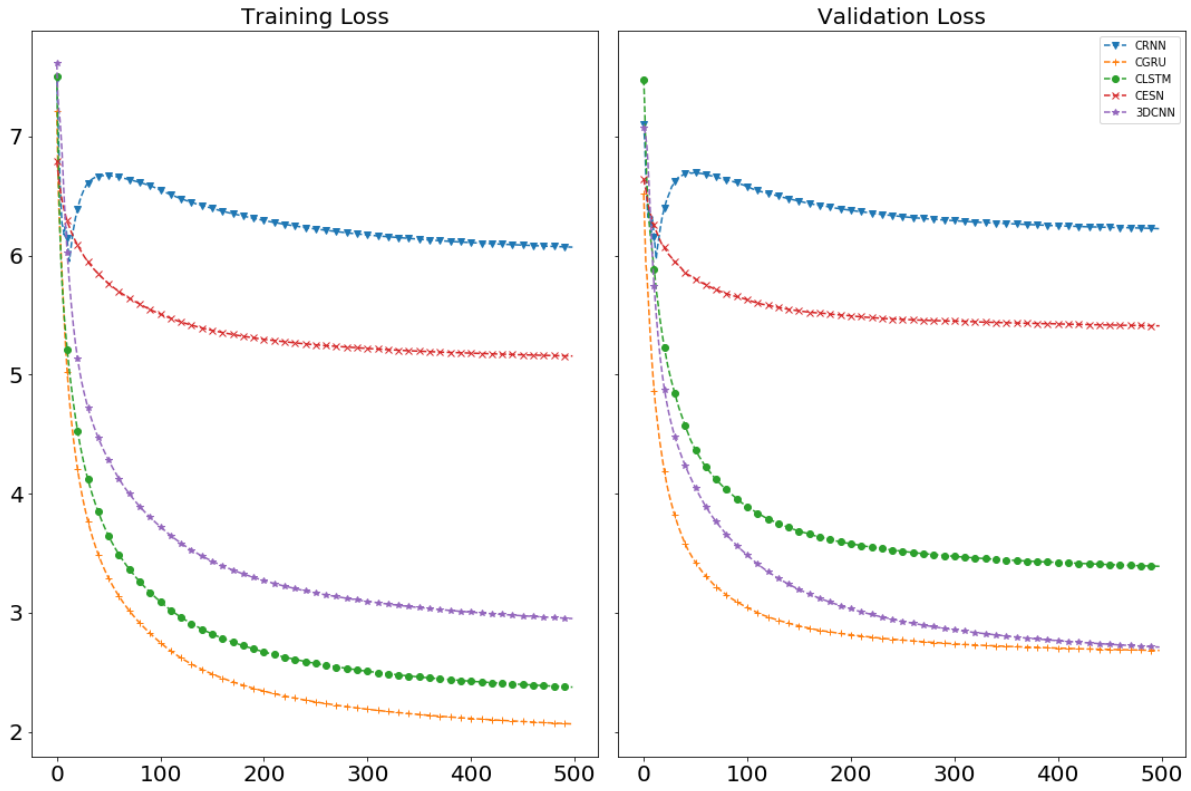

Figure 24: CIFAR-100 validation learning curves for the CESN, CLSTM, CGRU, RNN and 3DCNN

As shown in Figure 24, Both the validation and training losses decrease for all the models, which indicate their ability to learn the VLS segmentation process, except the CRNN. The CGRU has the lowest validation loss, followed by 3DCNN, CLSTM, CESN and interestingly the CRNN has the largest loss. The observed CRNN characteristics indicate unstable gradient information during back-propagation, and it is explained in Section 2.1. This problem is known as the vanishing and exploding gradient problem in RNNs [56], which results in sub-optimal performance as seen in Figure 25.

Figure 25 shows the precision, recall, F_1Score and IoU of the CESN, CRNN, CLSTM, CGRU and 3DCNN on the validation set.

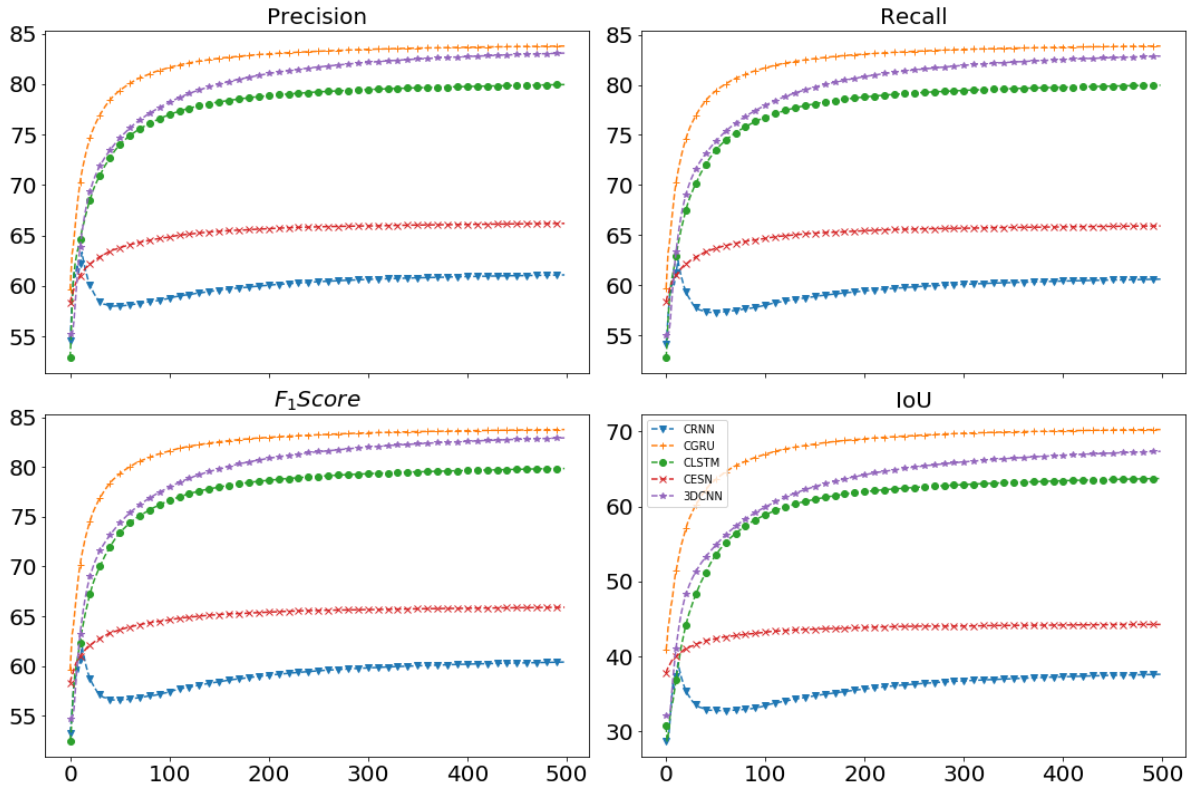

Figure 25: CIFAR-100 validation performance curves for the CESN, CLSTM, CGRU, RNN and 3DCNN

In Figure 25, the CGRU has the highest F_1Score, IoU, recall, and precision, followed by 3DCNN, CLSTM, CESN, and lastly the CRNN. The precision and recall curves are equivalent to each other, this means we should observe a balance between under-segmentation and over-segmentation [65, 72].

The CRNN performance decreases with further training, this phenomenon is due to an initial high learning rate, which causes the network to become unstable and results in vanishing and exploding gradients [56]. The decaying learning rate procedure mitigates this problem, as a result, the CRNN precision, recall, F_1Score and IoU increase with further training [84].

The CESN is performing better than CRNN, and this advantage is due to the sparsity that decouples the states of reservoir units, and since the readout layer is not trained using back-propagation, therefore, the CESN does not suffer from the vanishing and exploding gradients [30, 31, 43].

Table 9 shows the performance of the CESN, CRNN, CLSTM, CGRU, and 3DCNN on the testing set.

Table 9: CIFAR-100 testing results. Here we compare various models and noise is also included.

Method	**IoU**	**Precision**	**Recall**	$F_1 Score$
CGRU	**71.5**%	**84.2**%	**85**.0%	**83.9**%
3DCNN	69.4%	82.8%	84.0%	82.6%
CLSTM	65.3%	79.9%	81.8%	79.8%
CRNN	48.7%	68.0%	69.3%	69.7%
CESN	44.5%	64.8%	65.4%	66.0%
White Noise	30.2%	50.0%	50.0%	49.5%

As shown in Table 9, with a IoU of **71.5%** the CGRU outperforms the 3DCNN, CLSTM, CRNN and CESN with IoU of **69.4%**, **65.3%**, **48.7%** and **44.5%**, respectively. All the models can learn the temporal states of the VLS, this seen with IoU higher than white noise. The recall is still higher than precision, indicating that the models are prone to over-segmenting.

Figure 26 shows sample segmentations the 1st, 2nd, 3rd, 4th, 5th, 6th and 7th columns are the input images, ground-truth (i.e. VLS), CESN, CRNN, CLSTM, CGRU and 3DCNN, respectively.

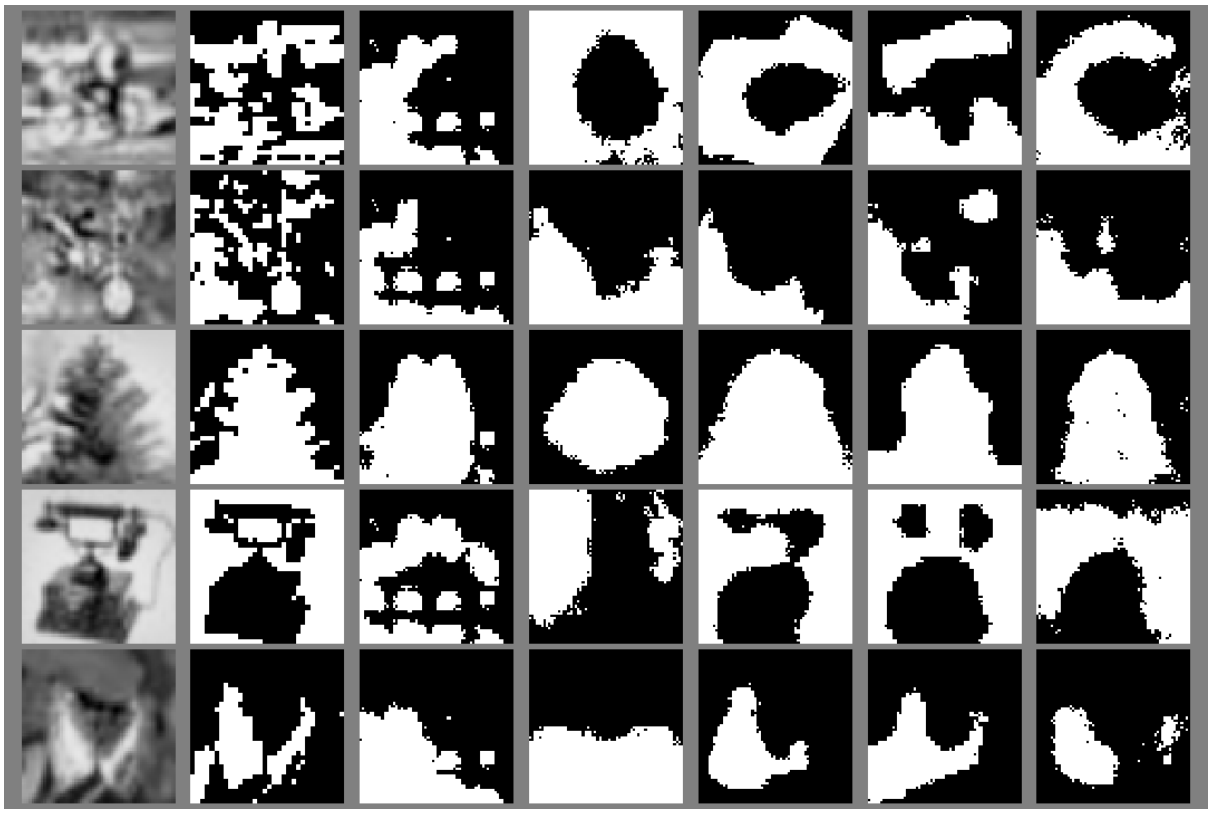

Figure 26: CIFAR-100 examples of iterative segmentation. 1st column input, 2nd Chan-Vese, 3rd column CESN, 4th CRNN, 5th column CLSTM , 6th column CGRU , 7th column 3DCNN.

Even though the CESN can localize the object in the images. The reservoir seems to be highly driven by the initial state of the VLS and struggles to segment the objects in the images. This is seen by the checkerbox structures in Figure 26 since the VLS is initialized as checkerbox, and these structures are common in all the samples in the data and the models are more likely to memorize these type of structures.

The CRNN, CLSTM, CGRU, and 3DCNN can localize and segment the objects in the images, but due to over-segmentation as seen from a higher recall than precision in Table 9, the segmentation boundaries are not well-defined and the fine details of the segmentation are lost. By up-scaling the image from 32×32 to 64×64, the quality of the input images is reduced, this is seen by the poor contrast between inhomogeneous objects, especially in the 1st, 2nd and 5th rows. The ground-truth images are very noisy compared to the WSD and BSD databases, this means artifacts were introduced when up-scaling the images. In the 2nd the CESN, CGRU, and CLSTM can segment the tree, although the details are lost and 3rd row the CGRU and CLSTM can segment out the telephone, even though the is originally incorrectly segmented by the VLS.

4.5.3 Conclusion

The CGRU outperforms the 3DCNN, CLSTM, and CRNN. The additional data samples improved the CGRU, 3DCNN, CLSTM, and CESN segmentation results when compared to the previous two experiments in Section 4.3 and 4.2. The CESN performed better than the CRNN on the validation than the testing set. The CESN segmentation results are better in comparison to all the previous experiments, although not optimal, this validates that the reservoir is task-specific and requires domain expertise to find optimal parameters [31, 43].

Under the conditions of this experiment, CGRU is the most effective architecture for learning VLS. However, it does not outperform the state-of-the-art deep recurrent levelsets (RLS) proposed in Le et al. 2018 [39], which achieved a $F_1 Score$ score of **99%** on the WSD for binary image segmentation. Although our proposed approach is benchmarked on the CIFAR-100 database not the WSD database in this experiment, it is still important to compare our proposed approach to RLS. As suggested in the previous experiments in Section 4.2 and 4.3.

4.6 CESN global parameter optimization

4.6.1 Introduction

In this section we aim to answer one of our research question mentioned in Section 1.2, namely:

- Is it possible to empirically determine which set of optimal parameters that satisfy the echo state property described in Section 2.5.2?

Investigating this question will enable us to understand the relationship between the configuration of the reservoir parameters and the performance. We look at how the main global parameters described in Section 2.4 and how they impact the performance of the CESN in learning the evolution of the VLS. When optimizing one of each parameter on the WSD database, we used a reservoir size of **4096**, a spectral radius of **0.9**, a sparsity of **0.8**, and a leaking rate of **0.01**. For each hyper-parameter optimization, we used 100 epochs with a decaying learning rate.

4.6.2 Results and discussion

Figure 27 shows the relationship between reservoir size, sparsity, leaking rate, and spectral radius, and how they impact the CESN performance in learning the LS.

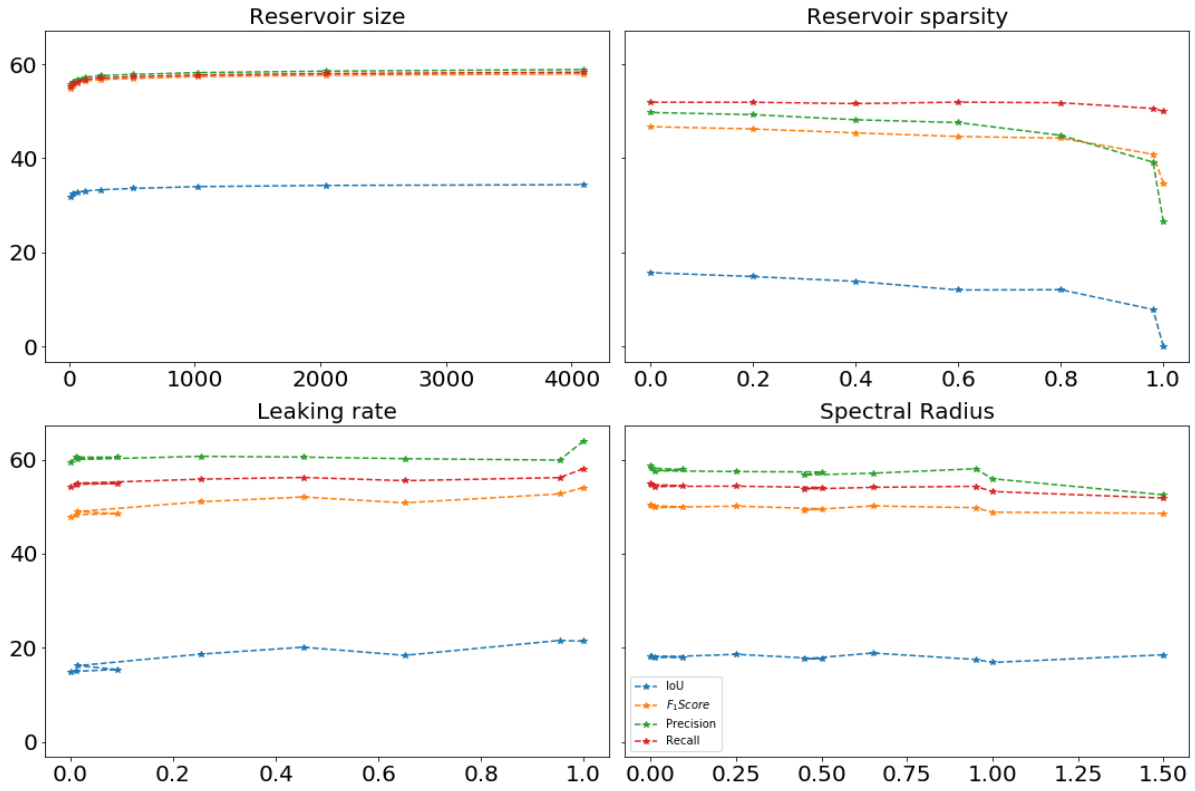

Figure 27: CESN hyper-parameter optimization

The reservoir size is proportional to the CESN's capacity, so more complicated problems necessitate larger reservoirs [30, 31]. According to Jaeger et al. 2000 [30], increasing the reservoir size improves performance (recall, accuracy, IoU, and $F1Score$). Due to the complexity of optimizing larger reservoirs, as reported by Jaeger et al. 2007 [33], the observed improvement in performance is insignificant.

The reservoir's sparsity is related to the reservoir's rich feature representation and short-term memory [22]. The efficiency of the reservoir decreases as the sparsity of the reservoir increases; this means that the more sparse the reservoir is, the less capacity it has and this has been shown by Lukosevivius et al. 2012 [42].

The rate at which the reservoir units react to the input is determined by the leak rate [30, 23]. Quick reservoirs with short-term memory result from high leak rates, whereas slow reservoirs with long-term memory result from low leak rates [23, 62]. We believe this trade-off makes solving problems that require both short-term and long-term memory challenging. As the leakage rate increases, performance improves; however, when the leaking rate increases above **0.4**, performance decreases slightly, this indicates a transition between the reservoir states updates by the input.

In practice, the spectral radius should be close to one for problems that need long-term memory and similarly for tasks where too much memory may be harmful [43]. As described by Jeager et al. 2001 [30, 31], a spectral radius greater than one results in a decrease in performance, as observed in Figure 27. Recent research by Yildiz et al. (2012) [83] shows that a reservoir matrix with a spectral radius less than one is insufficient to satisfy the echo state property described in Section 2.5.2.

4.6.3 Conclusion

As a result of the above observations, we respond to the following research question:

- Is it possible to empirically determine which set of optimal parameters satisfy the echo state property described in Section 2.5.2?

The spectral radius and leakage rate have a significant effect on the reservoir's performance and, as such, should be given further consideration

when configuring the ESN. However, it is unclear how the spectral radius influences reservoir's performance from literature, in this study we see that a spectral radius greater than one results in poor performance of the reservoir. While large reservoirs are typically used for difficult problems, optimizing a large reservoir to satisfy the echo state is difficult. The sparsity of the reservoir is related to the short-term memory of the reservoir.

Please keep in mind that the performance is used as a proxy for determining if the echo state property exists. This may not be a robust indicator to determine whether the echo state property exists, but we able are to identify the impact of the parameters of the reservoir on the ESN performance.

5 Conclusions, Contribution and Future Work

5.1 Overview

Training RNNs is computationally expensive. During training, the RNNs gradient information can explode or vanish, resulting in sub-optimal performance. ESNs were proposed as a computationally cheaper approach to training RNNs, however, finding a set of hyper-parameters of the ESN such that we have can achieve optimal results in a challenging and open problem to this day.

This research is twofold, 1) to investigate in detail the merits of using ESNs introduced in Chapter 2.2, as an alternative approach in training RNNs introduced in Chapter 2.1 in a computational inexpensive, and 2) proposal of a novel approach to contour evolution in VLS methods applied to binary image segmentation and how it compares to the current state-of-the-art in deep learning VLS mentioned in Section 2.6.4.

5.2 Research achievements

This study aims to compare the performance of ESNs and state-of-the-art RNNs in learning the variational level set. And to investigate how to formulate variational level set problems using deep learning frameworks as a data-driven approach. These research questions were explored to achieve the goal.

- If we consider the reservoir (i.e. recurrent layer) to have a similar updating mechanism as the variational levelset γ_t. How do we represent a single image $\mathbf{I}$ as spatiotemporal data X_t in ESNs?

Section 3.2 we presented the formulation of the variational levelset as a learnable data-driven approach via deep learning approach. Where the Chan-Vese implementation of the VLS is used to generate a sequence of segmented images X_t from a single image $\mathbf{I}$. Then the VLS is formulated as a spatiotemporal data-driven approach. As a result, the CESN is used to learn the spatiotemporal evolution of the VLS. The data is represented as a 2-channel input image $\mathbf{I}$, where the first channel is the input image, and the second channel s the VLS segmentation mask $\mathbf{M}_t$. This representation has proven to be robust since the model can capture the evolution of the VLS, although other input representations need to be investigated.

- Which of our proposed deep learning architectures is the most effective at learning the variational levelset? Is this deep learning architecture better than the current existing state-of-the-art deep learning variational levetset methods?

In Section 4.2, Section 4.3, Section 4.4 and Section 4.5, a comparative study was conducted on different databases with varying data samples, where we explored five architectures a CESN, CGRU, CLSTM, CRNN, and 3DCNN. The experimental results showed that using small databases such as WSD and BDS, the CGRU is the most effective deep learning architecture for learning the evolution of the VLS. And on large databases,

CIFAR-10 and CIFAR-100, the CLSTM, and CGRU are the most effective architecture for learning the evolution of the VLS. Furthermore, the results also showed that these models can localize the objects in the images, but fail to capture the boundaries of the objects.

We assume that long-term memory is required to preserve the initial state of the γ_t because the VLS is extremely reliant on the initialization of the velocities for propagating the levelset. The CGRU and CLSTM both have the advantage of being able to process long-term memory, making them ideal architectures for studying the evolution of the VLS. Both the CESN and the CRNN have short-term gradients, but unlike the CRNN, the CESN does not have vanishing and exploding gradients. However, our proposed approach does not outperform the state-of-the-art RLS proposed in Le et al. 2018 [39]. This may be as a result of the training being terminated too soon.

- Is it possible to empirically determine which a set of optimal parameters that satisfy the echo state property?

Finding a set of parameters that satisfy the echo state property can be difficult, particularly in complex architectures. The experimental results show that the leakage rate and spectral radius are the reason for the sub-optimal results. These parameters are connected to the reservoir's memory, and although they can be used to mitigate short-term memory in ESNs, tuning them also requires domain expertise. As a result, when using ESNs, the leakage rate and spectral radius should be given more consideration.

The sparsity is related to the amount of short-term memory in the reservoir. While the complexity of CLSTM, CGRU, and RNNs required a large amount of training to achieve optimal performance, the simplicity of CESN will limit its generalization capabilities, due to the large number of parameters that must be optimized to achieve optimal results.

In conclusion, since the ESN was unable to learn to solve the VLS iterative image segmentation, we reject the research hypothesis presented in Section 1.2.4. The second research hypothesis presented in Section 1.2.4 is accepted because the large number of hyper-parameters made it hard to ensure that the echo state property described in Section 2.5.2 is satisfied, resulting in the CESN failing to outperform the CLSTM, CGRU, CRNN, and 3DCNN.

5.3 Contributions and Future Work

By proposing a data-driven approach for learning VLS that has not yet been explored, this work contributes a novel approach to deep learning VLS and iterative segmentation methods. We are the first to suggest such an approach, to the best of our knowledge. The data-driven recurrent VLS has several limitations that will be addressed and investigated in future research:

- To use different threshold values for the predicted segmentation so that we can see if we obtain the same results for all threshold values. In our study, we only used a threshold of 0.5.

- The use of large, higher-resolution segmentation databases, such as PASCAL VOC 2012 and MS COCO 2014, will improve the results. In our current approach, there was a trade-off between the size and resolution of the image.
- To explore other alternatives of data representation of formulating VLS as a spatiotemporal problem.
- The current formulation of our deep recurrent VLS as a spatiotemporal problem is that the task is to predict the $n+1$ frame given the historical VLS segmentation (i.e. n previous frames). Since our proposed method is limited in practice. We aim to introduce a sliding window approach of length L in which the initial levelset $\mathbf{M}_0$ and input image $\mathbf{M}_0$ are given as input to the deep recurrent VLS during inference. Therefore the prediction from the n step is used as an input level set segmentation mask for the $n+1$ step.

Appendices

The code used to perform and generate the experiments and results presented in this work is available here:

https://github.com/LeparaLaMapara/ESNIterativeSegmentation

References

[1] Orhan Akal and Adrian Barbu. Learning chan-vese. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 1590–1594. IEEE, 2019.

[2] S Ali and B Dayyan. Segmentation model for noisy and intensity inhomogeneity images via logarithmic density function. *J Appl Computat Math*, 7(388):2, 2018.

[3] "Sharon Alpert, Meirav Galun, Ronen Basri, and Achi Brandt". "image segmentation by probabilistic bottom-up aggregation and cue integration.". In *"Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition"*, "June" "2007".

[4] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, May 2011.

[5] Amir F Atiya and Alexander G Parlos. New results on recurrent network training: unifying the algorithms and accelerating convergence. *IEEE transactions on neural networks*, 11(3):697–709, 2000.

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

[7] Akanksha Bali and Shailendra Narayan Singh. A review on the strategies and techniques of image segmentation. In *Advanced Computing &*

Communication Technologies (ACCT), 2015 Fifth International Conference on, pages 113–120. IEEE, 2015.

[8] Vicent Caselles, Francine Catté, Tomeu Coll, and Françoise Dibos. A geometric model for active contours in image processing. *Numerische mathematik*, 66(1):1–31, 1993.

[9] Vicent Caselles, Ron Kimmel, and Guillermo Sapiro. Geodesic active contours. *International journal of computer vision*, 22(1):61–79, 1997.

[10] Turgay Celik and Kai-Kuang Ma. Multitemporal image change detection using undecimated discrete wavelet transform and active contours. *IEEE Transactions on Geoscience and Remote Sensing*, 49(2):706–716, 2010.

[11] Tony F Chan, B Yezrielev Sandberg, and Luminita A Vese. Active contours without edges for vector-valued images. *Journal of Visual Communication and Image Representation*, 11(2):130–141, 2000.

[12] Tony F Chan and Luminita A Vese. Active contours without edges. *IEEE Transactions on image processing*, 10(2):266–277, 2001.

[13] Hanten Chang and Katsuya Futagami. Reinforcement learning with convolutional reservoir computing. *Applied Intelligence*, 50(8):2400–2410, 2020.

[14] Bo Chen and Wen-Sheng Chen. Noisy image segmentation based on wavelet transform and active contour model. *Applicable Analysis*, 90(8):1243–1255, 2011.

[15] Bo Chen, Qing-Hua Zou, Wen-Sheng Chen, and Yan Li. A fast region-based segmentation model with gaussian kernel of fractional order. *Advances in Mathematical Physics*, 2013, 2013.

[16] Bo Chen, Qing-Hua Zou, Wen-Sheng Chen, and Bin-Bin Pan. A novel adaptive partial differential equation model for image segmentation. *Applicable Analysis*, 93(11):2440–2450, 2014.

[17] Hongyan Cui, Xiang Liu, and Lixiang Li. The architecture of dynamic reservoir in the echo state network. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 22(3):033127, 2012.

[18] Kenji Doya. Bifurcations in the learning of recurrent neural networks. In *Circuits and Systems, 1992. ISCAS'92. Proceedings., 1992 IEEE International Symposium on*, volume 6, pages 2777–2780. IEEE, 1992.

[19] Chao Du, Fuxi Cai, Mohammed A Zidan, Wen Ma, Seung Hwan Lee, and Wei D Lu. Reservoir computing using dynamic memristors for temporal information processing. *Nature communications*, 8(1):2204, 2017.

[20] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[21] Ken-ichi Funahashi and Yuichi Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural networks*, 6(6):801–806, 1993.

[22] Claudio Gallicchio. Sparsity in reservoir computing neural networks. In *2020 International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*, pages 1–7. IEEE, 2020.

[23] Claudio Gallicchio, Alessio Micheli, and Luca Pedrelli. Deep reservoir computing: A critical experimental analysis. *Neurocomputing*, 268:87–99, 2017.

[24] Lluis Garrido, Marité Guerrieri, and Laura Igual. Image segmentation with cage active contours. *IEEE Transactions on Image Processing*, 24(12):5557–5566, 2015.

[25] Alireza Goudarzi, Sarah Marzen, Peter Banda, Guy Feldman, Christof Teuscher, and Darko Stefanovic. Memory and information processing in recurrent neural networks. *CoRR*, abs/1604.06929, 2016.

[26] Alex Graves. Supervised sequence labelling. In *Supervised sequence labelling with recurrent neural networks*, pages 5–13. Springer, 2012.

[27] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[28] Namdar Homayounfar, Yuwen Xiong, Justin Liang, Wei-Chiu Ma, and Raquel Urtasun. Levelset r-cnn: A deep variational method for instance segmentation. In *European Conference on Computer Vision*, pages 555–571. Springer, 2020.

[29] Ping Hu, Bing Shuai, Jun Liu, and Gang Wang. Deep level sets for salient object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2300–2309, 2017.

[30] Herbert Jaeger. The echo state approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13, 2001.

[31] Herbert Jaeger. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*, volume 5. GMD-Forschungszentrum Informationstechnik Bonn, 2002.

[32] Herbert Jaeger. Adaptive nonlinear system identification with echo state networks. In *Advances in neural information processing systems*, pages 609–616, 2003.

[33] Herbert Jaeger, Mantas Lukoševičius, Dan Popovici, and Udo Siewert. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural networks*, 20(3):335–352, 2007.

[34] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2012.

[35] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[36] Muhammad Waseem Khan. A survey: Image segmentation techniques. *International Journal of Future Computer and Communication*, 3(2):89, 2014.

[37] PA Kountouriotis, D Obradovic, Su Lee Goh, and Danilo P Mandic. Multi-step forecasting using echo state networks. In *EUROCON 2005-The International Conference on" Computer as a Tool"*, volume 2, pages 1574–1577. IEEE, 2005.

[38] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[39] Ngan Thi Hoang Le. *Contextual Recurrent Level Set Networks and Recurrent Residual Networks for Semantic Labeling.* PhD thesis, Carnegie Mellon University, 2018.

[40] T Hoang Ngan Le, Kha Gia Quach, Khoa Luu, Chi Nhan Duong, and Marios Savvides. Reformulating level sets as deep recurrent neural network approach to semantic segmentation. *IEEE Transactions on Image Processing*, 27(5):2393–2407, 2018.

[41] Thi Hoang Ngan Le, Khoa Luu, Marios Savvides, Kha Gia Quach, and Chi Nhan Duong. Recurrent level set networks for instance segmentation. In *Pattern Recognition-Selected Methods and Applications.* IntechOpen, 2019.

[42] Mantas Lukoševičius. A practical guide to applying echo state networks. In *Neural networks: Tricks of the trade*, pages 659–686.

Springer, 2012.

[43] Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

[44] Sheng Ma and Chuanyi Ji. Fast training of recurrent networks based on the em algorithm. *IEEE Transactions on Neural Networks*, 9(1):11–26, 1998.

[45] Wolfgang Maass, Thomas Natschläger, and Henry Markram. A model for real-time computation in generic neural microcircuits. In *Proceedings of the 15th International Conference on Neural Information Processing Systems*, NIPS'02, pages 229–236, Cambridge, MA, USA, 2002. MIT Press.

[46] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.

[47] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Computational models for generic cortical microcircuits. *Computational neuroscience: A comprehensive approach*, 18:575, 2004.

[48] David Royal Martin, J Malik, and D Patterson. *An empirical approach to grouping and segmentaqtion*. Computer Science Division, University of California Berkeley, 2003.

[49] MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010.

[50] Luca Melandri. *Introduction to Reservoir Computing Methods*. PhD thesis, Universita di Bologna, 2013.

[51] Thomas Natschläger, Henry Markram, and Wolfgang Maass. Computer models and analysis tools for neural microcircuits. In *Neuroscience Databases*, pages 123–138. Springer, 2003.

[52] Sergiu Oprea, Pablo Martinez-Gonzalez, Alberto Garcia-Garcia, John Alejandro Castro-Vargas, Sergio Orts-Escolano, Jose Garcia-Rodriguez, and Antonis Argyros. A review on deep learning techniques for video prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[53] Andrey Ovsyannikov. *Further development of Level Set method: modified level set equation and its numerical assessment*. PhD thesis, Ecole Centrale de Lyon, 2013.

[54] Nikos Paragios, Olivier Mellina-Gottardo, and Visvanathan Ramesh. Gradient vector flow fast geodesic active contours. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 67–73. IEEE, 2001.

[55] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012.

[56] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

[57] Gintaras V Puskorius and Lee A Feldkamp. Neurocontrol of nonlinear dynamical systems with kalman filter trained recurrent networks. *IEEE Transactions on neural networks*, 5(2):279–297, 1994.

[58] Md Atiqur Rahman and Yang Wang. Optimizing intersection-over-union in deep neural networks for image segmentation. In *International symposium on visual computing*, pages 234–244. Springer, 2016.

[59] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. In David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group, editors, *Parallel Distributed Processing*, chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[60] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[61] Nils Schaetti. Reservoir Computing : Etude theorique et pratique en reconnaissance de chiffres manuscrits. Master's thesis, Universite de Franche-Comte, 09 2015.

[62] Nils Schaetti, Michel Salomon, and Raphaël Couturier. Echo state networks-based reservoir computing for mnist handwritten digits

recognition. In *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*, pages 484–491. IEEE, 2016.

[63] Jürgen Schmidhuber. A fixed size storage o (n 3) time complexity learning algorithm for fully recurrent continually running networks. *Neural Computation*, 4(2):243–248, 1992.

[64] Benjamin Schrauwen, Lars Büsing, and Robert A Legenstein. On computational power and the order-chaos phase transition in reservoir computing. In *Advances in Neural Information Processing Systems*, pages 1425–1432, 2009.

[65] Jose Sigut, Francisco Fumero, and Omar Nuñez. Over-and under-segmentation evaluation based on the segmentation covering measure. *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2015.

[66] Guo-Zheng Sun, Hsing-Hen Chen, and Yee-Chun Lee. Greenś function method for fast on-line learning algorithm of recurrent neural networks. In *Advances in neural information processing systems*, pages 333–340, 1992.

[67] Xue-Cheng Tai and Chang-hui Yao. Image segmentation by piecewise constant mumford-shah model without estimating the constants. *Journal of Computational Mathematics*, pages 435–443, 2006.

[68] Min Tang, Sepehr Valipour, Zichen Zhang, Dana Cobzas, and Martin Jagersand. A deep level set method for image segmentation. In *Deep learning in medical image analysis and multimodal learning for clinical decision support*, pages 126–134. Springer, 2017.

[69] N Toomarian and Jacob Barhen. Adjoint-functions and temporal learning algorithms in neural networks. In *Advances in neural information processing systems*, pages 113–120, 1991.

[70] Nikzad Benny Toomarian and Jacob Barhen. Learning a trajectory using adjoint functions and teacher forcing. *Neural Networks*, 5(3):473–484, 1992.

[71] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

[72] Andrés Troya-Galvis, Pierre Gançarski, Nicolas Passat, and Laure Berti-Equille. Unsupervised quantification of under-and over-segmentation for object-based remote sensing image analysis. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 8(5):1936–1945, 2015.

[73] Phan Tran Ho Truc, Tae-Seong Kim, Sungyoung Lee, and Young-Koo Lee. Homogeneity-and density distance-driven active contours for medical image segmentation. *Computers in biology and medicine*, 41(5):292–301, 2011.

[74] Pooja R Upadhyay and Mr Shashwat Kumar. Survey on various image segmentation techniques. *Journal of Emerging Technologies and Innovative Research (JETIR)*, 2017.

[75] Luminita A Vese and Tony F Chan. A multiphase level set framework for image segmentation using the mumford and shah model. *International journal of computer vision*, 50(3):271–293, 2002.

[76] Jiang Wang, Yi Yang, Junhua Mao, Zhiheng Huang, Chang Huang, and Wei Xu. Cnn-rnn: A unified framework for multi-label image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2285–2294, 2016.

[77] Zian Wang, David Acuna, Huan Ling, Amlan Kar, and Sanja Fidler. Object instance annotation with deep extreme level set evolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7500–7508, 2019.

[78] Paul J Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, 1(4):339–356, 1988.

[79] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[80] Ronald J Williams. Complexity of exact gradient computation algorithms for recurrent neural networks. Technical report, Technical Report Technical Report NU-CCS-89-27, Boston: Northeastern University, College of Computer Science, 1989.

[81] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Comput.*, 1(2):270–280, June 1989.

[82] Ronald J Williams and David Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. *Backpropagation: Theory, architectures, and applications*, 1:433–486, 1995.

[83] Izzet B Yildiz, Herbert Jaeger, and Stefan J Kiebel. Re-visiting the echo state property. *Neural networks*, 35:1–9, 2012.

[84] Kaichao You, Mingsheng Long, Jianmin Wang, and Michael I Jordan. How does learning rate decay help modern neural networks? *arXiv preprint arXiv:1908.01878*, 2019.

[85] Kaihua Zhang, Lei Zhang, Huihui Song, and Wengang Zhou. Active contours with selective local or global segmentation: a new formulation and level set method. *Image and Vision computing*, 28(4):668–676, 2010.

[86] Yaoyue Zheng, Zhang Chen, Xiaojian Li, Xiangyu Si, Liangjie Dong, and Zhiqiang Tian. Deep level set with confidence map and boundary loss for medical image segmentation. In *2020 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2020.