TIME-SERIES CLASSIFICATION WITH KERNELCANVAS AND WISARD

Diego Fonseca Pereira de Souza

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadores: Felipe Maia Galvão França
Priscila Machado Vieira Lima

Rio de Janeiro
Dezembro de 2015

TIME-SERIES CLASSIFICATION WITH KERNELCANVAS AND WISARD

Diego Fonseca Pereira de Souza

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Edmundo Albuquerque de Souza e Silva, Ph.D.

Prof. Priscila Machado Vieira Lima, Ph.D.

Prof. Alberto Ferreira de Souza, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
DEZEMBRO DE 2015

Souza, Diego Fonseca Pereira de

Time-Series Classification with KernelCanvas and WiSARD/Diego Fonseca Pereira de Souza. – Rio de Janeiro: UFRJ/COPPE, 2015.

XIII, 66 p.: il.; 29, 7cm.

Orientadores: Felipe Maia Galvão França

Priscila Machado Vieira Lima

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2015.

Referências Bibliográficas: p. 59 – 62.

1. weightless neural networks. 2. time-series. 3. signal processing. 4. pattern classification. I. França, Felipe Maia Galvão *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

To my parents and my sister.

Agradecimentos

Tenho absoluta certeza que não teria chegado até aqui se não fosse pela compreensão e ensinamentos destes e portanto agradeço primeiramente a minha família por tudo que me deram até hoje. Em especial, agradeço a minha mãe, Elenice, por todo o amor e zelo que dedica por mim e por sempre ter me ensinado o quão importante é o ato de estudar. Ao meu pai, Arnaldo, por sempre me incentivar, muitas vezes se interessar pelas coisas que eu fazia e pela curiosidade com os computadores que acabou me levando a esta área. A minha irmã, Aline, por todo carinho, implicâncias, e por sempre cuidar e se importar muito comigo. Ao meu cunhado, Fernando, pelos incentivos, viagens e amizade. Agradeço a todos estes por terem tido muita paciência comigo, principalmente nos últimos dias.

Ao meu orientador Felipe pela ideia original de pesquisa e por todo o incentivo, conversas, amizade e ajuda para terminar os trabalhos e ir aos congressos. A minha orientadora Priscila, por todos os motivos anteriores e adicionalmente por ter me apresentado ao meu primeiro tema de trabalho, o qual me fez ganhar interece na área de redes neurais sem peso. Além de terem sido ótimos amigos e estarem sempre presentes, eu também gostaria de agradecer pelas cobranças, as vezes necessárias para conseguirmos concluir os prazos.

Aos ótimos amigos que trago desde a graduação e a todos aqueles que conheci nesses anos do mestrado, sejam eles do PESC ou de outras atividades. A todos estes eu agradeço pois sempre estiveram por perto quando eu precisava conversar, pelos muitos incentivos para continuar, e pelas vezes com que saimos (ou tentavamos). Um agradecimento especial a todos do LabIA, mesmo os que agora estão em lugares distantes, por todos os momentos em que trabalhamos juntos mas também pelos bons momentos de descontração.

Espero verdadeiramente que todas essas amizades perdurem por muitos anos...

Acknowledgments

I am pretty sure that I would not be accomplishing this were it not for the understanding and teachings of my family. Therefore, I primarily thank them for everything they gave me up to now. In special, I thank my mother, Elenice, for all the love and care dedicated to me and for teaching me how important the act of studying is. To my father, Arnaldo, for always encouraging me, finding interest on the thinks I did and for the curiosity with computers which in the end attracted me to this field. To my sister, Aline, for all the fondness, teasing, and for always caring about me. To my brother in law, Fernando, for the incentives, tours and friendship. I thank all these people for having a lot of patience with me, especially in recent days.

I also thank my advisor Felipe França for the main research idea and for all incentives, talks, friendship and the help when polishing the works and going to conferences. To my advisor Priscila Lima for all previous reasons and additionally for presenting me to my first topic of work, which made me gain interest on the weightless neural network subject. In addition to being great friends and attentive advisors, I would also like to thank them for the few demands, sometimes necessary for us to meet the deadlines.

To the great friends I bring since my graduation and to all others I met these years, whether from PESC or from other activities. I thank all these people for always being around when I needed to talk, for the many incentives to continue, and for all the times we went out (or tried). A special thank to all those from LabIA, even those now far away, for all the moments working together but also for the good moments of relaxation.

I truly hope that all these friendships will endure for many years...

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

CLASSIFICAÇÃO DE SÉRIES TEMPORAIS UTILIZANDO KERNELCANVAS E WISARD

Diego Fonseca Pereira de Souza

Dezembro/2015

Orientadores: Felipe Maia Galvão França
Priscila Machado Vieira Lima

Programa: Engenharia de Sistemas e Computação

Dados envolvendo séries temporais estão largamente presentes em nosso cotidiano, seja na forma de ondas de áudio, em sensores de instrumentos médicos, no movimento de pessoas e animais, ou em inúmeras outras. Considerando as diversas aplicações que podem ser desenvolvidas por meio desses, a classificação de séries temporais busca discriminar os padrões observados a partir de generalizações feitas sobre um conjunto finito de exemplos de treinamento. Este conjunto pode ter sido apresentado inteiramente durante uma fase prévia de desenvolvimento, ou obtido iterativamente no ambiente de execução, incrementando o conhecimento já existente. Inúmeras abordagens estão disponíveis na literatura para realizar esta tarefa e muitas vezes conseguem alcançar satisfatórias taxas de reconhecimento. Infelizmente, poucas conseguem se adequar a ambientes dinâmicos, onde novas amostras e classes são apresentadas a todo momento, e ainda conseguem manter características importantes, como operações de treinamento e classificação em tempo real.

Neste trabalho é apresentada uma nova forma de classificação de séries temporais. A metodologia, baseada na rede neural sem peso WiSARD, alcança todos esses objetivos por meio de uma abordagem que identifica aleatoriamente as principais características dos sinais apresentados, ao mesmo tempo que não prejudica as características do modelo neural utilizado. Para avaliar a eficácia do modelo proposto, cinco outros modelos e cinco bases de dados envolvendo três tipos diferentes de dados foram utilizados. Ao final dos experimentos, os resultados do modelo proposto são também comparados com os melhores resultados encontrados na literatura, na qual o modelo proposto, além de apresentar todas as características desejadas, também apresenta muitos resultados equiparáveis.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

TIME-SERIES CLASSIFICATION WITH KERNELCANVAS AND WISARD

Diego Fonseca Pereira de Souza

December/2015

Advisors: Felipe Maia Galvão França
Priscila Machado Vieira Lima

Department: Systems Engineering and Computer Science

Time series data is widely present in our world, whether in the form of audio waves, in sensors of medical instruments, on the movement of people and animals, or on numerous others. Among the diversity of applications that can be developed from them, the classification of time series seeks to discriminate observed patterns through generalizations made over a finite set of training examples. This set may have been fully presented at a preliminary stage of development, or obtained iteratively in the execution environment, increasing the existing knowledge. Countless approaches are available in the literature for this task and they often achieve satisfactory recognition rates. Unfortunately, few of them are able to adapt to dynamic environments, where new samples and classes are continuously presented, while still maintaining important features such as training and classification operations in real time.

This work presents a new form of time series classification. The methodology, based on the weightless neural network WiSARD, achieves all these goals through an approach that randomly identifies key features of the signals presented without hurting the characteristics of the neural model used. To evaluate the effectiveness of the proposed model, five other models and five data sets containing data from three different sources were used. At the end of the experiments, the results of the proposed model are also compared with the best results found in the literature, were it is observed that the proposed model, in addition to contain all the desired features, also presents many equivalent results.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Time series has attracted interest of many scientists over the last decades. Thanks to their advances, a lot has been achieved in this field and many challenging tasks are now much more tractable. Despite this, many improvements are still feasible, specially if characteristics from new and emerging areas such as weightless neural networks are considered. This work wants to contribute in this continuous development and proposes a new approach to deal with one of its subareas, the time series classification.

As the name suggests, a time serie is any type of data relating temporal or sequential data. Usually, these sequences are obtained by the measurement of a fixed number of variables over a period of time or space. If the time serie is composed by a single variable it is said to be univariate. If more than one variable is captured, however, then it is said that it is multivariate.

Time series may be obtained or created in many different ways. Some of them may be extracted from the stock market, handwritten characters, protein sequences, medical instruments such as an electroencephalogram or an electrocardiogram, and many others. Even raw images may be converted to time series data, e.g., by detecting the border of an object and calculating the distance from the edges to its center.

Depending on the desired application, different techniques may be used to extract different types of information from them. A few of these applications include anomaly detection, prediction of future values, classification of sequences, indexing, among others. In this work, however, we are particularly interested in time series classification. Time series classification is the classification of sequences given a set of classes and their corresponding samples. For instance, classification of an object's shape, drawn handwritten characters or the word represented on an audio sample. Although the main objective of this work is to create a general model that could be later applied to any type of time series classification, this work focus mostly on three types of data. These are video gestures, handwritten characters and audio recognition.

1.1 Motivation

With the recent advances of computational power on mobile devices in this decade, a lot of new applications involving time series computations have emerged and now take part of people's life. Some of these applications include practical voice to text conversions, song identification, lyrics tracking, intelligent gesture keyboards on touch screen devices, speaker identification, automatic musician evaluation based on music score cards, gestures recognition, among many others.

Time series interpretation, however, is not an easy task. Differently from other types of data, time series are composed by a discrete sequence of feature values collected over a certain amount of time. Depending on the application, a long time series may need to be split in multiple subsequences in a process called segmentation. A segmentation process may be used, e.g., to represent words on a word recognition application, or to represent musical notes on a music application, among others. Naturally, depending on the speed and quality of the capture device and the emitter of these signals, each sequence may contain distinct lengths and that way not linearly match each other. For this reason, it is said that to adequately compare two sequences an elastic matching needs to be performed.

Despite these difficulties, literature has a vast range of models that correctly address these issues. To name a few, among the most popular are the K-Nearest Neighbors (KNN) [2] with Dynamic Time Warping (DTW) [3], Hidden Markov Models (HMM) [4], Deep Learning [5], among others. Many of the models used, however, present many disadvantages when used in practice, like excessive time consumed during the training or classification phase, difficulties to increment the model's knowledge given new classes or samples, extremely complex architectures, among others. As a result of this, this research field still has many improvement opportunities and challenging characteristics, creating the following motivations for this work:

1. Time series is a large area of research with many practical applications. In particular, the field of artificial intelligence has demonstrated that much can be can be achieved with its processing. Seeking these applications and opportunities is one of the highest motivational effects for entering into this field;

2. With the recent advances in the processing capacity of mobile devices, its becoming increasingly easier to process this type of information on them. This creates a new range of applications and possibilities as they are also much more conveniently carried by users. Additionally, they are also entering unlikely

areas for desktop computers or notebooks such as trips, camping, walking on the street, and more recently even under the water;

3. Finally, it also provides the opportunity to study different approaches and characteristics of artificial intelligence, while its wide range of applications also creates many research opportunities. These can be explored in order to provide more robust solutions for time series classification, particularly with the use of emerging technologies such as weightless neural networks.

1.2 Contributions

In this work, a novel methodology for time series classification based on the weightless neural network named WiSARD is presented. The proposed methodology, named KernelCanvas, was tested on three types of time series data and does not contain many of the issues described on previous section. Alternative approaches, like the K-Nearest Neighbors (KNN) with Dynamic Time Warping (DTW), the Echo State Network (ESN), and basic resampling techniques are also applied under similar conditions for comparison.

Furthermore, three main characteristics were required for the proposed model. First, states that it should be able to achieve competitive results when compared to the models selected. Second, as the model should be used in practical applications in the future, it is important that it should still qualify for real time performance. Finally, some characteristics of the WiSARD model, like incremental, learning should also be preserved.

In addition to this, the use of the WiSARD network in conjunction with other basic approaches and as a replacement classifier on the output of the ESN was also investigated. The investigation was performed in order to compare the use of simpler techniques against the proposed methodology but also as a possible improvement of the other approaches.

Seeking a fair comparative evaluation against other models, two main experiments were made. The first was a direct comparison under similar preprocessing conditions against the selected alternative models. Then, with the best model configuration obtained on the first evaluation, a set of standard tasks proposed by the authors of the data sets used were performed and the results obtained were compared against the best results found in literature.

With all these concepts in mind, the main contributions of this work are:

1. A novel real time and incremental learning methodology for time series classification: The proposed methodology was built while still considering the

properties of the WiSARD network. The result was a fast and static encoding technique that prepares the input pattern to be used by the neural network;

2. An investigation of the WiSARD network when applied in conjunction with other techniques: In addition to the proposed model, experiments with the WiSARD network were also performed in order to compare its performance in traditional approaches. These traditional approaches included simple resampling techniques and a hybrid ESN and WiSARD model;

3. A comparison of related models and the proposed methodology under similar conditions: Related models such as KNN with DTW, ESN with linear regression/WiSARD and two resampling techniques with WiSARD were also used and their results are presented for comparison. In particular, the two resampling techniques were selected as they contain the simplest instructions and were used as computational time benchmarks. Similarly, the KNN with DTW has demonstrated its robustness in many other related works and was selected for an accuracy benchmark.

A few works were published during the development of this research. The complete list of published works is presented in Appendix A. In the most recent, a hybrid model based on the Markov Localization Algorithm and the WiSARD network was used on a music tracking task [6]. This approach was able to predict the position of an original audio signal by listening to the recorded signal captured on a mobile device. Next, almost the same methodology described in this work was also published [7], although with a more restricted set of experiments. Third, a credit analysis application using an algorithm named ClusWiSARD was also presented [8]. Thanks to the structure of the WiSARD network, this model extends its functionalities by creating new discriminators when the pattern presented is not similar enough to the knowledge previously stored, acting as a clustering model. This work was developed by the LabIA team after the BRICS-CCI competition, were we obtained the third place. And finally, an intelligent and adaptative agent that plays the game rock-paper-scissors was also developed [9]. This is a highly dynamic game when played consecutively by two bots as they constantly change their strategy based on the feedback obtained from the other player. In addition to this, the same methodology could also be applied to any application requiring constant adaptation based on the feedback received from the environment.

Finally, an Android application for demonstrating the functionality of the proposed model was also developed. This application is described in Appendix B and is available for download on the Android market with the name SketchReader. This application allows the user to draw patterns on the screen and then teach the model its corresponding class. When a similar pattern is presented later, the model is capable of predicting its most similar class.

1.3 Structure of this Document

This remaining of the dissertation is organized as follows. Chapter 2 presents some approaches for solving this type of task, along with the neural network used and a few other related subjects. Some direct approaches were also chosen and described in this chapter. These approaches were selected as they are considered of simple implementation and therefore might be used as basic benchmarks to be surpassed. Other approaches are also included as they contain one or more similarities with the proposed model or simply because they are largely used in literature and usually provide solid results.

Following this, Chapter 3 provides a solid description of the proposed model. In this chapter, a basic description of the main idea is first provided and then a more complete description of how the algorithm works is given. Additionally, a few different algorithms for the creation of its inner structure and the way its output integrates with the output classifier are also described. Finally, a description of the preprocessing steps presented for each type of data used is also presented.

After the description of the proposed methodology, a series of experiments is presented. These experiments are described in Chapter 4. This chapter also includes a description of the data sets used, a list of comparison models chosen and the way some of the approaches described on the previous chapters are integrated. Results include the traditional cross-validation technique, comparison with state of the art results and a few other experiments to determine the best configuration of the proposed methodology.

Finalizing this work, Chapter 5 presents a short summary of the work presented in this dissertation, along with the results and objectives achieved. In addition to this, a few more considerations for further investigation are also indicated for future research.

Chapter 2

Models and Related Subjects

Although very few works have explored the characteristics of weightless neural networks in the context of time series classification, many approaches based on other techniques were proposed in literature. Based on their popularity, simplicity and similarity with the proposed model, three of these approaches were chosen for comparison with our model, these include two analogous resampling techniques, temporal and spatial; the echo state network (ESN) and the K-nearest neighbors (KNN). These approaches are all briefly described on this chapter, in addition to related models such as the WiSARD neural network and Dynamic Time Warping (DTW) technique.

The first approach, based on resampling, was chosen due to its speed and simplicity. Although it was not expected that it would achieve the best classification performance, it is, however, considerably fast and may be considered as a baseline for comparisons regarding training and classification times. The second model (ESN), works very similarly to the proposed model when only the initial structure is considered. In both models, a statically randomly generated reservoir is responsible for encoding the input stream before classification. The third model (KNN), uses DTW as similarity measure and was chosen due to extremely good results in related works. These results, however, usually comes with the cost of higher classification times that need to be overcome with other optimization procedures. Finally, the WiSARD model is also described here since it is an essential part of the proposed model, being responsible for the final classification after representing the input stream through KernelCanvas.

2.1 Resampling

The main challenge when dealing with time series classification is to adequately generalize the input sequence from previously presented sequences with different lengths. These may not be input to most classification models, as they usually

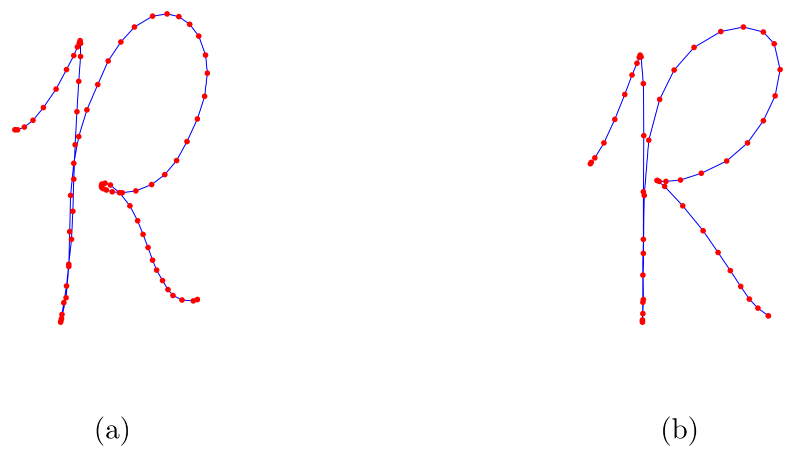


Figure 2.1: Two slightly different patterns to illustrate the same character being represented with different lengths. Pattern shown in (a) has 76 points, whereas (b) has only 53.

require samples with fixed input length. As an example, Figure 2.1 shows two handwritten characters representing the uppercase letter R created on the Android application created to demonstrated the proposed model in action and described in Appendix B. Letter (a), however, was draw more slowly than Letter (b) and is composed by 76 points, contrasting with only 53 points of letter (b). This happens quite frequently since the program captures each point approximately with the same time interval between each other, varying only depending on the amount of work on the CPU at that moment. Seeking a solution to this issue, this technique converts each sequence by interpolating their original points into a new sequence with fixed input length. After the application of this transformation, the new sequence is simply input to a traditional classification model.

Two methods were chosen for performing this transformation, these are the Temporal Resampling and the Spatial Resampling. Since the purpose of these approaches is to serve as a baseline for training and classification times, only the linear interpolation was used to generate the new points. Although simple, this approach is frequently applied, e.g., Dias et al. has used Temporal resampling to classify gestures performed on videos [1] and Alimoglu and Alpaydin have used Spatial Resampling before using other models to classify handwritten characters [10].

2.1.1 Temporal Resampling

In this technique sequences are resampled considering only the time each point was captured, i. e., its position on the list of points composing the sample. In order to apply it, the total number of points is divided in $N-1$ intervals of the same length, where N is the new number of points on the sequence. Each margin separating these intervals then generates a new point on the resulting sequence.

The entire procedure is described in details in Algorithm 1 and works as follows. For each point on the new sequence a linearly separated *index* is calculated and the position of its two nearest points on the original sequence are selected (*previous* and *next*). The new point is the linear interpolation of the points on these positions considering a rate *alpha*. The closer *alpha* is to zero the closer the new point is to the first point whereas the closer *alpha* is to one the closer it is to the second point. This procedure is repeated for every new point until the new sequence is complete.

```
Algorithm 1: TemporalResampling
   input  : $inputSequence$, the sequence of points to be resized
   input  : $newLength$, the length the inputSequence must be resized for
   output : $outputSequence$, the resized inputSequence with the desired
            length
 1 begin
 2     $outputSequence \leftarrow [\ ]$
 3     for $i$ in $0$ .. $newLength - 1$ do
 4         $index \leftarrow (inputSequence.length() - 1)/(newLength - 1) * i + 1$
 5         $previous \leftarrow floor(index)$
 6         $next \leftarrow previous + 1$
 7         $alpha \leftarrow index - previous$
 8         if $next > inputSequence.length()$ then
 9             $outputSequence.append(inputSequence[previous])$
10         else
11             $p1 \leftarrow inputSequence[previous]$
12             $p2 \leftarrow inputSequence[next]$
13             $outputSequence.append(linInterp(p1, p2, alpha))$
14         end
15     end
16 end
```

The main advantages of this method is that it is simple to implement and its execution time is considerably low. This method also keeps information about speed on the points, e.g., slower regions are represented with consecutive points with smaller distances and faster regions are represented with consecutive points with larger distances. This might be useful or not depending on the type of data being discriminated. When considering handwritten characters, a character that starts being drawn slowly and ends faster would not be perfectly matched to the same character being initially drawn rapidly and slower at the end. In this case, the first character would have more points describing its initial part whereas the second character would have more characters describing its end, so the middle of the sequences would not correctly match each other.

Figure 2.2 shows an example of the character in Figure 2.1a resized to length

20. As can be seen, the original density of points on each region of the character is proportionally preserved. For instance, the beginning, the end and each corner of the character are more densely populated than other regions.

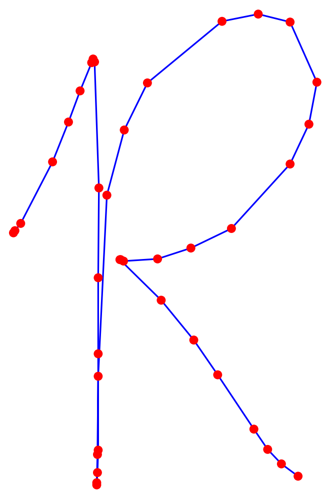


Figure 2.2: A handwritten character representing the letter R after application of Temporal Resampling.

2.1.2 Spatial Resampling

This procedure works very similarly to Temporal Resampling, but considers the overall stroke length when generating the new sequence, instead of each point position on the original pattern. This is accomplished by summing all distances from point to point, which corresponds to the total stroke length. After that, new points are generated by dividing the obtained stroke length in $N - 1$ intervals with the same length, where N is again the number of points on the new sequence. The rest of the procedure works similarly to the prior approach, where each margin on these intervals generates a point on the resulting sequence.

The entire procedure is described in Algorithm 2 and works as follows. For a given sequence to be encoded, the algorithm first calculates the entire length of the sequence (stored in the variable *total*) and then starts generating points by interpolating the points closer to the desired positions on this path. During each iteration the desired position is represented by the variable *desired* and corresponds to a linearly separated position on the entire path. The algorithm consumes new points until the walked distance is about to cross the desired position. When this happens the algorithm performs a linear interpolation between the next point to move and the current one. The variable *alpha* works very similarly to the previous algorithm, approaching the next point when it is closer to one and the current one when it is closer to zero. The only difference is that now it is calculated considering the walked distances instead of the distance to the point position on the array.

Algorithm 2: SpatialResampling

```
input   : $inputSequence$, the sequence of points to be resized
input   : $newLength$, the length the inputSequence must be resized for
output  : $outputSequence$, the resized inputSequence with the desired
          length
begin
    $outputSequence \leftarrow [\,]$
    $total \leftarrow 0.0$
    for $i$ in $2$ .. $inputSequence.length()$ do
        $total \leftarrow total + euclideanDistance($
                $inputSequence[i-1], inputSequence[i])$
    end
    $nextPoint \leftarrow 2$
    $walked \leftarrow 0.0$
    for $i$ in $0$ .. $newLength - 1$ do
        $desired \leftarrow i/(newLength - 1) * total$
        $step \leftarrow euclideanDistance(inputSequence[nextPoint],$
                $inputSequence[nextPoint - 1])$
        while $walked + step < desired$ do
            $nextPoint \leftarrow nextPoint + 1$
            $walked \leftarrow walked + step$
            $step \leftarrow euclideanDistance(inputSequence[nextPoint],$
                    $inputSequence[nextPoint - 1])$
        end
        if $step = 0$ then
            $outputSequence.append(inputSequence[nextPoint])$
        else
            $p1 \leftarrow inputSequence[nextPoint - 1]$
            $p2 \leftarrow inputSequence[nextPoint]$
            $alpha \leftarrow (desired - walked)/step$
            $outputSequence.append(linInterp(p1, p2, alpha))$
        end
    end
end
```

This implementation is equally simple when compared to the previous technique and performs similarly fast. In contrast to the previous technique, however, samples generated by this method do not tend to generate highly or little populated regions. In fact, most of the points appear to be approximately equidistant to their previous and next neighbors. Despite this, it has the disadvantage of giving more importance to longer distances between points, ignoring small details on the pattern. For instance, small circles and sudden changes on the stroke direction are discarded if the points distance is not big enough to represent these details.

Figure 2.3 shows an example of the character in Figure 2.1a resized to length 20 with this method. As expected, points are more homogeneously spaced between each neighbors and regions are almost equally populated. Furthermore, note that small details of the character on the main corners were lost.

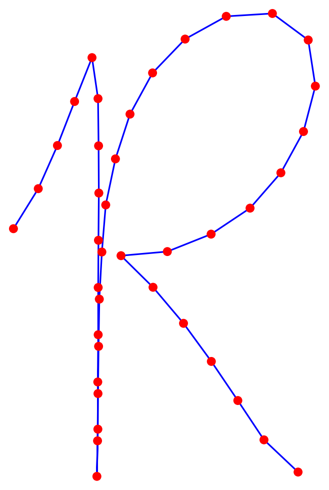


Figure 2.3: A handwritten character representing the letter R after application of Spatial Resampling.

2.2 *Echo State Network*

Echo State Network [11] is a special type of reservoir computing for dealing with streaming data and other related time series problems. The basic idea of the model consists in keeping an internal state x that represents the current context of the network. This internal state represents all inputs previously presented and is updated every time a new input u is received, while trying to preserve some of the existent information. This update takes into consideration not only the current input to the network, but also its current state and the last output o of the network. All this operation is done with the use of weight matrices, analogously to traditional feedforward neural networks. The main difference is that most of its connections are randomly defined when the model is initially created, never changing after that. Learning occurs only on the output layer. This layer receives as input the current internal state $x(n+1)$, the current input $u(n+1)$ plus the most recently output $y(n)$ and learns the next output $y(n+1)$ in a supervised way.

The main architecture of this model is shown in Figure 2.4, and the complete internal state's update equation is defined in Equation 2.1. This equation extracts information from all three sources: the input variable u, the current state x, and the previous network output y. The output feedback connection, however, is optional and sometimes omitted. In this case W^{back} may be considered entirely equal to zero, or simply omitted along with $y(n)$.

$$x(t+1) = f(W^{in}u(t+1) + Wx(t) + W^{back}y(t)) \tag{2.1}$$

The second function that must be defined in order to use this model is the output function, which is defined in Equation 2.2. In summary, this function concatenates all context variables $(u, x`and`y)$, applies a linear transformation (given by W^{out}) and finally applies the transformation function f^{out}. This is the only place were training occurs, and consists of calculate the best values for matrix W^{out} that closely match $y(n)$ to the desired output.

$$y(t+1) = f^{out}(W^{out}(u(t+1), x(t+1), y(t))) \tag{2.2}$$

Building and training these networks, when not considering the output feedback connection, can be accomplished in four steps [11]. These are: procure an echo-state network, choose input connection weights, run the network with training inputs, and compute the output weight minimizing the training error. The first step consists in building a reservoir which is rich in dynamics. This rich reservoir is essential to permit the internal state to navigate through a large portion of the state space, without falling on regions with no exit, therefore not losing the echo ability of the network. One simple method to prepare a rich reservoir consists in randomly setting the weights to values 0, +0.4 and -0.4 with probabilities 0.95, 0.025, and 0.025, respectively. This approach generates a sparse connectivity of 5% and encourages the development of small individual dynamics, which is the desired effect. The second step can be accomplished by randomly generating the input matrix without much special care, e.g., by setting their values to +1, -1 with equal probability. Next, starting with an arbitrary internal state, for instance $x = 0$, all input signals are input to the network and their corresponding representation on the internal state is saved. The initial internal state becomes irrelevant thanks to a property of this model called state forgetting. This property states that after some time the inputs presented to the system have effectively dissipated, therefore not interfering so much. Since this property is not valid for the beginning of the signals, it is a usual practice to discard the initial n states. The number n depends on the strength of the echo property, which depends on the weight values on W. Finally, with the list of internal states and their corresponding desired output it is possible to train the output layer. If the mean squared error (MSE) is used, this is usually achieved with the use of a linear regression algorithm.

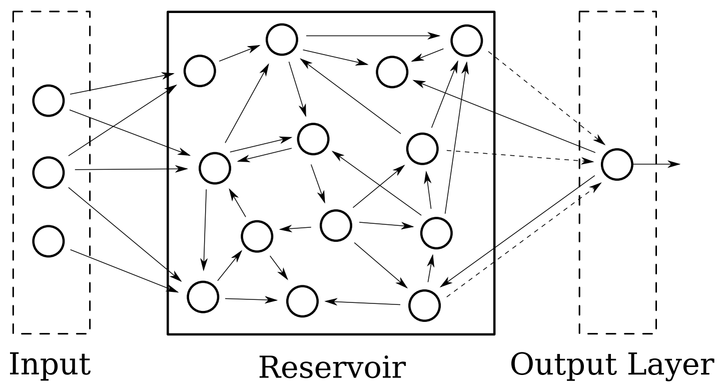


Figure 2.4: An example of an echo state network with three input variables, thirteen hidden nodes and one output neuron (Source [7]).

This model is not restricted, though, to be used on regression tasks. For example, training it to be used as a classifier can be accomplished by replacing the desired output to the corresponding sequence class. Later, during classification, the model outputs the class that appeared mostly during the processing of the signal received. Naturally, this procedure requires that the beginning and end of the input signal are known. However, it is a basic assumption that this information is previously known in this work.

Finally, it is important to notice that this model is not restricted to use linear regression on its output layer. In fact, any classifier or regression algorithm may be used, including feedforward and weightless neural networks. Despite this, using this basic algorithm has the benefit of achieving lower training times, specially when compared to other gradient-descent methods. These methods require multiple iterations to converge into a final model, therefore consuming much more computational time. On top of this, additional care needs to be taken to avoid other common problems, such as overfitting. For these reason, the echo-state network is usually used in conjunction with a linear regression algorithm. Still, even though its internal state contains a small echo of older inputs, this model gives much more importance to recent inputs instead of considering the entire sequence received. This characteristic possibly makes it harder to solve problems that need to correlate information from distant periods of time, for instance.

2.3 *K-Nearest Neighbors*

This is the classic nearest neighbors classifier [2]. Although it may seem like a naive approach at first, this is a popular choice applied in time series classification and frequently achieves results hard to beat [12]. Additionally, this model also presents some similarities to one of the mechanisms proposed, as discussed in more details in Section 3.1.

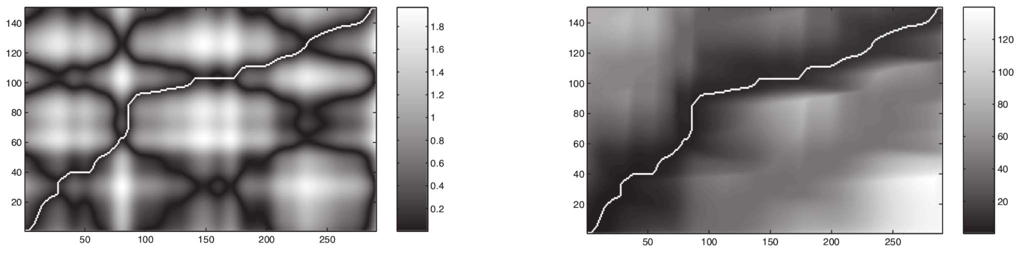
KNN is an algorithm based on similarity distance between two patterns. The idea consists in label unknown patterns by locating its K most similar patterns previously learned and choose the class that appears most frequently. Locating the most similar patterns is sometimes hard as it may require comparing the unknown sample to all known patterns, which is usually an expensive operation depending on the distance function used and the number of known patterns. If the euclidean distance is used, however, more complex algorithms like Spatial trees, such as [13], might be used to speed up the process. Despite this, this approach is relatively simple to implement and applicable to a large range of problems, including dimensional problems like recommendation systems. Naturally, euclidean distance is not a suited measure for time series data since sequences usually have different lengths. In order to bypass this issue, the Dynamic Time Warping technique is usually applied as solution. In short, this method is able to compare two sequences, matching each similar region between each other, and return the distance between both. More details about this method are described in Section 2.4

This approach also has the benefit that it only requires the definition of one parameter, the K. On the other hand, considering that the model needs to compare each pattern being classified to all other known patterns, it is expected that it consumes a reasonable large amount of time. This is particularly more evident in cases where the distance function has a higher order of complexity, such as the Dynamic Time Warping described in the next section.

2.4 *Dynamic Time Warping*

Dynamic time warping (DTW) [3] is a well known technique to compare time-dependent sequences. The main advantage of this technique is that it can apply nonlinear matchings between two patterns, called elastic matching. This characteristic allows this model to compare samples not only with different lengths, but also with internal variations.

DTW can be used to compare sequence features sampled at equidistant points in time. The only decision that needs to be made is what distance function to use when comparing the sequences. Usually, this function will return a low value for



(a) (b)

Figure 2.5: An example of DTW comparing two sequences. In 2.5a each cell represents the Manhattan distance from each point (i, j) whereas 2.5b represents the best accumulated path up to them. Both images are showing the best path found in white. (Source [3])

similar features and a high value for different ones, similar to the euclidean distance.

Once this function is chosen, the algorithm needs to build a cost matrix by comparing feature by feature from the sequences. Supposing these signals are S_1 and S_2, with respective lengths equal to L_1 and L_2, the resulting cost matrix would have a dimensionality of $L_1 x L_2$. Figure 2.5a illustrates a cost matrix generated by two signals using the Manhattan distance, darker regions correspond to similar regions whereas whiter regions correspond to more different regions. The next step is to find the best path P starting from coordinate $(0, 0)$ up to (L_1, L_2) which minimizes the sum of its values, i.e., the path cost. Intuitively, this corresponds to the best matching found between the two signals and the distance between them is the sum of the distances in the path. This sum is called the path cost and the accumulated cost past found for each cell on the cost matrix is shown in Figure 2.5b.

As expected, calculating the entire cost matrix is an expensive operation, with complexity $O(L_1 \cdot L_2)$. In order to speed up the algorithm, some alternatives to this model have been proposed, avoiding the need to calculate the entire matrix. The most commonly applied approaches are the Sakoe-Chiba [14] and the Itakura parallelogram [15]. The former considers that the nonlinearity in the signals is not so intense and therefore only the values around the main diagonal should be considered. All values outside of the center region are considered equal to infinity and ignored, without the need to calculate them. The second approach, on the other hand, considers it is more unlikely that the beginning and end of the patterns will differ and more likely that the middle of the patterns will. Only values inside their corresponding regions are considered, and all others are ignored.

2.5 *WiSARD*

The WiSARD model is a special type of weightless neural network used for pattern classification [16]. Like most models, training is performed through supervised learning, i.e., each example pattern is presented with its corresponding label to the network during a training procedure. It's main difference when compared to traditional approaches, however, lies in the way its knowledge is represented internally. Instead of using weight matrices, like in feedforward neural networks, the model uses a set of RAM units, each of them with a fixed number of input connections connected to the input pattern currently presented. One of the main advantages of this model is its fast training and classification times, making it ideal for real time applications.

This section is subdivided into three others. Section 2.5.1 describes the structure of this neural network and how its training and classification steps are performed, along with some advantages and disadvantages of the model. Section 2.5.2 presents a solution usually applied to solve the saturation problem of this network. Finally, Section 2.5.3 presents some methods commonly used to convert the input values into binary, a preprocessing step required to use this type of network.

2.5.1 Structure and Inner Workings

The structure of this model is quite modular, which makes it easy adapt and extend the model to handle different problems. Each WiSARD network is composed by units called discriminators, like in Figure 2.6, which corresponds to a class of the problem that needs to be solved. During the training phase, the pattern is presented only to the discriminator of its corresponding class. When a pattern is presented, its knowledge is incremented to recognize patterns similar to the pattern presented. If a similar pattern is presented later during classification, it is expected that this discriminator will output a high value, representing that it has seen a pattern similar to this before. Classification is done by simply presenting a pattern that needs to be classified to all discriminators, the class of the discriminator with the highest output is selected as the output of the model.

One useful measure of this model is the confidence C, which corresponds to the distance from the highest discriminator output to the output of the second highest discriminator, this confidence is defined by Equation 2.3. The more confident the network is, the more this variable approaches one whereas the less confident it is the more it approaches zero.

$$C = \frac{best_1 - best_2}{best_1} \tag{2.3}$$

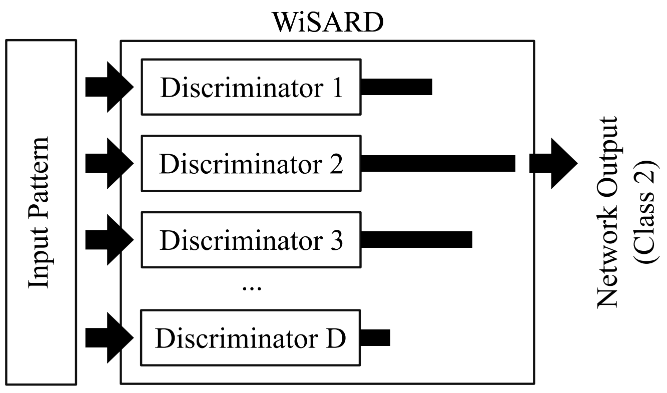


Figure 2.6: An example of a WiSARD neural network with D discriminators during classification. The class of the second discriminator has the highest output and therefore is chosen be the output of the network.

Selecting the discriminator with the highest output, however, doesn't make this model so different from traditional ones. What really differentiates it from others is the inner content of its discriminators. Each of these entities is composed of a set of RAM units, which store the knowledge acquired by the discriminator. Each of these RAM units has a certain fixed number of input bits which are randomly connected to the input pattern. The values received on these input bits, in turn, are used to address its content. During training, the value one is stored on the addressed position, whereas during classification the value stored is returned. If the position has never been addressed, however, the position is assumed to store the value 0. The output of the discriminator is the sum of all its RAMs, as seen in Figure 2.7. As can be easily inferred, the more similar the pattern is to patterns previously presented the higher the number of ones being summed, increasing the output of the discriminator. Naturally, since the input pattern is connected directly to the input bits on the RAM units, the input pattern must be properly converted to a binary representation first. Some methods for to make this conversion are briefly addressed in Section 2.5.3.

Thanks to the simple operations involved during training and classification, this model works considerably fast in many cases and is of particularly interest on real time applications. Additionally, as stated before, its structure is extremely modular making it easy to extend and integrate with other approaches. For instance, thanks to its segregation of each class on discriminators, adding a new class to the model is as simple as adding a new discriminator. Analogously, teaching a new style of pattern requires only the presentation of the example to the corresponding discriminator, this can even be performed between usage without the need to retrain the model. On the other hand, this model has some issues that need to be taken care. The first of them is the number of bits entering each RAM, the higher this

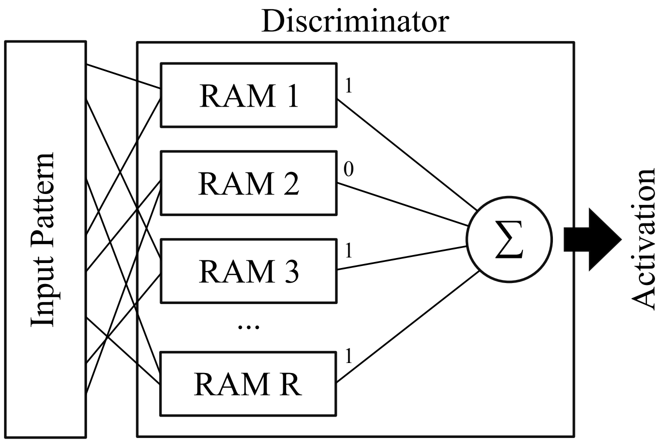


Figure 2.7: An example o a WiSARD discriminator composed of R RAM units. The output of the discriminator is the sum of all its RAM outputs.

number the higher the amount of memory required to represent the RAMs. Since any combination of zeros and ones might be received from the input bits, the total number of positions that might be addressed is 2^B, where B is the number of bits entering each RAM. This number grows exponentially but happily very few of these positions are actually addressed in practice and, therefore, a simple solution to avoid this issue is to use a hash structure, this way, only the relevant positions are used. The second problem which may occur is the saturation of the RAMs, which is when all, or almost all, of its positions are set to one. When this happens all discriminators output high values for every pattern presented. This is a common behavior when dealing with many misclassified examples or noisily data A simple approach for solving exists though, and is presented on the next section.

2.5.2 Bleaching

The bleaching technique was developed to surpass the saturation problem present on the classic WiSARD model. As stated on the previous section, the saturation problem happens when dealing with misclassified examples, noisy values, among other sources. This generates patterns with different classes but very similar between each other. The bleaching technique tries to solve this issue by considering not only if a position was written or not, but also how many times it was addressed. In the case of misclassified examples, it is expected that they would address a RAM position only a few times during training, whereas correct examples will address that same position several more times. To consider this scenario, the bleaching requires an integer to be stored on each RAM position, instead of a single bit. This integer is used to count how many times that position was accessed during training. Later, a dynamic threshold is used to accept if the RAM output is considered one or zero.

This procedure is illustrated in a classification being performed in Figure 2.8, where the threshold has a value of four. This means that only RAMs whose addressed position had a value higher or equals than four will output one, all others will output zero. A common way to define this threshold is to variate its value linearly from one up to the maximum value stored on a position. If the output confidence is too small, the threshold is increased, trying to solve the disambiguation. When the confidence is higher enough the model outputs the corresponding class.

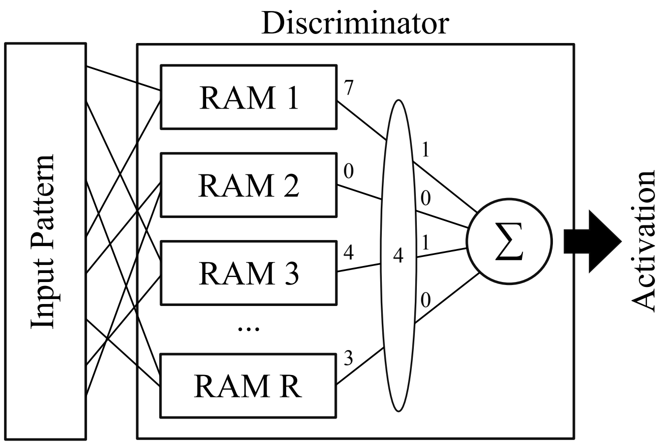


Figure 2.8: An example o a WiSARD discriminator applying a bleaching threshold with value four. Only RAM units with output higher or equal to this value take part on the sum.

2.5.3 Converting Real Values to Binary Representation

As stated before, the WiSARD network requires all inputs to be previously converted into a binary representation. This step is necessary because these inputs are used to address the RAM positions when presented to the discriminators, both during training and during classification. A good way to identify if the used binary representation is good is through Hamming Distance [17], which is represented as $h(X, Y)$ in this work and expressed in Equation 2.4. In this case, a binary representation is considered good if the hamming distance between two consecutive numbers is small and large between two distant numbers. This is a desired quality because if the Hamming Distance is small it means that few RAMs will be affected during addressing, generating similar addressed positions. The same is valid for more distant values with larger hamming distances. If the distance is large, it is expected that different RAM positions will be addressed, consequently the patterns will differ more.

$$h(X,Y) = \sum_{i=1}^{l} abs(X_i - Y_i) \tag{2.4}$$

The first binary representation usually considered is the representation of the number in base 2, however, this representation is not adequate as in many cases it exhibits big hamming distances between close numbers and small distances between larger numbers. For instance, consider the number 7, represented as 0111, this number has hamming distance 4 when compared to the number 8, represented as 1000. This is much larger than desired, specially when analyzing the distance from number 8 to 0, represented as 0000, which is equal to 1. A representation commonly used instead is the unary coding (sometimes called thermometer coding). This representation starts with an array of zeros representing the lowest number and this array with ones from right to left until the highest value. For instance, the number 0 could be represented by 00000, the number 4 by 00011 and the number 10 by 11111. The number of ones T on the pattern for a given value x, considering the minimum and maximum acceptable values, can be calculated trough Equation 2.5. As can be seen, this method proportionally fills the array while preserving the desired hamming distances, as desired. Although this representation has the desired qualities the model requires, it also requires a larger number of bits to represent numbers with an elevated precision. This characteristic, however, is usually not considered an issue as WiSARD scales well even with large input patterns.

$$T(x) = \frac{x - min}{max - min} \cdot n \tag{2.5}$$

Another decision that needs to be taken into account is related to the interval that must be mapped, therefore the constants max and min in Equation 2.5. On the previous example, this representation could map any number in the range $[0, 10]$, however, real problems might have intervals much larger and values not uniformly distributed. An approach for dealing with this is to normalize these values with Z-Score and than applying the math function *tanh*. The first operation subtracts the mean of each dimension on the input pattern and subsequently divides its value by the corresponding standard deviation. The result of this transformation is that each dimension has a mean equal to zero and standard deviation equal to one. Next, the application of function tanh, reduces the numbers space from $[-inf, +inf]$ to $[-1, 1]$. Thanks to the properties obtained after the application of the Z-Score, these numbers will be approximately distributed over the new range. This approach, however, assumes that the numbers on each dimension of the pattern follow a normal distribution, otherwise another approach must be considered.

2.5.4 Implementation Considerations

The biggest problem with the direct implementation of the WiSARD network is the amount of memory it requires to be executed. As stated before, the number of memory positions inside each RAM is equal to 2^B, where B is the number of bits entering each RAM. If B is larger enough a single RAM unit may easily occupy the entire computer memory, making this model completely unusable. In practice, however, this is never the case because most of the RAM inner positions are never accessed, therefore it is not necessary to allocate the complete memory array. Instead, a simple solution commonly used is to use a HashMap where for a given address position its corresponding memory content is returned. Similarly, if the position was never addressed before it is assumed that its content stores the value 0, i.e., a position never written during the training phase.

Besides this, there is also the issue related to the size of B. If B is larger than the basic type used (e.g. a long long), the key to the HashMap would be corrupted when decoded. This is not the case of languages such as Python and Ruby because these languages are able to automatically convert the number into a larger representation, but languages such as C, C++, Java and many others require further processing. In this case, the best solution is to define a custom object type with a customized hash function and equality method to be used as key on the HashMap, keeping the quality of the key used.

Finally, the traditional order of the WiSARD structure ($WiSARD \rightarrow$ *Discriminator* $\rightarrow$ *RAM* $\rightarrow$ *Position*) is not the most optimal in terms of implementation performance. In this case, for all D discriminators there are R rams to be accessed, that is, there are D times R HashMap accesses. If this order is reversed, that is, the RAM unit stores an array of length D ($WiSARD \rightarrow RAM \rightarrow$ *Discriminator* $\rightarrow$ *Position*) this becomes only R HashMap accesses. Even though the amortized complexity is the same, this usually improves the model performance by a constant factor. Furthermore, the HashMap inside the RAMs may also be used with alternative representations for the list of discriminators. For instance, consider a problem with thousands of classes (discriminators). In this scenario many RAM position are accessed only by a few of them and it would be wiser to use a more compact structure such as a TreeMap instead of an array. This approach usually saves large amounts of memory, while also speeds up the classification time.

2.6 Consolidation of Approaches

The models presented so far have many advantages but also some issues as consequence of their approaches. Most of these issues were considered when developing the proposed model and avoiding them was one of the main objectives of the approach described in the next Chapter. As result of this, almost none of them are present on the proposed methodology. A comparative table, with the characteristics considered relevant while addressing this task, is presented in Table 2.1, the Kernel-

Canvas approach is also shown as a reference to motivate the reading of Chapter 3.

Table 2.1: Comparison of model characteristics

	Training time	**Classification time**	**Incremental learning**	**Forgetting property**	**Elastic matching**
Temporal resampling	low	low	yes	no	no
Spatial resampling	low	low	yes	no	partial
KNN with DTW	low	high	yes	no	yes
ESN	low	low	no	yes	partial
KernelCanvas with WiSARD	low	low	yes	no	partial

Table 2.1 contains five relevant characteristics identified in this work. These are: Training and classification times, incremental learning, forgetting property, elastic matching, and complex matchings. Training and classification times are two important factors since most applications require real-time response. The only model considered in this work with problems in this criterion was the KNN with DTW. Although it essentially has a zero training time it needs to compare the unknown sequence with all known training sequences, which is a slow operation and not directly suitable for real-time applications. Next, incremental learning is the ability to continuously learn new examples and classes as the model is being used. Only the traditional ESN does not contain this property, requiring the model to be retrained when a new sequence is added. The forgetting property states that every time a new feature is presented the older ones are slowly being forgotten. The older the observation was presented the more likely it is that it does not influence the current internal state on the model. Since the beginning of the pattern may contain key information to help classification, this is not a desired property in this task, although it might be useful on regression tasks, as it helps the model focus on the most recent context. Only the ESN model contain this property. Finally, elastic matching considers the model capacity to deal with nonlinearities on the sequences, such as slowly drawing parts of a character as opposed to continuously drawing the entire stroke with the same speed. Direct matchings such as temporal resampling may end up comparing different observations after resampling simply because it

did not correctly aligned them, reducing the efficiency of the classifier used. This property also takes into consideration complex sequences, such as how many times a circular movement was performed. In this case, it is necessary that the model keeps track of the entire movement, completely matching it with previous known sequences.

Elastic pattern matching is the only characteristic not fully achieved by Kernel-Canvas, although it is capable of dealing with small nonlinearities. Despite this, the methodology still achieves very competitive accuracies on the data sets used even when compared to KNN with DTW, but with much faster response times.

Chapter 3

Methodology

As depicted previously, classification of time series data is a challenging task. This chapter aims to describe the model developed in this work the preparation of input sequences to be classified by a neural network, in this case the WiSARD. The initial inspiration for this approach came from observing a single feature as a point in space and the whole sequence as a unique stroke. Taking this perspective as a starting point, the model quickly generalized from the gesture recognition domain to other related time series tasks.

The objective of approaches like resampling and ESN is to find a relevant internal representation with fixed length which still keeps the original signal information. In many cases, this is accomplished by the application of a transformation procedure which is able to convert an input signal, composed of a sequence of features in R^n, into a single fixed point with dimensionality R^m, possibly in a much higher dimensionality ($m >> n$). This process is illustrated in Figure 3.1 and, ideally, shall preserve all its relevant properties. Relevant properties are characteristics that may help during classification and some examples include the speed of the stroke, position in space the observation occurred, direction of the movement, among others.

As expected, relevant properties depend on what needs to be classified, therefore the complete methodology is divided in three main components. The first component corresponds to the preparation of the input signal, including some basic transformations that highlight the desired properties. The next component is the KernelCanvas, which receives the prepared input signal and converts it into a fixed point in space, which is called canvas in an analogy to a painted canvas. Finally, the last component is the WiSARD model, which receives the prepared canvas and performs a training or classification with the generated canvas.

All these are described on the next subsections. Section 3.1 describes the details of the KernelCanvas and how its output is used with the WiSARD network. Section 3.2 describes three algorithms used to generate the kernels. Finally, Section 3.3 describes all preprocessing steps used in this work when preparing the input

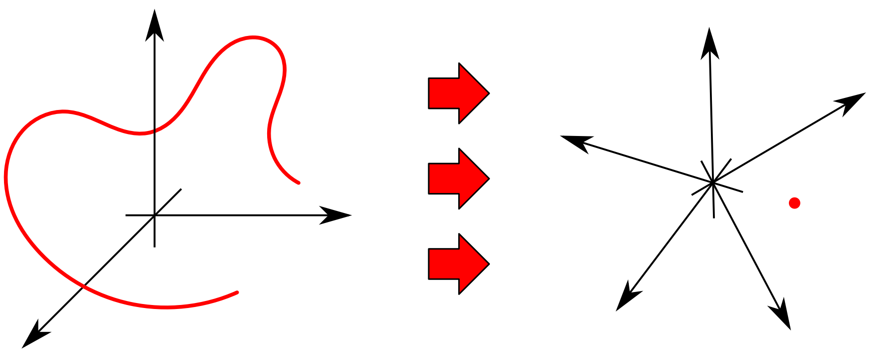


Figure 3.1: An illustration of the approach taken in this work where an input stroke is converted into a fixed and representative point in the new space.

patterns, which depend on the nature of the data.

3.1 KernelCanvas: Structure and Usage

After analysis of the models presented in Chapter 2, it was understood that they approach this problem in two distinct ways. These are transforming the input sequence into a fixed length input pattern, and dynamically matching the current pattern against all its stored knowledge. The model described here resembles these two characteristics through a simple, yet efficient, approach. Additionally, since only few and simple operations are required, this method encodes input sequences extremely fast, making it an excellent option for real time applications. Summing up to these benefits, due to the nature of the algorithm, the output can easily be expressed in binary format, i.e., zeros and ones. This is specially useful when weightless neural networks are used, as they already expect the input to be in this format, eliminating further preprocessing steps. Nevertheless, the model could also be used with other types of network since binary numbers are just a special case of numbers restricted to only two values.

Most insights for developing this methodology came from the idea of considering any time series sequence as a stroke that could be drawn on a limited surface, like a canvas. This is essentially how the method works, after generating a representation of the strokes on a canvas it presents the generated pattern as input to the neural network. First, the input sequence is prepared with the application of preprocessing steps. Next, the input sequence is input to the KernelCanvas, which draws the stroke received into an inner memory, called canvas. And, finally, the canvas is presented as input to a neural network, in this case the WiSARD.

3.1.1 Drawing Patterns Over a Grid Based Canvas

The initial idea for representing the canvas on the KernelCanvas was to slice the input space into fixed regions with equal area on each of its dimensions. This modeling generates a grid that could be represented internally as a matrix, and painting would occur after presentation of each input feature to the canvas. If the input feature passes over a region, or close enough to it, the region is painted in black (0), otherwise it remains white (1). After all observations were presented, the canvas has an internal representation of the sequence presented and may be input into a neural network, as shown in Figure 3.2. This approach works not only with two dimensional observations but also with much larger ones, requiring only the use of multidimensional matrices.

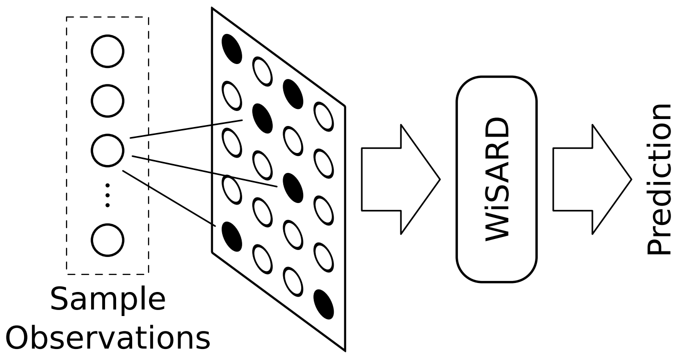


Figure 3.2: Representation of the painting process on the canvas which is later input to the WiSARD. Each observation is presented individually and the nearest kernels are selected and painted black (0).

The biggest drawback on this approach, as can be trivially perceived, is that the number of cells in this matrix grows exponentially as a function of the dimensionality of the feature space and the number of regions each dimension is divided. For instance, suppose a two dimensional example were each dimension was divided into 10 slices, generating a matrix with 100 cells, as shown in Figure 3.3a and 3.3b. This may seem like a reasonably approach at first, however, considering the case that the preprocessing step has added two more dimensions representing the direction of the movement and, additionally, concatenated the current feature with its previous and next neighbors. This process increases the number of dimensions from 2 to 12, requiring an amount of 10^{12} cells, which is a large amount of memory to be stored and processed. The problem here is that the number of cells is directly linked to the number of divisions and dimensionality of the feature space. A more efficient approach that does not face this issue is presented next.

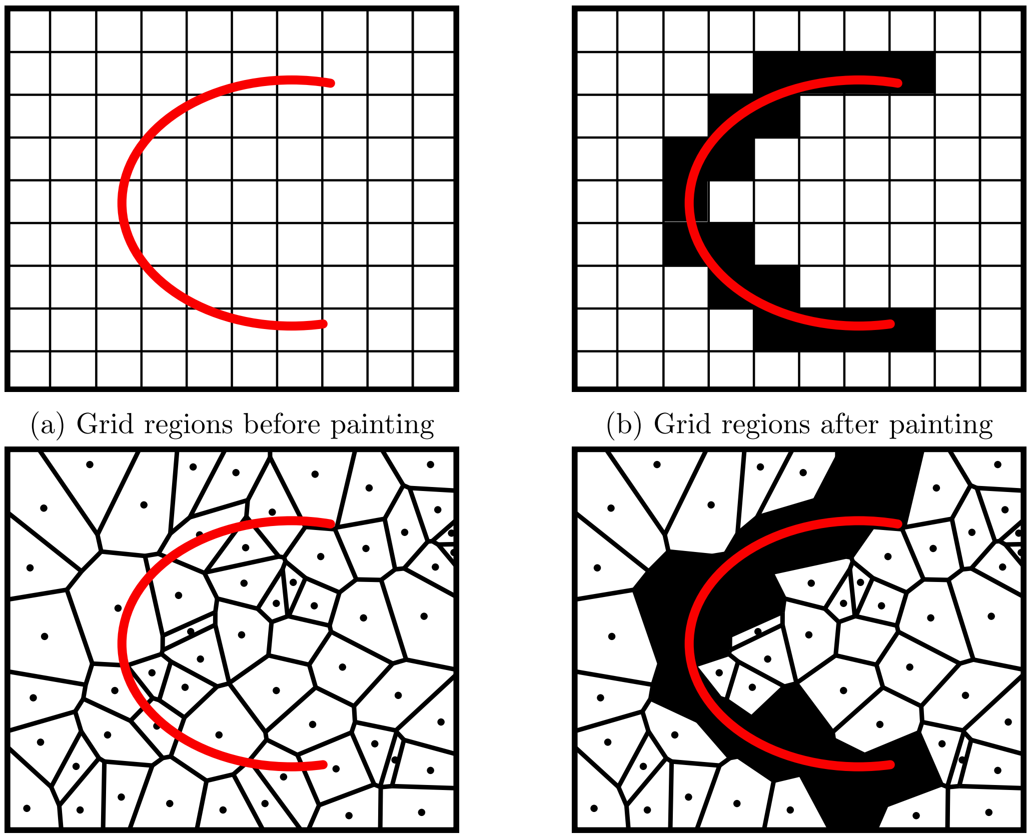
3.1.2 Representing Kernels by Randomly Generated Points

In order to avoid the exponential growth of memory in the grid based canvas, a slightly different approach was considered. Instead of defining squared regions as a function of the space size, a fixed number of regions is defined and each one is represented by a kernel located in its center. These kernels are randomly generated during the creation of the canvas and do not change after that. Painting occur as follows, the nearest kernels are first selected and painted the same way as before, selected kernels are painted black (0) whereas unselected kernels remain white (1). Only a certain number of kernels is painted during the presentation of a single observation, and this number is equal to $\alpha \cdot K$, where K is the number of kernels on KernelCanvas and α is named the activation rate, which is the percentage of kernels that are painted after each feature is presented. This procedure continues until all features on the current sequence were presented. Figure 3.3c and Figure 3.3d show an example of this approach over the stroke representing the letter C before and after painting the kernel regions, respectively. As can be seen, each region becomes defined by its nearest kernel, producing a picture similar to a Voronoi diagram [18]. Although there are many smarter approaches for generating the kernels, as described in Section 3.2, assuming they are uniformly generated should suffice for now.

3.1.3 Training and Classification

Preparation for training and classification with the WiSARD network is quite straightforward. As described before, at the end of the process the KernelCanvas has each Kernel marked as either black (0) or white (1), Preparation of the output consists in attributing a fixed number of bits B to each kernel, and setting its value to 0, if it was marked black, or 1, otherwise. The replication of the output color in B bits is a useful technique to avoid only a single RAM receiving that kernel information. If a different model was used, however, it would make sense to have only a single value for each kernel, reducing dimensionality of the output canvas.

With the canvas properly built, it may then be input to WiSARD for either training or classification. Alternatively, just as the ESN model, any other supervised learning model could be used on its output. The only drawback is that the number of binary bits on the canvas is between a few hundreds or thousands, which might be an issue on the convergence of some models. Also, since the kernels construction occur only once at the beginning and are always kept the same, the KernelCanvas works by quickly encoding sequences on the same way. Thanks to this, all other properties of the WiSARD model, like incremental learning and speed, are preserved.


(a) Grid regions before painting

(b) Grid regions after painting

(c) Kernel regions before painting

(d) Kernel regions after painting

Figure 3.3: An example of a two dimensional canvas being painted after presentation of a stroke representing letter C on grid based matrix (3.3a and 3.3b) and the kernel approach (3.3c and 3.3d).

3.1.4 Related Characteristics and Limitations

A few similarities of this methodology when compared to other approaches like KNN and ESN models were identified during the development of this work. In addition to this, two issues related to the way this model works were also observed and are described here.

First, a comparison with the KNN method becomes immediately apparent when analyzing the way kernels are selected and painted. When a feature is presented to KernelCanvas, its nearest kernels are selected and painted black. This is essentially the same done in KNN, however, KernelCanvas works differently as each feature is not compared against features from other samples, but to kernels that exist inside its structure. Additionally, KNN directly compares the current sample against all other samples presented before. On KernelCanvas, the current features are only compared against the available kernels, which set is usually much smaller than the number of samples available on the training set used by KNN. On top of that, the comparison function used is simply the euclidean distance, which is much faster than DTW. Comparatively, KNN with DTW can be seen as a model that compares the complete list of input patterns whereas a "mini-KNN", present on KernelCanvas, compares only smaller features of the pattern being presented. For instance, the direction and position in space that the stroke is moving. In the end, the painted regions on the canvas contain the main features that compose the stroke, in a way similar to a small signature.

When compared to ESN, differences are much more representative. ESN works in a similar fashion as it receives features one by one and updates its internal state. This update, however, occurs in a different way as it considers the current input and the current context, which is defined by all inputs received previously. This approach might result in problems if, e.g., the beginning of the signal had some sort of noise or distortion. A problem like this could disturb its current context, changing it into an incorrect state which would converge into a completely different state in the end of the sequence. This issue does not happens on KernelCanvas thanks to the absence of a feedback connection to the canvas. When an invalid feature appears, it paints a few regions closest to that feature but this does not affect the remaining of the pattern, which will be painted on the correct regions. When the canvas is presented to the neural model later, only a small fraction corresponding to the percentage of noisy observations will be painted differently whereas all others will be painted accordingly. This difference makes the KernelCanvas give similar importance to every observation on the pattern, making it a more resilient model for dealing with such problems.

Finally, there are currently two issues identified in this model. First, the model is not well suited for continuously data streams alone, such as a continuous audio signal. In this case a previous step called segmentation must be performed in order to identify the begin and the end of regions of interest. This is a necessary step as the KernelCanvas needs to be cleaned before the features are presented to be painted and the classification is only performed after the last feature is presented at the end of the sample. This, however, is not the case in every type of data. For instance, when dealing with handwritten characters this information is naturally present as the pattern begins when the pen touches the capture surface and it ends when the pen leaves the surface. The second issue that must be considered is related to the way samples are compared. In this approach each kernel represents a random characteristic in the feature space and the classes are discriminated based on this information. If two or more classes, however, require the order they were performed or a summation of the number of times a characteristic was seen it may not be able to discriminate them adequately. This happens thanks to the nature of this approach, which is not based on a moving belief. Nonetheless, this is also the desired behavior because thanks to this characteristic the model gives equal importance to characteristics presented at the begin and at the end of the sample. If a moving belief was used, behaviors such as the forgetting property and errors from the beginning largely propagating to the final belief would also be present, which was not desired.

3.2 Kernel Sampling Algorithms

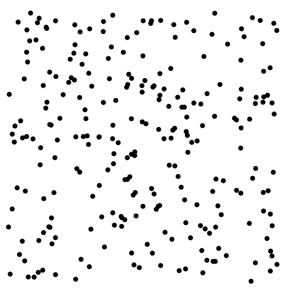
The task of kernel generation is an important step when using the KernelCanvas model. In fact, a good set of kernels might result in much better accuracy and less computation time. On of the possible issues with the purely random algorithm is that it may create many overlapping kernels, or large regions in space being represented by a single kernel. In the first case, the overlapping kernels may become irrelevant and are basically competing against each other to represent the same feature in space. On the second case, a large region being represented by a single kernel means that this region is being poorly represented, and perhaps should be divided and contain more kernels in it. In addition to this, the structure used to store these kernels has a relevant impact on the model performance. To compare all observations against all kernels is not an exhaustive procedure but might become an issue depending on the amount of kernels chosen. Assuming the euclidean distance is used when comparing their similarity, however, a space-partitioning tree could be used in order to reduce the search time for the nearest kernels. Such structures are constructed in a tree like graph where similar points (kernels) are stored together in organized regions of space, which are represented by leafs on the tree. Search is accomplished by accessing the tree and search starts on regions closest to the query point to more distant ones. One example of such structure is the k-d tree [13], which can query exactly or approximate nearest neighbors.

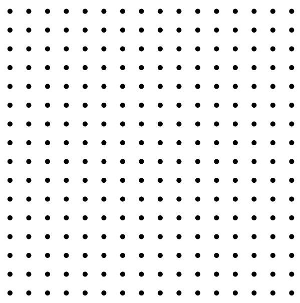
Alternatively, kernels could be generated taking into consideration the observations from the training set. Although promising, this approach was not investigated as one of the desired properties was to keep the incremental learning ability of the WiSARD network. Generating kernels considering only observations from the training set could generate large regions in space represented by a single distant kernel, considered of low importance. Therefore, the process of adding new classes would be handicapped as these regions could contain critical information that would help discriminating the classes. Despite this, generating the kernels considering the training set could also reduce the number of required kernels, speeding up the model and, perhaps, achieving higher accuracies. This approach, however, was not considered in this work but might be investigated in future researches.

The matter regarding completely randomly generated points is that there is absolutely no correlation between the kernels generated. The consequence of this is that there is no guarantee that the kernels are well distributed over space, in fact, it is quite common that many of them will be very close to each other whereas other areas will be completely unrepresented. Seeking an alternative to prevent this, the poison disk sampling algorithm was proposed to mitigate the issue. A set of poison disk points is a set of points with a minimum distance between each other but also tightly packed in space, resulting in a more organized distribution of points. This is also called a blue noise property, and it also states that its Fourier spectrum contains more high frequencies than lowers. Algorithms to generate Poisson disk sampling have been largely investigated and used to generate better anti-aliased images in computer graphics [19] [20] [21] [22] [23] [24] and [25]. Figure 3.4 compares three two dimensional kernel sets generated by the three algorithms described in this section. The Random Sampling algorithm, which is the random approach described in Section 3.2.1. The approximate grid algorithm, described in Section 3.2.2 which tries to map the given number of kernels into a multidimensional matrix, avoiding the exponential growth problem. And the Mitchell's best candidate algorithm, which generates approximate Poisson disk sampling kernels and is described in Section 3.2.3. As can be observed, random sampling generates much lower quality kernels whereas the other two algorithms generate much more well structured kernels.

3.2.1 Random Sampling

This is the simplest approach were each kernel is generated independently and uniformly random. Its algorithm is shown in Algorithm 3 which basically creates a


(a) Random sampling


(b) Approximate grid

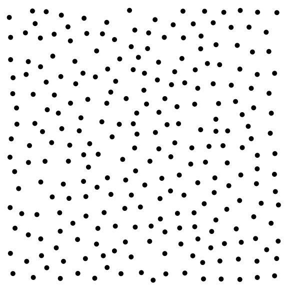

(c) Mitchell's best candidate

Figure 3.4: Three images with 256 kernels each comparing the quality of the kernels generated by each algorithm on a two dimensional space.

list of kernels to be returned as output. The function $createRandomKernel(dims)$ creates a kernel with $dims$ dimensions where each dimension is randomly set with a value in the range $[-1, +1]$. This algorithm represents the original idea and due to its simplicity is used as a benchmark against other methods.

A kernel set generated by this algorithm is shown in Figure 3.4a. As expected, the kernels are not well distributed over space, containing regions more densely populated and regions without any kernel. This is not the desired situation since it is possible that those regions badly represented contain key features that would help classification. Similarly, densely populated regions would probably just prejudice classification as multiple meaningless kernels would represent the same region.

3.2.2 Approximate Grid

Although it is not practical to take the grid based matrix described in Section 3.1.1, it is possible to build an approximated grid given a certain number of kernels. This may be accomplished by estimating the number of cuts on each dimension that will result in the approximate number of kernels desired. The algorithm used to generate grid based kernels is divided in two parts, these are Algorithm 4 and Algorithm 5. Algorithm 4 is the main algorithm but contains a recursive step which is represented

Algorithm 3: RandomSampling

```
input   : $numKernels$, number of kernels to be generated
input   : $dims$, dimensionality of the kernels
output  : $kernels$, the list of kernels generated
1  begin
2      $kernels \leftarrow [\,]$
3      for $i$ in 1 .. $numKernels$ do
4          $kernel \leftarrow createRandomKernel(dims)$
5          $kernels.append(kernel)$
6      end
7  end
```

by Algorithm 5. The first thing this algorithm does is estimate the number of cuts to be performed on each dimension. This step in done from lines 2 to 7 and the result is stored in list variable $coeffs$. Later, the recursive algorithm builds all possible kernels on the array $currentKernel$ using the calculated cuts, recursively iterating from the first dimension to the last one. When a kernel is complete it is copied and inserted inside the list variable $kernels$.

As expected, the grid based approach works well with low dimensionality problems like the two dimensional example in Figure 3.4b. In this case, the input values $numKernels = 256$ and $dims = 2$ resulted in $coeffs = [16, 16]$, which generated the representative grid observed. If the input values were $numKernels = 256$ and $dims = 12$, however, it would obtain $coeffs = [2, 2, 2, 1, 2, 1, 2, 2, 1, 2, 1, 2]$, which as can be seen does not representatively splits space over the dimensions.

Algorithm 4: ApproximateGrid

```
input   : $numKernels$, number of kernels to be generated
input   : $dims$, dimensionality of the kernels
output  : $kernels$, the list of generated kernels
1  begin
2      $coeffs \leftarrow [\,]$
3      for $i$ in 0 .. $dims - 1$ do
4          $k \leftarrow round(pow(numKernels, 1.0/(dims - i)))$
5          $coeffs.append(k)$
6          $numKernels \leftarrow numKernels/k$
7      end
8      $kernels \leftarrow [\,]$
9      $currentKernel \leftarrow newdouble[dims]$
10     $CreateNGKernels(dims, coeffs, 1, currentKernel, kernels)$
11 end
```

Algorithm 5: CreateNGKernels

```
input   : $dims$, dimensionality of the kernels
input   : $coeffs$, array of length $dims$ containing the number of
          divisions on each dimension
input   : $index$, the current dimension being divided, must start at 1
input   : $currentKernel$, a base kernel that will be used as temporary
          memory to build the kernels
in/out  : $kernels$, the list that will receive the kernels

begin
    if $index = dims + 1$ then
        $kernels.append(currentKernel.clone())$
    else
        for $i$ in 1 .. $coeffs[index]$ do
            $currentKernel[index] \leftarrow i/(coeff[index] + 2) * 2.0 - 1.0$
            $CreateNGKernels(dims, coeffs, index + 1,$
                    $currentKernel, kernels)$
        end
    end
end
```

3.2.3 Mitchell's best-candidate

Mitchell's best candidate [22] is an adaptation of the classic Dart-throwing [20] method that limits the number of tries the algorithm can perform. The classic algorithm starts with a single randomly generated point and keeps generating new candidates until the desired number of points is reached. If, however, a new point is closer than a distance d to any other point already inserted it is discarded and a new candidate is randomly generated. This process may take a lot of time to reach an end or even never stop. To overcome this issue, Mitchell's algorithm limits the number of tries before inserting a new point. This is accomplished by generating a fixed number of candidates C during every insertion. Among these candidates, their minimum distance to any other inserted point is calculated and only the candidate with the maximum distance is inserted. Naturally, this process generates a nearly optimal poison set as it may still generate points too close to each other or regions low populated, although with a much smaller probability when compared to random sampling. A more complete description of the algorithm is shown in Algorithm 6.

Many alternative algorithms were proposed to address the task of generating Poisson disk sampling, as referenced in the beginning of this Section. Despite this, the basic algorithm [22] was used because only a small number of kernels are required to be generated, consuming at most a second to be generated, and those algorithms are useful when dealing with hundreds of thousands points. Additionally, this process occurs only once during the creation of the canvas, does not impacting the other operations and not justifying the extra work to implement more complex algorithms.

Algorithm 6: MitchellsBestCandidate

```
input  : $numKernels$, number of kernels to be generated
input  : $dims$, dimensionality of the kernels
input  : $numCandidates$, number of candidate kernels
output : $kernels$, the list of generated kernels
begin
    $kernels \leftarrow [\,]$
    for $i$ in 1 .. $numKernels$ do
        $candidates \leftarrow [\,]$
        for $j$ in 1 .. $numCandidates$ do
            $kernel \leftarrow createRandomKernel(dims)$
            $candidates.append(kernel)$
        end
        $bestDistance \leftarrow -1$
        $bestCandidate \leftarrow -1$
        for $j$ in 1 .. $numCandidates$ do
            for $kernel$ in $kernels$ do
                $distance \leftarrow euclideanDistance(kernel, candidates[j])$
                if $bestDistance = -1$ or $distance < bestDistance$ then
                    $bestDistance \leftarrow distance$
                    $bestCandidate \leftarrow j$
                end
            end
        end
        $kernels.append(candidate[bestCandidate])$
    end
end
```

3.3 Signal Processing and Feature Extraction

As described in Section 3.2 all Kernels are randomly generated by sorting their attribute values within the range $[-1, +1]$. As expected, most samples come from different data sets, containing values much larger or smaller than these and, therefore, a normalization must be performed to fit their sequence values within this range. In addition to this, a few more preprocessing steps were applied depending on the type of data being encoded. After some experimentation, it was found that data sets with similar type of data did not required much specialized preparation. For instance, gesture recognition and handwritten characters basically shared the same preprocessing chain, which is described in Section 3.3.1. Other two data sets, containing audio samples, presented similar behavior and also shared a common chain, which is described in Section 3.3.2.

3.3.1 Handwritten Characters and Video Gestures

Video Gestures and Handwritten characters contain much similar data in the sense that in both cases their sequences may be represented in only two dimensional feature space. The former containing the X and Y variables corresponding to the hand of a person, for instance, on a video frame, and the later corresponding to the position of the pen on the capture surface. Additionally, in both cases the size of the movement as well the position it was performed were not considered relevant whereas the direction of their movement was.

The preprocessing chain used to prepare these types of data contain six transformations and is shown in Figure 3.5. The transformations represented in rounded rectangles were applied to all samples before splitting the data set in train and test set whereas normal rectangles were transformations applied after these sets were created in order to do avoid contamination of the train set with test samples. For instance, the operation “Add Rotations” creates two more samples, one slightly rotated clockwise and on counterclockwise. If this operation was applied before creating the train and test sets slightly similar test sequences could be present in the train set, which did not happen. The remaining of this section describes in more details each of these transformations.

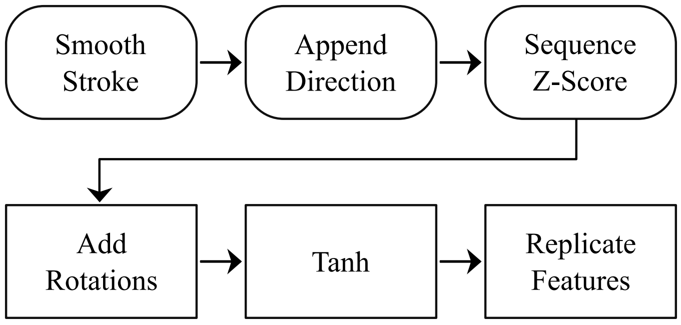


Figure 3.5: Preprocessing chain applied on gesture and handwritten character data sets.

Smooth Stroke: A lot of noise and irrelevant details come from these two types of data. For instance, handwritten characters may stop in a given point and prejudice the calculation of the movement direction on the next preprocessing step. As we are not interested in discriminate classes based on small stops performed on characters, this first step tries to remove these small details from the original sequence.

To remove these characteristics, a simple procedure based on the distance from current sequence feature to the last valid feature is described in Algorithm 7. As different data sets may contain samples with different sizes, each dimension is also divided by the largest standard deviation of each attribute, allowing the same distance value to be used in all data sets. The largest standard deviation is calculated in line 3 of the algorithm and line 8 checks if the current feature has a squared distance to the last valid feature larger than the minimum required distance. In this case, the minimum distance was estimated empirically and the value 0.01 was set.

Algorithm 7: SmoothStroke

```
input    : $inputSequence$, the sequence of points to be cleaned
input    : $minDistance$, accepts features at least this distant from the
           last accepted feature
output   : $outputSequence$, the smoother inputSequence
1  begin
2      $outputSequence \leftarrow [\,]$
3      $maxStd \leftarrow max(std(inputSequence))$
4      $lastFeature \leftarrow inputSequence[1]/maxStd$
5      $outputSequence.append(lastFeature)$
6      for $feature$ in $inputSequence$ do
7          $feature \leftarrow feature/maxStd$
8          if $sdistance(lastFeature, feature) >= minDistance$ then
9              $outputSequence.append(feature)$
10             $lastFeature \leftarrow feature$
11         end
12     end
13 end
```

Append Direction: Next, the direction of movement is calculated and concatenated to the current feature point. In the gesture scenario this is the direction of the part of the body being tracked, while for handwritten characters this is simply the direction of the stroke. This is composed of two attributes, which are the sine and cosine of the vector from the current feature and the next feature.

Sequence Z-Score: As each attribute may still present different distributions in the feature space, a Z-Score normalization is applied individually on each sequence to generate a zero mean and one standard deviation on every attribute. This helps the values to paint the entire canvas region, specially when used in conjunction with the function Tanh, as explained later.

Add Rotations: Handwritten characters and gestures frequently suffer small rotations when created by humans. This effect may happen because of many reasons, for instance, due to misalignment between the user and the capture device or simple, due to the indifference from the user when creating the sample, among others. The objective of this preprocessing step is to help the model discriminate these samples by replicating and adding a small rotation to each sample from the training set. In this case, each sequence was used to generate three samples, the first slightly rotated by -5 degrees, the second is the original sample and the last one is rotated by +5 degrees. The sequences were rotated by multiplying each of their sequence values to the rotation matrix, expressed in Equation 3.1. In this case θ is the desired angle to apply on the rotation.

$$R = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \tag{3.1}$$

Tanh: This is the mathematical function tanh applied on each attribute value to limit their range from the range $[-\inf, +\inf]$ to the range $[-1, +1]$, which is where the kernels were generated. As these values had already being normalized with Z-Score, these values will be approximately well distributed over the new range. As explained in Section 2.5.3 this is a useful technique to prepare the data to be converted to binary format. In this case, however, it behaves pretty similarly but helps the sequence use the entire canvas when presenting the stroke.

Replicate Features: Finally, each feature f_t on the sequence is concatenated with the features f_{t-1} and f_{t-2}. This is a short operation that enhances the amount of information each kernel represents, allowing more longer and complex characteristics to be represented.

3.3.2 Audio Signals

Similarly to handwritten characters and video gestures, a set of preprocessing steps were also applied on all sequences coming from the audio data sets. These steps made them adequately fit their signals on the same size of the canvas and additionally enhanced their characteristics, contributing to a better discrimination. The preprocessing chain applied is summarized in Figure 3.6 and, as can be observed, contains many similar steps from the previously described chain. Furthermore, it also follows the same representation rules, i.e., transformations represented in rounded rectangles were applied to all samples before splitting the data set in train and test set whereas normal rectangles were transformations applied after these sets were created.

Framing and Windowing: The LPCC, FFT and MFCC only applies to a small

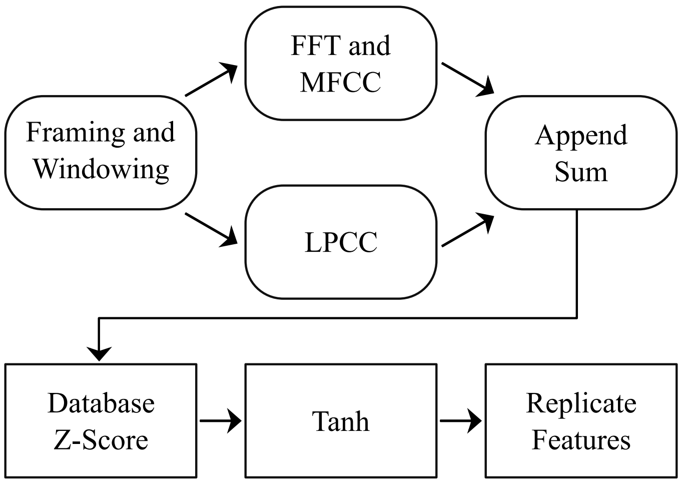


Figure 3.6: Preprocessing chain applied on audio data sets.

subsequence of fixed length N, representing the sound of only a small captured slice. Therefore, a previous technique called Framing and Windowing are also required. The former consists in sampling the audio signal at specific time interval and with fixed length, e.g., samples of 25ms captured every 10ms. The latter, multiplies each value on the frame in order to keep the continuity of the values inside the frame. A typical window applied is the Hamming window, defined by Equation 3.2. The parameter a defines the strength of the Hamming window, the closer it is to 1 the stronger it is.

$$w(n,a) = (1-a) - a \ cos\left(\frac{2\pi n}{N-1}\right), \qquad n = 0, 1, ..., N-1 \tag{3.2}$$

FFT and MFCC: Raw audio signals are usually represented by a sequence of real values containing the amplitude of the captured audio over time. This representation, however, is not well suited for direct work as an audio stream may contain distinct information separated on multiple frequencies. To separate this information, the Fourier Transform [26] is usually applied to convert this representation into the frequency domain. Sound, however, is captured in discrete intervals and as the Fourier Transform is defined on continuous domain, the Discrete Fourier Transform is used instead, which is represented by Equation 3.3. To calculate the Fourier Transform through Equation 3.3, however, is a slow operation with complexity $O(N^2)$. In practice, a Fast Fourier Transform algorithm, like the Cooley-Tukey algorithm [27] is usually applied.

$$X_j = \sum_{k=0}^{N-1} x_k e^{-i2\pi j \frac{k}{N}}, \qquad j = 0, 1, ..., N-1 \tag{3.3}$$

The frequency domain, however, contains issues when representing audio signals. First, the size of the frame is usually large, typically around thousands of values, and, second, the human auditory system does not give equal importance to higher frequencies as it does lowers. To address this, Mel-Frequency Cepstrum Coefficients (MFCC) [28] [29] are used to reduce the number of attributes on the frames and, additionally, apply a non-linear transformation to better simulate the human ears. This is accomplished with the application of two equations. The first, expressed in Equation 3.4 creates a relation between Mel frequency values f_{mel} and the normal frequencies f. This equation uses the log function to apply a non-linear transformation on the frequency domain, mapping more larger frequencies to the same Mel frequency banks. Also based on this equation, Q triangular filter banks are created linearly spaced in the Mel frequency domain and their corresponding energy banks E_k is calculated. Each energy bank is calculated by multiplying the corresponding triangular filter value to the corresponding frequency value. A typical number of filter banks used is 20.

$$f_{mel} = 25951\ log_{10}\left(1 + \frac{f}{700}\right) \tag{3.4}$$

Finally, a Discrete Cosine Transform (DCT) is applied to calculate each cepstrum coefficient C_i. The DCT is calculated by Equation 3.5 using the calculated energy bank values from the previous step. Usually, only 13 cepstrum coefficients are generated, therefore $M = 13$.

$$C_i = \sum_{k=1}^{Q} E_k\ cos\left(i\left(k - \frac{1}{2}\right)\frac{\pi}{Q}\right), \qquad i = 1, 2, ..., M \tag{3.5}$$

LPCC: The Linear Predictive Cepstrum Coefficient (LPCC) [30] [31] is another type of spectral envelop used to represent evolving signals. In this representation, cepstral coefficients are calculated from Linear Predictive coefficients which can be calculated through much simpler operations. In this approach, each framed signal s is assumed to be adequate to be expressed as a linear combination of its past values. The equation that correlates the LP coefficients and the framed signal is expressed in Equation 3.6, where $\hat{s}$ is the approximation of the framed sequence s represented as a linear combination of its past p values and a_k are the LP coefficients.

$$\hat{s}(n) = \sum_{k=1}^{p} a_k \cdot s(n - k) \tag{3.6}$$

Naturally, the coefficients must be bound to some restriction. In this case, they are bound to minimize the difference between $\hat{s}$ and s, i.e., minimize Equation 3.7

$$E = \sum_{-\infty}^{\infty} [s(n) - \hat{s}(n)]^2 \tag{3.7}$$

One common approach to estimate these values is by differentiating Equation 3.7 for each a_k, $k = 1, 2, ..., p$ and equating it to 0. This results in p linear equations and p unknown coefficients. With some further manipulation, it is possible to represent these equations as the autocorrelation sequence expressed in Equation 3.8, where N is the length of the frame.

$$R(i) = \sum_{n=i}^{N-1} s(n)s(n-i) \tag{3.8}$$

This representation is useful as it may be used to represent this problem in the matrix form shown in Equation 3.9.

$$\begin{bmatrix} R(0) & R(1) & R(2) & ... & R(p-1) \\ R(1) & R(0) & R(1) & ... & R(p-2) \\ R(2) & R(1) & R(0) & ... & R(p-3) \\ ... & ... & ... & ... & ... \\ R(p-1) & R(p-2) & R(p-3) & ... & R(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ ... \\ a_p \end{bmatrix} = \begin{bmatrix} R(1) \\ R(2) \\ R(3) \\ ... \\ R(p) \end{bmatrix} \tag{3.9}$$

This is a linear system of the form $AX = B$ which solution is given by $x = A^{-1}B$. In addition to this, matrix A is also a Toeplitz matrix, i.e., it is symmetric and all its diagonals are identical. This means that its inverse can be efficiently obtained with the Levinson-Durbin algorithm, providing the solution in $O(p^2)$.

After the acquisition of the LP coefficients, they may finally be converted to cepstrum coefficients through Equation 3.10.

$$c_m = \begin{cases} R(0), & m = 0 \\ a_m + \sum_{k=1}^{m-1} \left(\frac{k}{m}\right) c_k a_{m-k}, & 0 < m <= p \\ \sum_{k=m-p}^{m-1} \left(\frac{k}{m}\right) c_k a_{m-k}, & m >= p \end{cases} \tag{3.10}$$

Append Sum: After the original audio signal is finally converted into cepstrum coefficients, each of these attributes is summed, starting from the origin, and concatenated to the corresponding coefficients. This process is similar to an integration and, at this moment, the signal is convert into a stroke analogous to those present on the previous subsection with their corresponding direction movement, but in a much higher dimensional space.

Database Z-Score: Next, the signal is transformed with the application of Z-Score, i.e., each attribute has its mean subtracted and is later divided by its standard deviation. This time, however, the entire training set is used to estimate the and standard deviation, as opposed to the previous section, which used just the current sample to estimate these values.

Tanh: Keeping the same goal in mind, the mathematical function tanh is applied individually on each attribute to limit their values inside the range $[-1, +1]$, making each sample completely fit on the canvas size. This time, however, the resulting values also presented a much smaller variation as the Z-Score normalization considered the entire training set.

Replicate Features: Each feature was also replicated and concatenated on the same procedure performed before, feature f_t was concatenated with features f_{t-1} and f_{t-2} to give kernels more context information.

Chapter 4

Experiments and Analysis

In order to discover how well the proposed methodology performs when compared to other related models, a series of experiments was performed and described in this Chapter. Naturally, time series classification is a large area with many specific applications and different levels of complexity on each of them. Therefore, it seems like a fair condition that the proposed methodology should be evaluated in at least three task scenarios. For this reason, three types of tasks and five public available data sets were chosen. Furthermore, additional investigation was performed to determine how well the model performs as the number of kernels is reduced and which of the three kernel sampling algorithms, described in Section 3.2, was the most well suited for the majority of tasks experimented.

The remainder of this chapter is divided as follows, Section 4.1 describes the data sets, models, software and hardware used to develop or run the models. Next, Section 4.2 compares the three proposed kernel sampling algorithms and Section 4.3 compares the effects of different numbers of kernels on the average performance. Following this, Section 4.4 compares this model against the chosen models on all five data sets and also obtains the corresponding results of the proposed methodology on standard tasks. These are later compared to state of the art results from other works. Finally, Section 4.5 provides a partial conclusion of these results obtained in the chapter.

4.1 Experimental Base

This section detail the main characteristics on the simulation environment used to evaluate the models. It starts by describing the chosen data sets and their corresponding preprocessing chains applied, as well as any transformation applied before they were made public. Next, the models defined in Chapter 2 and the Proposed methodology, described in Chapter 3, are listed and a more detailed parameter configuration for each data set is provided. Finally, the hardware configuration used

Table 4.1: Summary of data sets characteristics

	Libras	**Pendigits**	**UJIPenchars2**	**Japanese Vowels**	**Arabic Digit**
Number of Samples	360	7494	7440	640	8800
Number of Classes	15	10	35	9	10
Length (std)	45 (0.0)	40.59 (12.18)	60.86 (25.89)	15.56 (3.62)	39.81 (8.55)
Recognition Task	Hand Movement	Handwritten Characters	Handwritten Characters	Speaker	Speech

to develop and run the experiments is also described, as it may impact experiment results regarding training and classification times.

4.1.1 Data Sets

As described previously, three types of time series data and five data sets were used in this work. The data sets were all chosen from the public available website UCI Machine Learning Repository[1] and the three tasks are related to video gestures, handwritten characters and audio recognition. The chosen sets that present these characteristics are: Libras [1], Pendigits [10], UJIPenchars2 [32], Japanese Vowels [33] and Arabic Digits [34]. A summary of the details of these sets is also shown in Table 4.1. The first three data sets, Libras, Pendigits and UJIPenchars2, were preprocessed according to the preprocessing chain described in Section 3.3.1, and the data sets Japanese Vowels and Arabic Digits with the preprocessing chain described in Section 3.3.2. The Japanese Vowels, however, used LPCC after application of the Hamming technique whereas Arabic Digits applied FFT and MFCC. The audio task is also divided in two more sub tasks, which are speaker and speech recognition. These two tasks are related to audio classification, however, the first sub task tries to identify the person speaking to the system whereas the second sub task is concerned about the content being said. A few more details about each of these sets is described below.

Libras: In this set, 15 of the most common types of movements from the Libras (Brazilian sign language) universe were chosen to be classified. Four people were filmed performing the different types of movements during two days, the result was a total of 360 videos which later were processed to extract their

[1]Available at: https://archive.ics.uci.edu/ml/datasets.html

hands coordinates. After locating the hand on each video frame, its centroid was calculated and the resulting coordinate stored. Only 45 frames of the complete video length were linearly selected to be processed, generating sequences with constant lengths. This restriction worked similarly to the application the Temporal Resampling technique, described in Section 2.1.1. Additionally, this set is divided in 10 subsets to facilitate comparison with other works.

Pendigits: This set contains a total of 7494 sequences created by 44 people representing numerical digits. All samples were created using a pressure sensitive tablet with an LCD and a cordless stylus pen. Despite the capability to capture pressure information, only the stroke coordinates were made available as the objective was to make the model compatible with devices without pressure information. This set is also divided in two other sets, to allow better comparison between different works, the train subset contains sequences created by the first 30 writers and the test subset contains the samples created by the remaining 14 writers.

UJIPenchars2: This publicly available data set is composed by a total of 11640 samples divided in 97 classes and created by 60 writers in total. All these samples were also captured on a cordless stylus with a tablet PC. This set, however, includes digits, letters and non-ASCII characters such as "¿" and "¡". A subset of this big set, however, is also provided for comparison and is composed by 7440 samples and 35 classes. These classes contain all 26 letters, lowercase and uppercase included on the same class, and 9 digits, from 0 to 9. The digit zero is included in the same class as o's. Only the smaller subset, with 35 classes, was used in this work as this was the preferred option used in other works [32] [35].

Japanese Vowels: Japanese Vowels contains samples from 9 male subjects uttering the vowels /ae/. The objective of this task is to correctly identify which of the subjects has spoken the vowels, i.e., a 9 class problem. This task is also divided in a standard train and set sets with 270 and 370, respectively. Samples have also being previously transformed by the application of LPCC, described in Section 3.3.2.

Arabic Digits: Arabic Digits is another digits only data set but with samples generated from audio recordings. It was generated by 88 individuals, 44 males and 44 females between the ages 18 and 40, each of them repeating the same digit 10 times. Each sample corresponds to a time serie with 13 Mel Frequency Cepstrum Coefficients, as FFT and MFCC were used in this set.

4.1.2 Comparison Models

In addition to the proposed model, five other approaches were selected to be compared with the proposed methodology and are described in this section. These models are the same models described in Chapter 2. Some of them, however, were combined in order to explore further ideas, e.g., to investigate the use of weightless neural networks with the ESN model. Each of these combinations are described bellow, along with the corresponding parameters used. Additionally, all approaches shared the same preprocessing chain, except by small adaptations when said otherwise, and all parameters were defined empirically during the cross-validation described in Section 4.4.

KernelCanvas and WiSARD: The first model is the proposed methodology. The KernelCanvas is used to encode the input sequence into an input pattern which is then presented to the WiSARD network for either training or classification. Regarding its parameters, the activation rate was set to 0.075 on all data sets, the number of output bits per kernel was set to 32 on Japanese Vowels and 16 on all others, and the number of kernels was set to 1024 on Libras and 2048 on all others. The kernel sampling algorithm applied was always the Random Sampling. In respect to the WiSARD network, the number of input bits on each RAM was set to 8 on Japanese Vowels, 32 on Pendigits and 16 on all others. Further details about the choice of the kernel sampling algorithm and the number of kernels are described in Section 4.2 and Section 4.3, respectively.

KNN and DTW: This is exactly the same model as described in Section 2.3 and Section2.4. We fixed $k = 1$ on all data sets as this was the optimum value obtained during cross validation. Additionally, the use of the Sakoe-Chiba band is a useful technique not only to improve performance but also to improve accuracy [36]. Therefore, the current implementation also contains a limit for the warping window which is equal to 10. Furthermore, the only preprocessing difference applied on the preprocessing chain was to remove the step that applies the function tanh, as it is a required step only for the KernelCanvas.

Echo State Network and Linear Regression: This is the default ESN model described in Section 2.2. All networks were created with the use of the Oger toolbox[2], containing 300 neurons on the hidden reservoir. Additionally, all input and inner weights were randomly set within the range $[-0.05, +0.05]$ and the leaking rate was set to 0.2. All reservoir neurons use the function tanh

[2]Available at: http://reservoir-computing.org/organic/engine

to propagate signals and the output layer uses linear regression as training algorithm.

Echo State Network and WiSARD: This is the same ESN implementation as above but using a WiSARD network replacing the linear regression algorithm on the output layer. The network parameters were also modified, the number of neurons on the reservoir was set to 200, the input weights were randomly set within the range $[-0.1, +0.1]$, the inner weight matrix was randomly set within $[-0.005, +0.005]$, and the leaking rate was kept at 0.2. The WiSARD implementation is the same used on all other models but wrapped on a python module through Cython language[3]. The connection between ESN and WiSARD is made by presented the entire input sequence to the ESN network and the corresponding inner state values are saved and transformed through Z-Score, tanh and unary coding with 16 bits, respectively. The prepared sequence is then presented to WiSARD. The number of input bits on the RAM units was set to 8 on Libras and Japanese Vowels, 16 on Arabic Digits, and 32 on Pendigits and UJIPenchars2.

Temporal Resampling and WiSARD: The default algorithm described in Section 2.1.1 with a WiSARD network as classifier. The sequences were resized to length 32 on UJIPenchars2 and 16 on all others. The number of input bits of WiSARD's RAM units were 8 on Libras and Japanese Vowels, and 16 on all others.

Spatial Resampling and WiSARD: Similar to the previous algorithm but following the algorithm described in Section 2.1.2. Sequences were also resized to length 32 on UJIPenchars2 and 16 on all others. The number of input bits on each RAM was set to 8 on Libras and Japanese Vowels, 16 on Pendigits and Arabic Digits, and 32 on UJIPenchars2.

4.1.3 Hardware Platform

As training and classification times are measured in milliseconds, it is also important to describe the hardware and software environment used to obtain the results presented. In fact, all experiments were performed on the same hardware architecture under similar CPU workloads. The computer used was a Dell VOSTRO 3360 with Intel® Core$^{\text{TM}}$ i5-3337U CPU and two dual-channel 4GB Corsair Vengeance memories at 1600MHz (CMSX8GX3M2B1600C9). Additionally, the operating system used was the Fedora 21 Desktop Edition installed on a Samsung EVO 120 GB SSD. Finally, in respect to the models used, with exception of ESN, which we used

[3]Available at: http://cython.org/

the implementation available on Oger toolbox[4], all other models were implemented using c++ as programming language and compiled with clang++ 3.5.0.

4.2 Comparison of Kernel Sampling Algorithm

As described in Section 3.2, the way kernels are created might influence the quality of the KernelCanvas and, therefore, three kernel sampling algorithms were proposed. One purely random approach, a Neareast Grid algorithm, and a Poisson Disk Sampling algorithm. The first algorithm creates kernels at random, without concern about previously created kernels or kernels on the same position. The second algorithm was created as a benchmark for the other two when compared to a multidimensional grid with equivalent number of kernels. Finally, the third algorithm is an extension of the first approach but creates more structured kernels and tries to avoid creation of similar kernels and low populated regions.

The three algorithms were used on all five data set and the average accuracy obtained is shown in Figure 4.1. This accuracy was obtained through stratified cross-validation on the same way as described in Section 4.4. As observed, except by the approximate grid algorithm the difference between the three algorithms is not much significative. As a conclusion, despite the extra care of the Poisson Disk Sampling algorithm, the random sampling algorithm seems to perform equally well or slightly better, not justifying the extra work.

4.3 Selection of the number of Kernels on KernelCanvas

Another relevant parameter that needs to be set for each data set is the number of kernels used. Therefore, this section provides a comparison experiment to justify the number of kernels used on cross-validation experiment and, additionally, supplementary information like training and classification times on each configuration were also calculated and summarized.

The experiment in this section was performed by applying a the same cross-validation technique described in Section 4.4 but exponentially ranging the number of kernels from 1 to 4096, doubling the number of kernels after each iteration. During this process, the average accuracy, standard deviation and computational times were calculated and all results were summarized in Figure 4.2, where the log2 of the number of kernels is represented on the X axis of all charts. As expected, the higher the number of kernels, the higher is the obtained accuracy. This behavior, however,

[4]Available at: http://organic.elis.ugent.be/organic/engine

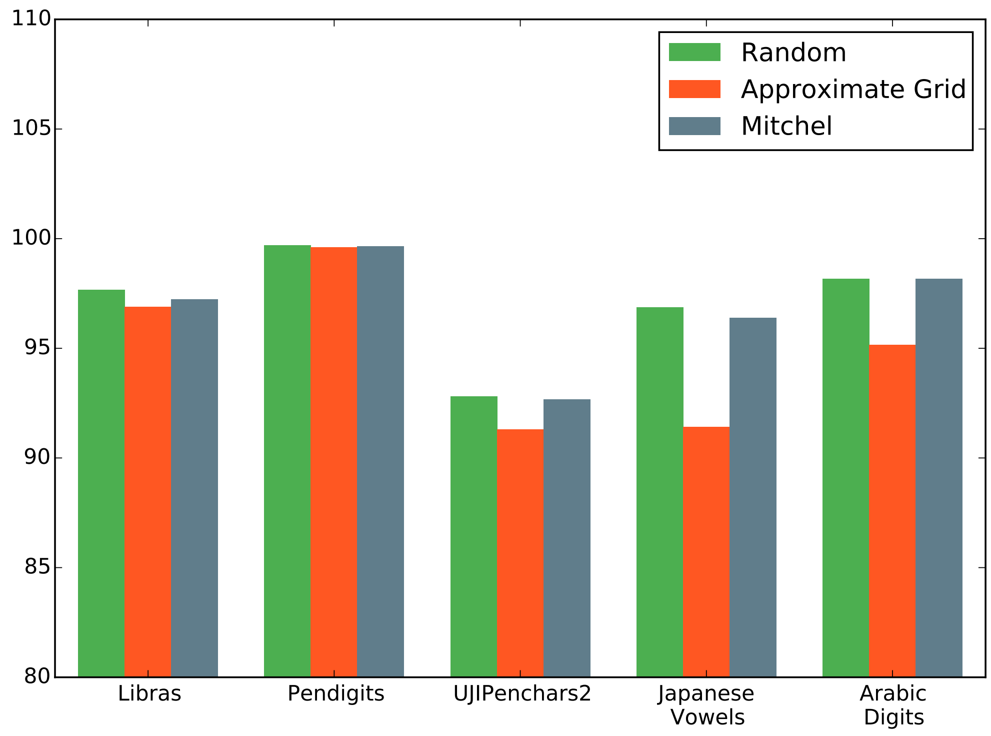


Figure 4.1: Average KernelCanvas accuracy with different kernel sampling algorithms.

does not endure indefinitely and for most data sets it stops close to $x = 10$, i.e., close to 1024 kernels. The inverse behavior is also observed on the standard deviation, the higher the number of kernels, the lower the deviation, i.e., the more confident we are with the obtained model accuracy. Finally, the two charts summarizing the training and classification times presented very similar behavior, demonstrating that the model complexity grows approximately linear as the number of kernels grow. As a result of all this, it was decided that 1024 kernels were enough for the Libras data set, and 2048 were enough for all others as the accuracy is approximately constant, the standard deviation is smaller and the training and classification times still qualify for real-time applications.

4.4 Comparison of the Models

The six selected models were compared in two distinct experiments. These were the average performance on a user-dependent cross-validation and distinct user-independent experiments, the latter using the standard tasks provided within each data set. User-dependent tests are tests were the training and testing sets contain samples generated from the same user base whereas user-independent tests contain their sets generated from distinct user bases. User-independent tests are interesting

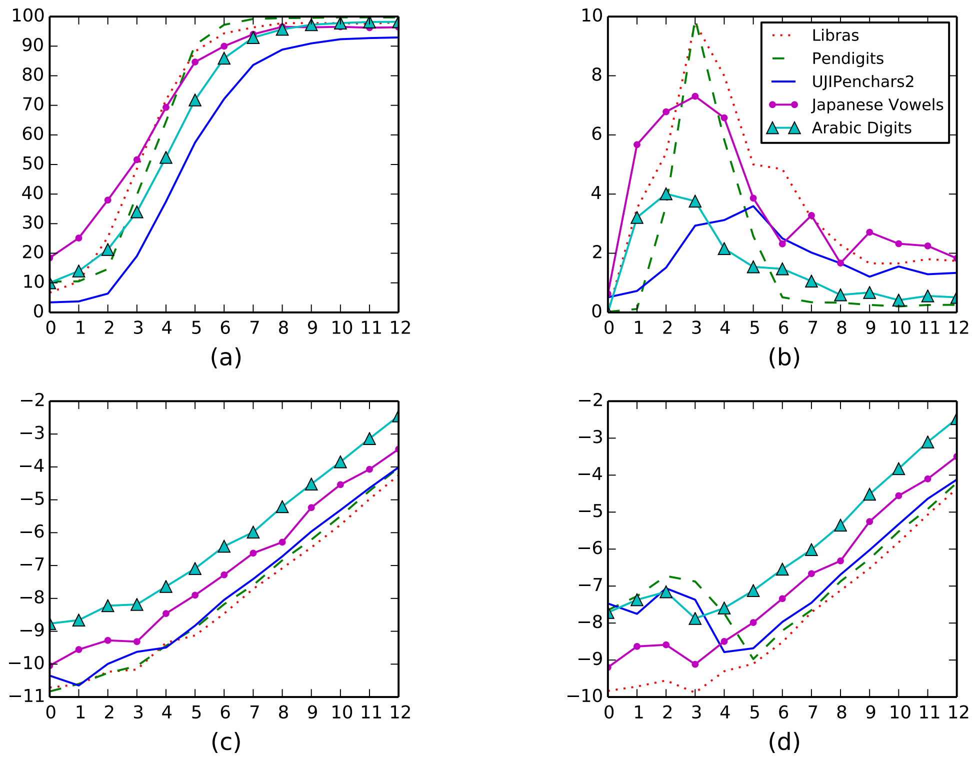


Figure 4.2: Results obtained by KernelCanvas in function of the number of kernels (normalized In log2): (a) Average accuracy, (b) Standard deviation of the accuracy, (c) Log2 of the average time needed to train each sample, (d) Log2 of the average time needed to test each sample.

because they better simulate real life scenario as final users do not participate on the development process. The first experiment, however, was also important to compare the proposed methodology to other models using similar preprocessing chains. The second experiment, is composed by standard tasks proposed by the authors of the respective data sets used and is useful to compare against the state of the art results.

Some results presented here contrast from those present on our previous work [7]. We justify this discrepancy to multiple modifications performed on our implementations and the newly created preprocessing chain. Among these modifications there have been the implementation of the Sakoe-Chiba [14] optimization, reducing the processing time of DTW, and additional configurations during the creation of the ESN. Such modifications were performed to obtain more competitive models and provide a fairer comparison.

4.4.1 User-dependent Comparison

This experiment was performed with the use of the well-established stratified 10-fold cross-validation. In this experiment, 10 subsets are randomly created from

Table 4.2: Accuracy and Standard Deviation on Cross-Validation (%)

	Libras	**Pendigits**	**UJI Penchars2**	**Japanese Vowels**	**Arabic Digits**
KernelCanvas + WiSARD	**97.78 (2.28)**	99.72 (0.20)	92.69 (1.23)	**96.22 (2.38)**	98.22 (0.49)
KNN + DTW	93.00 (5.21)	**99.87 (0.14)**	**95.24 (0.76)**	95.63 (2.26)	**99.83 (0.13)**
ESN + Linear Regression	93.89 (3.32)	96.86 (0.56)	75.65 (1.98)	95.94 (1.31)	91.98 (1.24)
ESN + WiSARD	93.00 (3.81)	99.44 (0.22)	91.13 (0.86)	95.47 (2.10)	99.61 (0.25)
Time Resample + WiSARD	88.33 (5.03)	95.05 (0.64)	82.07 (1.71)	94.96 (3.23)	98.83 (0.46)
Length Resample + WiSARD	88.78 (6.01)	94.81 (0.85)	80.52 (1.48)	95.27 (3.04)	98.35 (0.35)

the original data set and 10 evaluation procedures are performed. During each evaluation, one of the subsets is used as test set and the remaining is used as the entire training set. Training and classification are performed 10 times and at the end of the process average information such as accuracy and computational times are calculated and summarized. Additionally, these subsets were created in a stratified way, i.e., retaining statistic properties such as the percentage of samples per class of the original data set.

The accuracy results obtained for each model on each data set are all shown in Table 4.2, along with their corresponding standard deviation. As can be observed, the best results were basically divided between the models KernelCanvas + WiSARD and KNN + DTW. The former obtained the best results on Libras and Japanese Vowels, and the latter obtained the best results on Pendigits, UJIPenchars2 and Arabic Digits. Regarding the two ESN approaches, the WiSARD model obtained consistently better results when compared to the traditional ESN model using Linear Regression on the output layer. This was mostly observed on UJIPenchars2 data set where the accuracy improved by 15.48 units. In addition to this, this model also provided very competitive results on Pendigits and Arabic Digits. Finally, the two simple resampling techniques obtained the lowest results, winning only when compared to ESN + Linear Regression on UJIPenchars2 data set.

Similarly, the training and classification times were also measured and the obtained values were summarized in Table 4.3. As clearly observable, the two simple resampling approaches obtained the best training and classification times in the two categories. This is the expected behavior as they share equally simple algorithms and also use a WiSARD network on the output, which operates equally simply. Further, despite the best results of KNN on some data sets, it also presented considerably higher classification times, specially on the larger data sets, such as Pendigits, UJIPenchars2 and Arabic Digits. Additionally, most of its measured classification times became above one tenth of a second and would most likely be perceived by a final user. The worst case was for Arabic Digits, were the average classification time reached beyond a second. Likewise, it can also be seen that despite the improved results achieved by the WiSARD network as the ESN output layer when compared to the traditional approach, it also incremented the computational time. Further investigation should be considered, however, to explain this discrepancy. Finally, it can be seen that the proposed methodology presented really competitive computational times when compared to all other approaches.

4.4.2 Standard Tasks

As previously stated, the data sets used in this work provide subsets in order to allow easy comparison with other works. In this section, our results are compared with the best results we could find at the present time on these sets, i.e., the state of the art results. Most of these tests also differ from the previous experiment in the sense that they are directed towards user-independent scenarios. An stated previously, user-independent test is a type of test were all samples used during the training phase comes from one set of people and the set of samples used for testing comes from another. This way, it is expected that this type of test will simulate more accurately the results that would be found when the recognition system would finally be used by final users. The only exception to this rule is the Japanese Vowels data set, which is a speaker identification problem and therefore may not be tested on different sets of users.

These comparison results are summarized in Figure 4.3 and the best results found were published at [35], [35], [37] and [38], respectively from left to right. As can be observed, although it has not surpassed any of them, the model performed considerably well on all data sets. It is important to notice, additionally, that the proposed methodology has many other relevant properties such as scalability, computational time, incremental learning and simplicity.

Regarding Libras, the authors of this data set did not provide a single standard pair of training and classification sets. Instead, they proposed nine different tasks based on multiple samples collected along two days of experimentation. The test results obtained by the proposed methodology on such tasks are summarized in Table 4.4, the notation used works as follows. Four participants engaged on this

Table 4.3: Average Training (T) and Classification (C) Times Obtained During Cross-Validation (ms)

	Type	Libras	Pendigits	UJI Penchars2	Japanese Vowels	Arabic Digits
KernelCanvas + WiSARD	T	3.55	3.28	3.01	4.55	10.60
KNN + DTW	T	0.00	0.00	0.00	0.00	0.00
ESN + Linear Regression	T	4.63	4.53	5.27	2.85	6.04
ESN + WiSARD	T	50.29	52.24	62.87	25.23	66.49
Time Resample + WiSARD	T	0.17	0.03	0.15	0.68	0.83
Length Resample + WiSARD	T	0.82	0.08	0.28	0.53	1.12
KernelCanvas + WiSARD	C	3.35	3.24	2.81	4.26	10.08
KNN + DTW	C	128.41	117.95	245.45	27.81	1322.48
ESN + Linear Regression	C	3.57	3.38	4.24	1.90	4.31
ESN + WiSARD	C	55.40	54.52	59.71	28.44	65.12
Time Resample + WiSARD	C	0.31	0.04	0.17	0.61	0.76
Length Resample + WiSARD	C	0.83	0.09	0.30	0.47	0.95

experiment (A, B, C, D) and the samples for participant X on day Y are represented as X_Y. Participant A was the only one with two sets of samples on day 1, these are differentiated as A_1^1 and A_1^2. Following the evaluation procedure described in the authors paper [1], instances 1, 4 and 6 were randomly split 25 times in training and testing sets. For each pair, training and testing occurred and their results calculated (mean recognition rate, standard deviation, median recognition rate and max recognition rate). The remaining instance sets, (2, 3), (5) and (7, 8, 9), were used to further evaluate the best models found over all 25 iterations of, respectively, instances 1, 4 and 6. The original article selects only the best model to be evaluated and therefore did not provide average and median information. In this work, however, all trained models were used and these values calculated.

To facilitate comparison, the authors best results on this data set are replicated

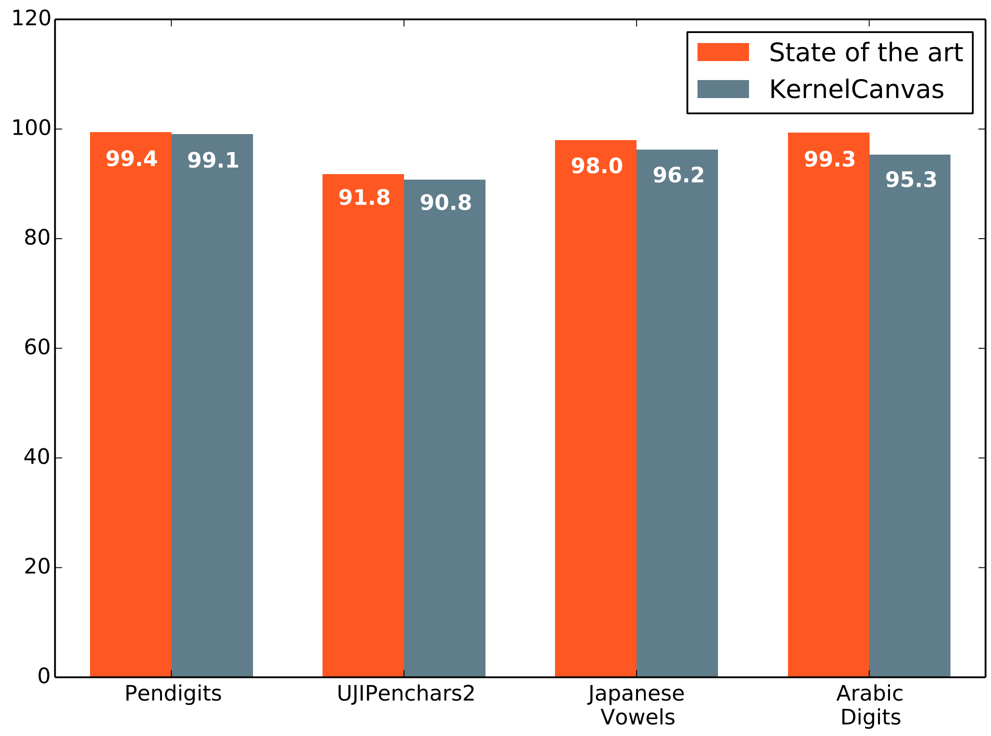


Figure 4.3: Results of the proposed method on standard tasks compared against to the best results found in literature.

Table 4.4: KernelCanvas results for Libras standard tasks (%)

instance id	**people id.**	**mean recognition rate**	**standard deviation**	**median recognition rate**	**max recognition rate**
1	A_1^1, A_2	0.98	0.02	0.96	1.00
2	A_2	0.99	0.01	0.98	1.00
3	A_1^1	0.99	0.01	1.00	1.00
4	A_2	0.98	0.03	1.00	1.00
5	A_1^1	0.83	0.01	0.82	0.84
6	A_2, B_2, D_2	0.98	0.01	0.99	1.00
7	A_2	0.99	0.01	0.99	1.00
8	A_1^1	0.82	0.03	0.87	0.87
9	A_1^2, B_1, C_1	0.81	0.03	0.78	0.87

in Table 4.5. The column ”max recognition rate” from his table, however, was moved to ”mean recognition rate” on instances (2,3,5,7,8,9). As stated before, in order to discard invalid models and avoid overfitting the author only evaluated the best network created over the 25 iterations. As these characteristics are not present

Table 4.5: Results of the best model on the paper of the data set author - Unsupervised Fuzzy-LVQ [1] (%)

instance id	**people id.**	**mean recognition rate**	**standard deviation**	**median recognition rate**	**max recognition rate**
1	A_1^1, A_2	0.94	0.19	0.94	0.96
2	A_2	0.96	-	-	-
3	A_1^1	0.98	-	-	-
4	A_2	0.95	0.18	0.94	0.99
5	A_1^1	0.49	-	-	-
6	A_2, B_2, D_2	0.88	0.21	0.89	0.92
7	A_2	0.89	-	-	-
8	A_1^1	0.56	-	-	-
9	A_1^2, B_1, C_1	0.58	-	-	-

on WiSARD, all trained models were used on these instances and the average and standard deviation were collected. Comparatively, our results performed better than all the results presented by the author.

4.5 Partial Conclusion

This chapter presented a series of experiments regarding the proposed methodology in addition to experiments involving the WiSARD neural network and the ESN model. Furthermore, an empirical evaluation of the best kernel sampling algorithm along with an estimation of the number of kernels on the considered data sets were also provided.

As observed, the configuration required for the proposed model does not differ much from each data set, making it easy to configure and use the model. When compared to simple approaches, such as time and length resampling, this model obtained considerable improved results at the additional cost of nearly 10 ms. The only equiparable model observed was the KNN with DTW which despite the slightly improved results also required considerably much more processing time. Thus, the overall result of the proposed methodology was considered satisfactory as it was able to surpass most comparison models on almost all data sets considered. When compared towards the state of the art results, it was seen that this model achieved nearly the same results on some tasks and competitive results on others. In addition to this, is was also observed that the WiSARD network provides a good alternative for the Linear Regression algorithm usually used as output layer of the ESN. The only issue that must be dealt, however, is that it also increased the training and classification times.

On top of this, the three kernel sampling algorithms were compared and the obtained results indicated that the best option on all data sets was the Random Sampling algorithm, which was also the simplest of the three algorithms. Finally, is was estimated that 1024 and 2048 kernels are usually enough for the tasks considered. Despite this, even when fewer kernels were used the model could provide average results and linearly decrease the computational time, possibly presenting a good alternative for low end devices.

Chapter 5

Conclusion

In this work a new architecture for time series classification in conjunction with the WiSARD network was proposed. The proposed architecture, named KernelCanvas, is able to convert a stroke of points from the observed space into a fixed point in a much higher binary dimensional space. This representation, in turn, fits remarkably well when combined with weightless neural networks as these neural networks are capable of easily processing large input patterns and additionally requires them to be in binary format.

At the essence of this new architecture, kernels had to be randomly sampled in order to represent different possible characteristics of the input patterns and three sampling algorithm were considered. Despite the extra care to reduce the number of kernel collisions and to better distribute them in space with the use of Poisson Disk samples, it was empirically observed that the simplest and purely random approach also obtained the best results.

When the methodology was compared to related approaches, such as ESN, KNN and basic resamplings, it presented superior results to most of them and at worst competitive results when compared to KNN with DTW. When compared to KNN, however, the proposed methodology represented a considerably reduction on the computational power required and easily qualifies for real-time applications.

When compared to state of the art results, the proposed model obtained nearly the same results on Pendigits data set, and competitive results on all others. When considering the nine tasks proposed by the authors of the Libras data set, though, it obtained improved results on all tasks if compared to the results presented on the authors paper.

Furthermore, all the initial proposals were completely achieved. The proposed methodology kept all the main characteristics of the WiSARD neural network, including incremental learning, simplicity on its operations and low training and classification times. The only consideration observed when this model was used is that it requires a little of knowledge about the type of data being used. This is a required information as there is still in order to correctly choose the preprocessing chain used and resize the input signal to fit the size of the canvas.

5.1 Future Works

In addition to all that has been proposed and evaluated, many other connections are still open for further investigation in the future. One of them, for instance, is the generation of kernels based on samples from the training set. Each kernel, in essence, represents a characteristic of the signal to be classified. Thus, an algorithm to identify this type of information during the training phase could possibly increase the model confidence or reduce the number of kernels and speed up classification time. This approach, however, would create kernels specific to the samples and classes present on the training set, losing the incremental learning property.

Additionally, auxiliary structures to query approximate nearest neighbors, could also be used to speed up the nearest kernels search. These could present a significant gain in speed but the impact of this approximation would need to be measured on the final accuracy.

In respect to the integration between ESN and WiSARD, it was observed that the WiSARD network successfully improved classification results of the ESN with the traditional Linear Regression algorithm. This improvement, however, also increased the training and classification times of the model but also brought incremental learning capability to the model. Following this observation, it is conjectured that similar gains could be achieved on analogous models, such as Extreme Learning Machines.

Another characteristic still not investigated in this work is the integration of the proposed model with the Hidden Markov Model (HMM) [4]. Recently, we have successfully obtained a real time music tracker by mixing the WiSARD network and the Markov Localization Algorithm [6]. Although this work involved slightly different objectives, it is possible that many of the techniques used could also be applied to improve the model proposed in this work. If successful, a comparison of the proposed technique, HMM, and a hybrid Markov-WiSARD model would certainly produce interesting characteristics.

Finally, with the recent advances in artificial intelligence and improved computational power on portable devices, we have seen many applications changing the way we interact with machines and the type of data they can collect. Some of the applications involving time series include gesture keyboards on touch screen devices, practical voice recognition, music identification, wearable devices that keep track of the user fitness activities and health status, among others. With all this in mind, it would be really interesting to see a few practical applications of the methodology here described.

Bibliography

[1] DIAS, D. B., MADEO, R. C., ROCHA, T., et al. "Hand movement recognition for brazilian sign language: a study using distance-based neural networks". In: *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, pp. 697–704. IEEE, 2009.

[2] COVER, T., HART, P. "Nearest neighbor pattern classification", *Information Theory, IEEE Transactions on*, v. 13, n. 1, pp. 21–27, 1967.

[3] MÜLLER, M. "Dynamic Time Warping". In: *Information Retrieval for Music and Motion*, Springer Berlin Heidelberg, pp. 69–84, 2007. ISBN: 978-3-540-74047-6. doi: 10.1007/978-3-540-74048-3_4.

[4] RABINER, L. "A tutorial on hidden Markov models and selected applications in speech recognition", *Proceedings of the IEEE*, v. 77, n. 2, pp. 257–286, Feb 1989. ISSN: 0018-9219. doi: 10.1109/5.18626.

[5] HINTON, G. E., OSINDERO, S., TEH, Y.-W. "A fast learning algorithm for deep belief nets", *Neural computation*, v. 18, n. 7, pp. 1527–1554, 2006.

[6] SOUZA, D. F., FRANCA, F. M., LIMA, P. M. "Real-Time Music Tracking Based on a Weightless Neural Network". In: *Complex, Intelligent, and Software Intensive Systems (CISIS), 2015 Ninth International Conference on*, pp. 64–69. IEEE, 2015.

[7] DE SOUZA, D. F. P., FRANCA, F. M. G., LIMA, P. M. V. "Spatio-temporal Pattern Classification with KernelCanvas and WiSARD". In: *2014 Brazilian Conference on Intelligent Systems*, pp. 228–233. IEEE, out. 2014.

[8] CARDOSO, D., CARVALHO, D., ALVES, D., et al. "Credit analysis with a clustering RAM-based neural classifier", *European Symposium on Artificial Neural Networks*, , n. November 2015, pp. 517–522, 2014.

[9] DE SOUZA, D. F., CARNEIRO, H. C., FRANCA, F. M., et al. "Rock-Paper-Scissors WiSARD". In: *2013 BRICS Congress on Computational Intel-*

ligence and 11th Brazilian Congress on Computational Intelligence, pp. 178–182. IEEE, set. 2013.

[10] ALIMOGLU, F., ALPAYDIN, E. “Methods of Combining Multiple Classifiers Based on Different Representations for Pen-based Handwritten Digit Recognition”. In: *Proceedings of the Fifth Turkish Artificial Intelligence and Artificial Neural Networks Symposium (TAINN)*, 1996.

[11] JAEGER, H. “The “ echo state ” approach to analysing and training recurrent neural networks – with an Erratum note 1”, *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, , n. 148, pp. 1–47, 2010. ISSN: 18735223.

[12] XI, X., KEOGH, E., SHELTON, C., et al. “Fast time series classification using numerosity reduction”. In: *Proceedings of the 23rd international conference on Machine learning (ICML)*, pp. 1033—-1040, 2006. ISBN: 1595933832. doi: 10.1145/1143844.1143974.

[13] BENTLEY, J. L. “Multidimensional binary search trees used for associative searching”, *Communications of the ACM*, v. 18, n. 9, pp. 509–517, 1975.

[14] SAKOE, H., CHIBA, S. “Dynamic Programming Algorithm Optimization for Spoken Word Recognition”, *IEEE Transactions on Acoustics, Speech and Signal Processing*, v. 26, n. 1, pp. 43–49, 1978.

[15] ITAKURA, F. “Minimum Prediction Residual Principle Applied to Speech Recognition”, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, v. 23, n. 1, pp. 67–72, 1975.

[16] ALEKSANDER, I., THOMAS, W. V., BOWDEN, P. A. “WISARD: A Radical Step Foward in Image Recognition”, *Sensor Review*, v. 4, pp. 120–124, July 1984.

[17] HAMMING, R. “Error detecting and error correcting codes”. In: *Bell System Technical Journal*, v. 29, pp. 147–160, 1950.

[18] AURENHAMMER, F. “Voronoi diagrams — A survey of a fundamental data structure”, *ACM Computing Surveys*, v. 23, n. 3, pp. 345–405, 1991. ISSN: 03600300. doi: 10.1145/116873.116880.

[19] BRIDSON, R. “Fast Poisson Disk Sampling in Arbitrary Dimensions”. In: *ACM SIGGRAPH 2007 Sketches*, SIGGRAPH ’07, New York, NY, USA, 2007. ACM. doi: 10.1145/1278780.1278807.

[20] COOK, R. L. “Stochastic sampling in computer graphics”, *ACM Transactions on Graphics*, v. 5, n. 1, pp. 51–72, 1986. ISSN: 07300301. doi: 10.1145/7529.8927.

[21] MITCHELL, D. P. “Generating Antialiased Images at Low Sampling Densities”, *SIGGRAPH Comput. Graph.*, v. 21, n. 4, pp. 65–72, ago. 1987. ISSN: 0097-8930. doi: 10.1145/37402.37410.

[22] MITCHELL, D. P. “Spectrally optimal sampling for distribution ray tracing”, *ACM SIGGRAPH Computer Graphics*, v. 25, n. 4, pp. 157–164, 1991. ISSN: 00978930. doi: 10.1145/127719.122736.

[23] WEI, L.-Y. “Parallel Poisson disk sampling”, *ACM Transactions on Graphics*, v. 27, n. 3, pp. 1, 2008. ISSN: 07300301. doi: 10.1145/1399504.1360619.

[24] YING, X., LI, Z., HE, Y. “A parallel algorithm for improving the maximal property of Poisson disk sampling”, *Computer-Aided Design*, v. 46, pp. 37 – 44, 2014. ISSN: 0010-4485.

[25] DUNBAR, D., HUMPHREYS, G. “A spatial data structure for fast Poisson-disk sample generation”, *ACM Transactions on Graphics*, v. 25, n. 3, pp. 503, 2006. ISSN: 07300301. doi: 10.1145/1141911.1141915.

[26] DUHAMEL, P., VETTERLI, M. “Fast Fourier transforms: a tutorial review and a state of the art”, *Signal processing*, v. 19, n. 4, pp. 259–299, 1990.

[27] COOLEY, J. W., TUKEY, J. W. “An algorithm for the machine calculation of complex Fourier series”, *Mathematics of computation*, v. 19, n. 90, pp. 297–301, 1965.

[28] DAVIS, S. B., MERMELSTEIN, P. “Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences”, *Acoustics, Speech and Signal Processing, IEEE Transactions on*, v. 28, n. 4, pp. 357–366, 1980.

[29] DASH, K., PADHI, D., PANDA, B., et al. “Speaker Identification using Mel Frequency Cepstral Coefficient and BPNN”, *International Journal of Advanced Research in Computer Science and Software Engineering*, v. 2, n. 4, 2012.

[30] HUANG, X., ACERO, A., HON, H.-W., et al. *Spoken language processing: A guide to theory, algorithm, and system development.* Upper Saddle River, New Jersey, Prentice Hall PTR, 2001.

[31] ANTONIOL, G., ROLLO, V. F., VENTURI, G. “Linear predictive coding and cepstrum coefficients for mining time variant information from software repositories”. In: *ACM SIGSOFT Software Engineering Notes*, v. 30, pp. 1–5. ACM, 2005.

[32] LLORENS, D., PRAT, F., MARZAL, A., et al. “The UJIpenchars Database : A Pen-Based Database of Isolated Handwritten Characters”. pp. 2647–2651, 2008. ISBN: 2951740840.

[33] KUDO, M., TOYAMA, J., SHIMBO, M. “Multidimensional curve classification using passing-through regions”, *Pattern Recognition Letters*, v. 20, n. 11, pp. 1103–1111, 1999.

[34] HAMMAMI, N., BEDDA, M. “Improved tree model for Arabic speech recognition”. In: *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, v. 5, pp. 521–526. IEEE, 2010.

[35] CASTRO-BLEDA, M. J., ESPAFIA, S., GORBE, J., et al. “Improving a DTW-based recognition engine for on-line handwritten characters by using MLPs”. In: *Document Analysis and Recognition, 2009. ICDAR’09. 10th International Conference on*, pp. 1260–1264. IEEE, 2009.

[36] RATANAMAHATANA, C. A., KEOGH, E. “Three myths about dynamic time warping data mining”. In: *Proceedings of SIAM International Conference on Data Mining (SDM’05)*, pp. 506–510. SIAM, 2005.

[37] CHATZIS, S. P., KOSMOPOULOS, D. I. “A variational Bayesian methodology for hidden Markov models utilizing Student’s-t mixtures”, *Pattern Recognition*, v. 44, n. 2, pp. 295–306, 2011.

[38] HAMMAMI, N., BEDDA, M., FARAH, N. “Spoken Arabic Digits recognition using MFCC based on GMM”. In: *Sustainable Utilization and Development in Engineering and Technology (STUDENT), 2012 IEEE Conference on*, pp. 160–163. IEEE, 2012.

Appendix A

Published Works

The following works were published during the development of this dissertation.

- Diego F. P. de Souza, Felipe M. G. França, Priscila M. V. Lima. Real-Time Music Tracking Based on a Weightless Neural Network. In proceedings of International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS), IEEE, Blumenau, 2015.

- Diego F. P. de Souza, Felipe M. G. França, Priscila M. V. Lima. Spatio-temporal Pattern Classification with KernelCanvas and WiSARD. In proceedings of Brazilian Conference on Intelligent Systems (BRACIS), pp. 228–233. IEEE, October 2014.

- Douglas O. Cardoso, Danilo S. Carvalho, Daniel S. F. Alves, Diego F. P. de Souza, Hugo C. C. Carneiro, Carlos E. Pedreira, Priscila M. V. Lima, Felipe M. G. França. Credit Analysis with a clustering RAM-based neural classifier. In Proceedings of European Symposium on Artificial Neural Networks (ESANN), Computational Intelligence and Machine Learning, Bruges, p. 517-522, 2014.

- Diego F. P. de Souza, Felipe M. G. França, Priscila M. V. Lima. Rock-Paper-Scissors WiSARD. In proceedings of 2013 BRICS Congress on Computational Intelligence (BRICS-CCI) and 11th Brazilian Congress on Computational Intelligence (CBIC), pp. 178–182. IEEE, September 2013.

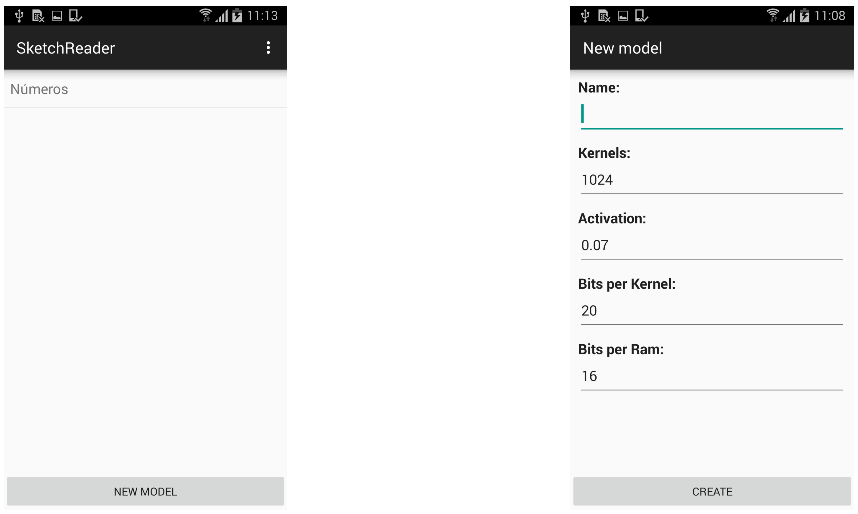
Appendix B

SketchReader

In order to easily demonstrate the functionality of the proposed methodology, a small Android application was also developed and released. This application contains some of the main characteristics of the model such as fast training and classification times and creation of new classes during its execution. Part of this implementation, however, is not syncronized with the most recent implementation version of the methodology. As a result of this, it is not possible to change the kernel sampling algorithm, for instance.

The application interface is organized as follows. The initial screen, shown in Figure B.1a, contains the list of all models already created, in this case only one model named "Números". If the user wants to create a new model it must click the button with caption "NEW MODEL" and set or confirm the model parameters, shown in Figure B.1b. The first parameter, the model's name, is used just to represent the model on the list. The next three, "Kernels", "Activation" and "Bits per Kernel", are the three parameters of the KernelCanvas, as described in Chapter 3. The last parameter, called "Bits per Ram", correponds to the number of bits entering each RAM unit on the WiSARD network. The number of input bits entering the RAM does not need to be set as it is equal to the numbers of "Kernels" multiplied by "Bits per Kernel". It is also not necessary to define the number of classes as the application dynamically creates new discriminators when required. When the model is created its name appears inside the list on the initial screen. Clicking on the name of a model shows the main interface, where patterns may be entered for either training or classification.

Initially the model does contain any knowledge and in order to make classifications it needs to be trained. To train a new pattern, such as the number 4 drawed in Figure B.2a, its stroke needs to be drawed over the drawing surface and later the button "Train" pressed. When this button is pressed, the dialog shown in Figure B.2b is opened and a new class may be added. In order to create a new class, the new class name must be typed on the text field and the plus sign clicked. To



Números

Name:

Kernels:
1024

Activation:
0.07

Bits per Kernel:
20

Bits per Ram:
16

(a) (b)

Figure B.1: Initial interfaces showing (a) the list of models created and (b) the create new model screen.

train a class that already exists it is only necessary to click on the corresponding class name.

Classification is achieved similarly, the user draws the input pattern over the drawing surface but then it must press the button ”Guess”, on bottom of the Main interface. This will make the application display a dialog similar to that in Figure B.2c. This dialog shows the class of the corresponding prediction and the confidence of the WiSARD model in parenthesis. If the model guessed correctly, the user may press the button ”Yes” and the drawing surface will be cleared ant this pattern will be used to reinforce the model knowledge. If the button ”No” is pressed, the main interface is shown again and the user may press the button ”Train” and select the correct class.

These are the main usage instructions of the application. Pressing the Android back button, however, will also prompt the user if the application should persist the changes made to the model. If the program receives a positive answer the changes are persisted, otherwise all modifications are lost. This is usefull an usefull option if, e.g., the user has trained the model with a wrong pattern and would want to restore to the previous state.

The application is named SketchReader and is available for free at the Google Play Store[1]. It is extremelly probable that this application will remain the same as currently described, however, plans to release this approach as a library are not eliminated. Therefore, further improvements and modifications might also appear in the future.

[1]Available at: https://play.google.com/store/apps/details?id=com.github.diegofps.sketchreader

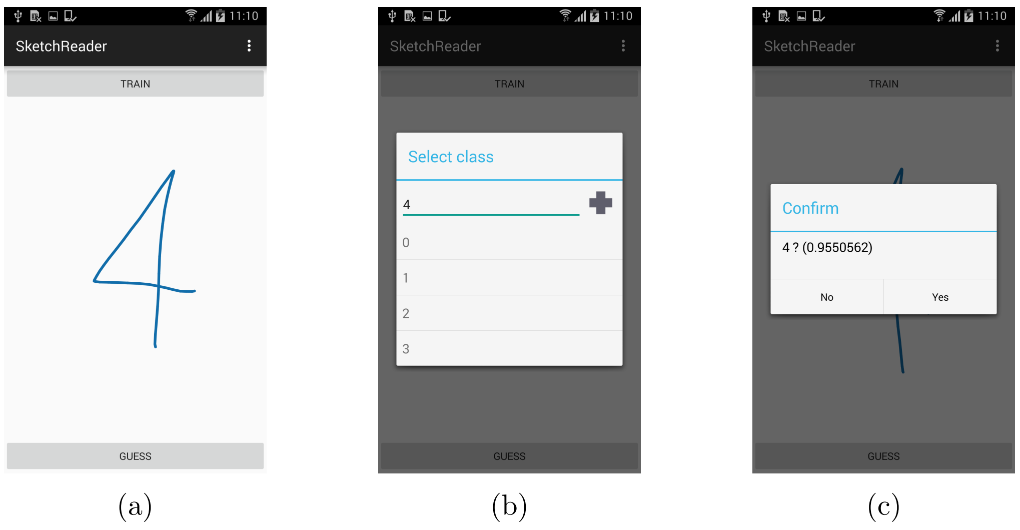


Figure B.2: Main interfaces containing (a) the canvas where the user may input the pattern, (b) the dialog to tell the model which class the current pattern belongs and (c) the model answer to a given pattern.