Interpretable time series classification

Yichang Wang

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES 1
COMUE UNIVERSITÉ BRETAGNE LOIRE

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par
Yichang WANG

Interpretable time series classification

Thèse présentée et soutenue à Rennes, le 20/09/2021
Unité de recherche : Inria/IRISA (UMR 6074)
Thèse N° :

Rapporteurs avant soutenance :

Miguel COUCEIRO Full Professor – Université de Lorraine, France
Céline ROBARDET Full Professor – Institut National des Sciences Appliquées de Lyon, France

Composition du Jury :

Président :	Alexandre TERMIER	Full Professor – Université de Rennes 1, France
Examinateurs :	Mikaela KELLER	Associate Professor – Université de Lille, France
	Vincent LEMAIRE	Senior researcher, HDR – Orange Labs, France
Dir. de thèse :	Élisa FROMONT	Full Professor – Université de Rennes 1, France
Co-encadrants :	Rémi EMONET	Associate Professor – Université Jean Monnet, Saint-Etienne, France
	Romain TAVENARD	Associate Professor, HDR – Université de Rennes 2, France

Invité(s) :

Simon MALINOWSKI Associate Professor – Université de Rennes 1, France

To my parents, Xia Gao and Naijie Wang,

for their enormous love and support.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my esteemed supervisors, Elisa Fromont, Romain Tavenard, and Rémi Emonet. My supervisors trusted my potential when I wanted to change my field of study from physics to computer science, and they taught me how to do research with a lot of patience. During my PhD study in Rennes, I was educated and benefited from their rich perspectives and sharp insights. I would not have completed this research work without their patient guidance, invaluable advice, constant encouragement, and timely support. I take this opportunity to thank my collaborator, Simon Malinowski, for the many inspiring research conversations.

I would like to heartily thank Miguel Couceiro and Céline Robardet for evaluating my thesis and providing very relevant revision advice, and Mikaela Keller and Vincent Lemaire for their feedback during my defense. Special thanks to Guillaume Gravier and Vincent Lemaire for being members of my thesis committee and for providing advice during my thesis.

I am very lucky to work in a friendly and pleasant environment within LACODAM research team. Many thanks to the PhDs who graduated before me, Yann Dauxais, Clément Gautrais, and Maël Guillemé, for being present at my PhD defense and for being supportive, Kevin Fauvel, Colin Leverger, and Raphaël Gauthier, who shared a "time series office" with me; Johanne Bakalara, Grégory Martin, Julien Delaunay, Camille-Sovanneary Gauthier, Thomas Guyet, Romaric Gaudel and Luis Galárraga, with whom I interacted during my day-to-day research life. Many thanks to the interns who worked in our team, Muaz Twaty, Théo Losekoot, Issei Harada, and Guillaume Latour, and especially to Loïc Mosser and Etienne Ménager, who have made a preliminary implementation of the main contribution in this thesis. And, last but not least, to Heng Zhang, my best friend in Rennes, I will always remember the moments when we ate hot pots and dumplings together.

I would like to thank all the staff members at Inria/IRISA laboratory. I sincerely thank our team assistants, Marie-Noëlle Georgeault and Gaëlle Tworkowski, for helping me with many administration processes, and Guillermo Andrade Barroso and other members in SED technical support team, for making IGRIDA computing platform easy to use.

Since my childhood, I have always been interested in computer science, and my career path in computer science is influenced by the spirit of my friend Samuel Guizani in Orléans: he would do anything in pursuit of writing good code. During my study in Orléans, I was also helped a lot by Guillaume Munier, Olivier Nogues, Ilan Robin, Paul Cardera, Yue Zhao, Mingxu Xing, Chenyu Yue, Junda Liu, Zuokun Ouyang, Guanglie Ouyang, Dubo Huang, and Salomé Renoult. I can't thank you all enough!

Finally, my gratitude goes to my girlfriend Xiang Gu. She has been thoughtful and supportive during the writing of my thesis. I also want to thank my parents Xia Gao and Naijie Wang, to whom this thesis is dedicated.

Yichang

Table of Contents

Résumé en Français

L'intelligence artificielle (IA) est un domaine scientifique prenant racine à la fin de la Seconde Guerre mondiale (Russell et al. 2009). À ses débuts, les recherches dans le domaine de l'intelligence artificielle se sont concentrées sur les algorithmes de jeu et les mécanismes de raisonnement. Ses succès (et échecs) n'impactaient pas particulièrement le monde. Depuis les succès très médiatisés de l'apprentissage profond à partir de 2012 (Krizhevsky et al. 2012; Silver et al. 2016; Savage 2019; Lan et al. 2020), des questions éthiques et juridiques ont été soulevées et ont donné naissance à un nouveau domaine de recherche aujourd'hui très populaire : *l'intelligence artificielle explicable* (Adadi et al. 2018)).

Au contraire des algorithmes de programmation traditionnels, utilisant les instructions de manière explicite, les algorithmes d'apprentissage automatique (et en particulier les algorithmes d'apprentissage profond) apprennent des modèles directement à partir des données.

Les modèles d'apprentissage automatique sont généralement des fonctions mathématiques qui transforment des données d'entrée en une sortie attendue (dans le cas de l'apprentissage supervisé, la sortie attendue est explicitement donnée pendant l'apprentissage de la fonction). L'algorithme d'apprentissage vise à estimer les meilleurs paramètres pour cette fonction. L'une des méthodes d'apprentissage automatique les plus populaires, et la plus performante de nos jours, est appelée apprentissage profond (Goodfellow et al. 2016). Cependant, les modèles récents d'apprentissage profond (par exemple Krizhevsky et al. (2012)) possèdent des millions de paramètres et les raisons pour lesquelles ces modèles prennent automatiquement certaines décisions, sont devenues opaques. Ces modèles complexes, dont la correspondance entre les données d'entrée et la sortie attendue (que nous appelons "décision" ou "prédiction") n'est pas suffisamment explicite pour être compréhensible par des humains, sont appelés modèles "boîtes noires".

L'utilisation de modèles "boîtes noires" pour des applications sensibles (par exemple dans le droit, la médecine, l'assurance, la conduite autonome (Guidotti et al. 2018)), quel que soit leur performance, est la principale raison du questionnement mentionné au début de cette introduction. Au sein de l'Union Européenne par exemple, de nouvelles

lois ont été votées imposant un “droit à l’explication” dans le Règlement général sur la protection des données (RGPD) [1]. Aux États-Unis d’Amérique, le programme de recherche “eXplainable AI (XAI)” de la DARPA [2] souligne également l’importance de ce domaine de recherche.

Les explications les plus intuitives pour un modèle “boîte noire” peuvent prendre plusieurs formes en fonction du public visé. Par exemple, elle peuvent prendre la forme d’informations que nous percevons dans la vie de tous les jours comme les informations visuelles (images), le langage humain (vocabulaires, sons), les odeurs, etc. Pour un public plus expert les informations peuvent être également liées à la spécialité de tels experts, par exemple des séries temporelles de température pour des médecins, des séries temporelles de puissance électrique pour des fournisseurs d’électricité, etc. Grâce à ces explications, il est possible de comprendre les mécanismes de décision d’une boite noire.

Dans cette thèse, nous nous concentrons sur un type de données particulièrement omniprésent : les séries temporelles. Une série temporelle est une collection finie de données réelles indexées par le temps. C’est l’un des types de données les plus courants dans la vie réelle et il modélise de nombreux processus tels que l’évolution de la bourse, la consommation quotidienne d’énergie, l’évolution de la température corporelle d’un patient particulier, etc. (Shumway et al. 2005).

L’analyse des séries temporelles consiste à extraire des caractéristiques statistiques et significatives de ces séries temporelles et à prendre des décisions sur la base de ces caractéristiques. Certains problèmes d’analyse de séries temporelles, tels que la prévision (*e.g.*, la prédiction de la ou des prochaines valeurs d’une série donnée), la régression (l’association d’une valeur numérique à une série temporelle donnée), le *clustering* (le regroupement des séries temporelles en fonction de leurs similarités), la détection d’anomalies (la recherche de séries temporelles qui se comportent différemment des autres ou la recherche de valeurs surprenantes dans une série donnée), et la classification (la classification d’une série temporelle dans un ensemble de classes d’intérêt prédéfinies), peuvent être abordés en utilisant des méthodes d’apprentissage automatique. Dans cette thèse, nous nous concentrons sur le problème particulier de la classification des séries temporelles.

Nous utilisons des données de référence concernant la classification des séries temporelles issues du dépôt de l’UCR collectées par l’Université de Californie Riverside (Chen

1. `https://gdpr-info.eu/recitals/no-71/`
2. `https://www.darpa.mil/program/explainable-artificial-intelligence`

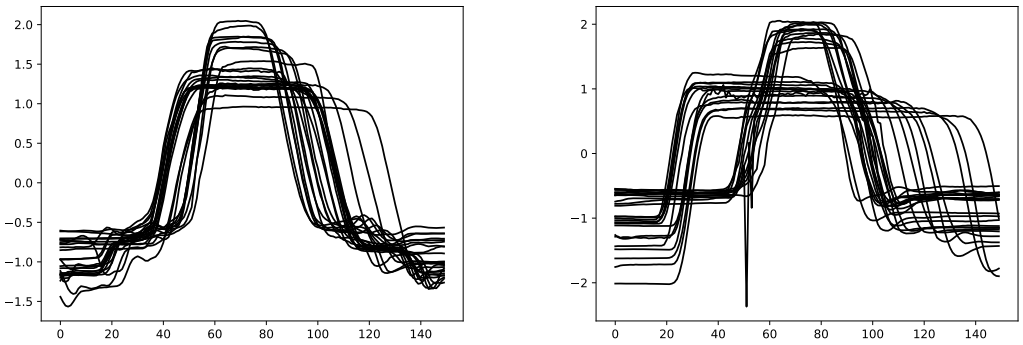


Figure 1 – Exemple de séries temporelles d'entraînement provenant du jeu de données de séries temporelles *GunPoint* pour la classe "GunDraw" (à gauche) et la classe "Point" (à droite) du dépôt de l'UCR. (Chen et al. 2015).

et al. 2015). Le nombre de jeux de données de cette archive ayant augmenté avec le temps, ce dépôt s'appelle maintenant "UEA & UCR time series classification repository" et contient 128 jeux de données de séries temporelles univariées et 30 jeux de données multivariées (disponibles sur `http://www.timeseriesclassification.com/`). Nous n'utilisons que les 85 jeux de données de l'UCR (Chen et al. 2015), car il s'agit des références les plus utilisées dans le communauté du domaine de la classification des séries temporelles. Ces 85 ensembles de données de séries temporelles univariées sont collectés à partir de différentes sources de données, tels que des ECGs (électrocardiogrammes), des images, des capteurs de mouvement, des exemples synthétiques, des spectrographes, etc. Ces jeux de données sont de taille modérée, mais peuvent varier d'une taille allant de 16 instances d'entraînement (pour le jeu de données *DiatomSizeReduction*) à 8926 (*ElectricDevices*), d'une longueur allant de 24 points (*ItalyPowerDemand*) à 2709 points (*HandOutlines*), et d'un nombre de classes allant de 2 (*GunPoint*) à 60 classes (*ShapesAll*).

La Figure 1 montre un célèbre jeu de données de séries temporelles à 2 classes appelé GunPoint, ayant été collecté à partir des mouvements de la main de deux acteurs. Il s'agit de séries temporelles pour la classe "GunDraw" (l'acteur sort une arme d'un étui monté sur la hanche, la pointe vers une cible pendant environ 1 seconde et la remet en place) et "NoGun" (l'acteur agit avec ses doigts).

De nombreuses méthodes ont été développées pour la classification des séries temporelles (voir par exemple (Bagnall et al. 2017; Tavenard et al. 2017; Lods et al. 2017; Guillemé et al. 2019a; Fauvel et al. 2019)). Nous classons les méthodes existantes en quatre groupes : les méthodes basées sur la distance, l'extraction de caractéristiques, les

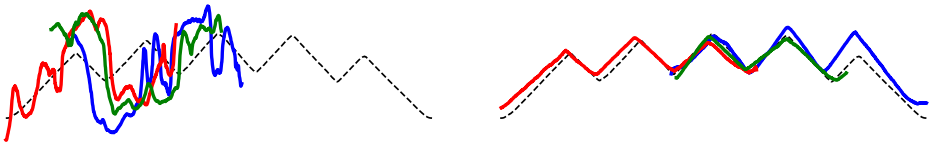


Figure 2 – Exemple de série temporelle de test (lignes noires en pointillées) et trois *shapelets* les plus discriminantes utilisées pour sa classification pour un modèle de base (Grabocka et al. 2014) (à gauche) et pour notre modèle "XCNN" proposé (à droite) sur le jeu de données "HandOutlines" (du dépôt de l'UCR). (Chen et al. 2015)).

réseaux neuronaux et les méthodes ensemblistes. Les justifications de ces catégories ainsi qu'une revue de chaque méthode sont présentées dans le Chapitre 1.

Une catégorie de méthodes très étudiée consiste à trouver des sous-séquences discriminantes indépendantes d'indexes temporels, appelées *shapelets*, qui peuvent être utilisées pour classer ou décrire la série. Les *shapelets* sont des sous-séquences, i.e., des éléments visuels qui sont utiles pour expliquer les décisions prises par un modèle appris à partir de données de séries temporelles. Elles sont donc de bonnes candidates pour s'attaquer à notre problème d'interprétabilité de la classification de séries temporelles. Dans les premiers articles sur la classification des séries temporelles basée sur les *shapelets* (Ye et al. 2009; Rakthanmanon et al. 2013), les *shapelets* étaient directement extraites de l'ensemble d'apprentissage et les *shapelets* sélectionnées pouvaient ensuite être utilisées pour expliquer la décision du classificateur. Cependant, les processus d'énumération et de sélection des *shapelets* étaient soit très coûteux, soit peu coûteux mais ne donnant pas de bonnes performances (comme indiqué dans la Section 1.3.2). L'apprentissage conjoint d'une représentation basée sur les *shapelets* des séries du jeu de données et la classification des séries en fonction de cette représentation (Lines et al. 2012; Grabocka et al. 2014) permet d'obtenir des *shapelets* discriminantes de manière beaucoup plus efficace. Un exemple de telles *shapelets* obtenues avec la méthode de Grabocka et al. (2014), est donné dans la Figure 2 (à gauche). Cependant, si les *shapelets* apprises sont bien discriminantes, elles sont souvent différentes (visuellement) des morceaux réels d'une série temporelle du jeu de données. En tant que telles, ces *shapelets* peuvent ne pas être adaptées pour expliquer la décision d'un classificateur particulier puisqu'elles ne constituent pas des éléments de langage compréhensibles par un expert, ou mieux, par un utilisateur lambda.

Dans cette thèse, nous abordons le problème de la classification des séries temporelles interprétables. Dans le Chapitre 1, nous présentons les méthodes existantes de classification de séries temporelles, en particulier les méthodes basées sur les *shapelets* et les réseaux

de neurones. Dans le Chapitre 2, nous clarifions la notion d'interprétabilité d'un modèle : un modèle est interprétable tant qu'il peut être compris par un humain. En fonction du processus d'apprentissage du modèle, l'interprétabilité d'un modèle peut être considérée comme *ad-hoc*, *in-situ*, et *post-hoc*. Nous discutons de l'importance de l'interprétabilité et passons en revue l'interprétabilité de différents modèles d'apprentissage automatique sous différents angles, i.e., une interprétabilité au niveau global, au niveau modulaire et au niveau local. Pour les modèles non interprétables, nous cherchons une interprétabilité *post-hoc* pour donner une explication particulière. Dans le Chapitre 3, nous adaptons le cadre proposé par Fauvel et al. (2020b) pour évaluer l'interprétabilité de la classification de séries temporelles. Ce cadre fournit des informations sur l'interprétabilité du modèle et évalue la qualité des explications. Dans le Chapitre 4, nous proposons une méthode appelée "eXplainable Convolutional Neural Network" (XCNN). Notre méthode apprend des *shapelets* contraintes à être proches des sous-séquences de la série temporelle d'apprentissage. Notre XCNN est interprétable *in-situ* à un niveau modulaire et produit des modèles discriminants de séries temporelles. A la fin de ce document, nous fournissons une conclusion concernant ce travail ainsi que quelques perspectives.

Publications

Le travail présenté dans cette thèse, a été publié dans les conférences suivantes :

1. Yichang Wang, Rémi Emonet, Elisa Fromont, Simon Malinowski and Romain Tavenard. *Adversarial Regularization for Explainable-by-Design Time Series Classification*, ICTAI'2020 32th International Conference on Tools with Artificial Intelligence, p1079-1087, Virtual
2. Yichang Wang, Rémi Emonet, Elisa Fromont, Simon Malinowski, Etienne Menager, Lo ic Mosser, et Romain Tavenard, *Classification de séries temporelles basée sur des shapelets interprétables par réseaux de neurones antagonistes*, CAp'2019 (Conférence d'Apprentissage), Toulouse, 2019
3. Yichang Wang. *Explainable Time Series Classification*, Workshop "Fair and Explainable Models" at the the 31th EURO Conference (European Conference on Operational Research), Virtual 2021

INTRODUCTION

Artificial Intelligence (AI) is a science field which dates back to the end of the second world war (Russell et al. 2009). At its early stage, AI was focusing on game playing algorithms and reasoning mechanisms and its successes (and failures) did not particularly threaten the world. Since the very high-media profile successes of deep learning which have started in 2012 (Krizhevsky et al. 2012; Silver et al. 2016; Savage 2019; Lan et al. 2020), ethical and legal questions have started to be raised and have given birth to a very recent and yet nowadays extremely popular new field of research: eXplainable Artificial Intelligence (XAI) (Adadi et al. 2018).

Different from traditional programming algorithms, which use instructions explicitly, machine learning algorithms (and in particular deep learning ones) *learn* models directly from data. Machine learning models are usually mathematical functions that map the input data into an expected output (in case of supervised learning, the output is explicitly given while learning the function). The learning algorithm aims at estimating the best parameters for this function. One of the most popular and successful machine learning method is called deep learning (Goodfellow et al. 2016). However, recent deep learning models (e.g. Krizhevsky et al. (2012)) have millions of parameters and the reasons why they take a particular decision have become opaque. These complex models whose mapping between the input data and the expected output (that we call "decision" or "prediction") is not explicit enough to be understandable by mere humans are called "black-box" models.

The use of black-box models for sensitive applications (e.g. in law, medicine, insurance, autonomous driving (Guidotti et al. 2018)), regardless of how successful they are, is the main reason for the questioning mentioned at the beginning of this introduction. In EU for example, new laws have been voted which impose a "right to explanation" in the General Data Protection Regulation (GDPR) [3]. In the USA, the DARPA's eXplainable AI (XAI) research program [4] also claims the importance of the XAI research field.

The most intuitive explanations for a black-box model are the ones which are the most

3. https://gdpr-info.eu/recitals/no-71/
4. https://www.darpa.mil/program/explainable-artificial-intelligence

familiar to human, i.e., the information we perceive in day-to-day life, such as visual information (images), human language (vocabularies, sound), smell, etc. for broad audiences, and the information related to the specialty of the domain experts, *e.g.* temperature time series for physicists, power time series for electricity providers. With these explanations, one may understand what is happened during the decision process in a black-box.

In this thesis, we focus on one particularly ubiquitous type of data: time series. A times series is a finite collection of real data indexed by time. It is one of the most common real life data types and models many processes such as stock exchange evolution, daily energy consumption, body temperature evolution of a particular patient, etc. (Shumway et al. 2005).

Time series analysis consists in extracting meaningful statistical characteristics of the series and making decisions based on these characteristics. Some time series analysis problems such as forecasting (such as predicting the next value(s) of a given series), regression (associating a numerical value to a given time series), clustering (grouping the time series according to their similarities), anomaly detection (finding time series that behave differently from others, or finding surprising values in a given series), and classification (classifying a time series into a set of predefined classes of interest), can be tackled using machine learning methods (Siffer et al. 2017; Leverger et al. 2019; Gauthier et al. 2021; Zuo et al. 2021). In this thesis we focus on the particular problem of *time series classification*.

In this thesis, we use the time series datasets from the UCR time series classification repository collected by University of California Riverside (Chen et al. 2015). The number of datasets of this archive increased with time. Now it is called UEA & UCR time series classification repository and contains 128 univariate and 30 multivariate time series datasets, available at `http://www.timeseriesclassification.com/`. We only use the 85 datasets from the UCR repository (Chen et al. 2015), because it is the most used time series classification (TSC) benchmark in the community. These 85 univariate time series datasets are collected from different data sources, *e.g.* ECG (electrocardiograms), images, motion sensors, synthetic examples, spectrographs, etc. The datasets are usually of moderate sizes, but they vary in size from 16 training instances (for the DiatomSizeReduction dataset) to 8926 (ElectricDevices dataset), vary in length from 24 (ItalyPowerDemand) to 2709 (HandOutlines), and vary in class numbers from 2 (*e.g.* GunPoint) to 60 (ShapesAll).

We show in Figure 3, a famous 2-class time series dataset called GunPoint, which has been collected from the hand motions of two actors. It involves time series for class

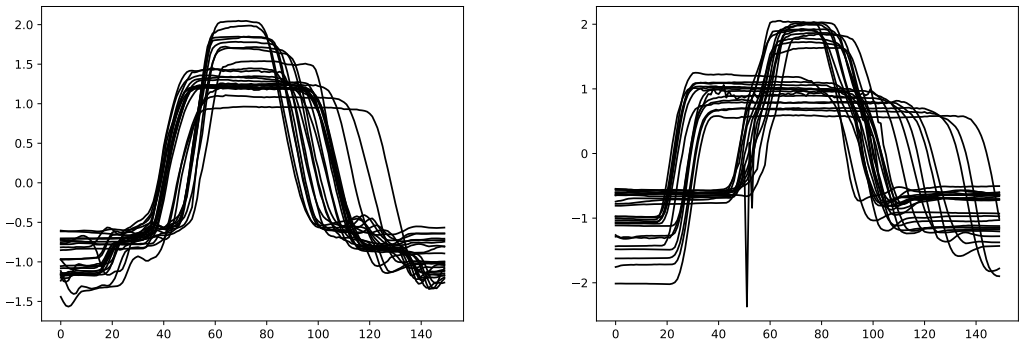


Figure 3 – Example of training time series from GunPoint time series dataset for GunDraw class (left) and Point class (right) from UCR repository (Chen et al. 2015).

GunDraw (the actor draws a gun from a hip-mounted holster, point it at a target for approximately 1 second and return the gun back) and *Point (also called NoGun)* (the actor acts with his/her fingers).

Many methods have been developed for Time Series Classification (TSC) (see for example (Bagnall et al. 2017; Tavenard et al. 2017; Lods et al. 2017; Guillemé et al. 2019a; Fauvel et al. 2019)). We categorize the existing methods into four groups: distance-based, feature extraction, neural networks, and ensemble methods. The rationals for these categories along with a review of each method are presented in Chapter 1.

One very successful category of methods consists in "finding" discriminative phase-independent subsequences, called *shapelets*, that can be used to classify or describe the series. The shapelets are subsequences, i.e., visual elements that are helpful for explaining the decisions taken by a model learned from time series data. They are thus a good candidate for tackling our *interpretable TSC* problem. In the first papers about shapelet-based time series classification (Ye et al. 2009; Rakthanmanon et al. 2013), the shapelets were directly extracted from the training set and the selected shapelets could then be used to explain the classifier's decision. However, the shapelet enumeration and selection processes were either very costly or the selection was fast but did not yield good performance (as discussed in Section 1.3.2). Jointly learning a shapelet-based representation of the series in the dataset and classifying the series according to this representation (Lines et al. 2012; Grabocka et al. 2014) allows to obtain discriminative shapelets in a much more efficient way. An example of such learned shapelets, obtained with the method from Grabocka et al. (2014), is given in Figure 4 (left). However, if the learned shapelets

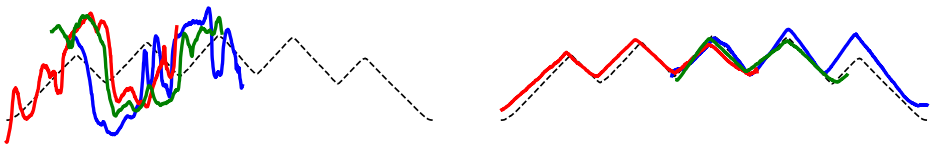


Figure 4 – Example test time series (dashed black lines) and three most discriminative shapelets used for its classification for a baseline (Grabocka et al. 2014) (left) and for our proposed XCNN model (right) on the HandOutlines dataset (from UCR repository (Chen et al. 2015)).

are well discriminative, they are often different (visually) from actual pieces of a real series in the dataset. As such, these shapelets might not be suited to explain a particular classifier's decision.

In this thesis, we address the interpretable time series classification problem. In Chapter 1, we introduce existing time series classification methods, in particular shapelet-based methods and neural networks. In Chapter 2, we clarify the notion of *model interpretability*: a model is interpretable as long as it can be understood by human. Based on the training process of the model, the interpretability of a model can be considered as *ad-hoc*, *in-situ*, and *post-hoc*. We discuss the importance of interpretability and review the interpretability of different machine learning models under different scopes, i.e., global level, modular level, and local level interpretability. For non-interpretable models, we discuss the notion of *post-hoc* interpretability to give a particular explanation. In Chapter 3, we adapt the framework proposed by Fauvel et al. (2020b) to evaluate the interpretability for time series classification. This framework provides information about the model interpretability and evaluates the quality of the explanations. In Chapter 4, we propose a method called eXplainable Convolutional Neural Network (XCNN). Our method learns shapelets that are constrained to be close to subsequences of the training time series. Our XCNN is *in-situ* interpretable in a modular level, and produces time series discriminative patterns. In the end of this document, we provide a conclusion for this work along with some perspectives.

Publications

The work presented in this thesis, has been presented and published in the following venues:

1. Yichang Wang, Rémi Emonet, Elisa Fromont, Simon Malinowski and Romain Tavenard. *Adversarial Regularization for Explainable-by-Design Time Series Classification*, ICTAI'2020 32th International Conference on Tools with Artificial Intelligence, p1079-1087, Virtual
2. Yichang Wang, Rémi Emonet, Elisa Fromont, Simon Malinowski, Etienne Menager, Lo ic Mosser, et Romain Tavenard, *Classification de séries temporelles basée sur des shapelets interprétables par réseaux de neurones antagonistes*, CAp'2019 (Conférence d'Apprentissage), Toulouse, 2019
3. Yichang Wang. *Explainable Time Series Classification*, Workshop "Fair and Explainable Models" at the the 31th EURO Conference (European Conference on Operational Research), Virtual 2021

Part I

Background

Chapter 1

Time series classification

The Time Series Classification (TSC) problem has been studied in countless applications (see for example (Shumway et al. 2005)) ranging from stock exchange evolution, daily energy consumption, medical sensors, videos, etc. This chapter introduces the necessary background for this problem.

1.1 Definition and taxonomy

Definition 1 (Time series)**.** A time series (TS) Z is a series of time-ordered values, $Z = [z^{(1)}, z^{(2)}, \ldots, z^{(T)}]$, where $z^{(t)} \in \mathbb{R}^d$, T is the length of our time series and d is the dimension of the feature vector describing each data point.

If $d = 1$, Z is said *univariate*, otherwise it is said *multivariate*.

Definition 2 (Time series classification)**.** Given a training set $\mathcal{T} = \{(Z_1, y_1), \ldots, (Z_n, y_n)\}$, composed of n time series Z_i and their associated labels y_i (target variable), our aim is to learn a function h such that $h(Z_i) = y_i$, in order to predict the labels of new unseen time series.

There exists a rich literature on the TSC problem (see for example (Bagnall et al. 2017; Tavenard et al. 2017; Lods et al. 2017; Guillemé et al. 2019a; Fauvel et al. 2019)). We have decided to group TSC approaches from literature into four main families as shown in Figure 1.1. First, a dedicated metric can be used to compare the distances between the time series. The most well-known and used metrics are the Euclidean distance and the Dynamic Time Warping (DTW) similarity. Note that if the DTW is often loosely qualified as a distance, in theory, it is not a valid metric (cf. Section 1.2.2).

The second family is based on the extraction of features from the time series. The feature extraction process extracts meaningful patterns or transforms the time series into different representations than the original one, including decomposition, aggregation approximation, and shapelet-based methods. Among the feature extraction category,

shapelet-based classifiers have attracted a lot of attention from the research community and are the main focus of this thesis.

The third family includes all neural network-based approaches, including multilayer perceptron, convolutional neural networks, recurrent neural networks, and attention-based networks such as "Transformers". If some neural networks could be included into the "feature extraction" family, we believe that the end-to-end treatment made by neural networks makes them different in essence. Compared with traditional (hand-crafted) features extraction methods (see Section 1.3), the features extracted by deep learning models are learned directly from the data without any *a priori* consideration to the "shape" of the learned features and their usefulness for interpretations.

The best performing (in terms of classification) methods are ensemble methods. State-of-the-art ensemble methods combine representations in different domains (e.g. time domain, frequency domain, shapelet domain, etc.). Both the combination scheme and the heterogeneous representations makes them poor candidates for interpretations. This problem is discussed in Section 2.3.

The following of this chapter presents these four TSC method families.

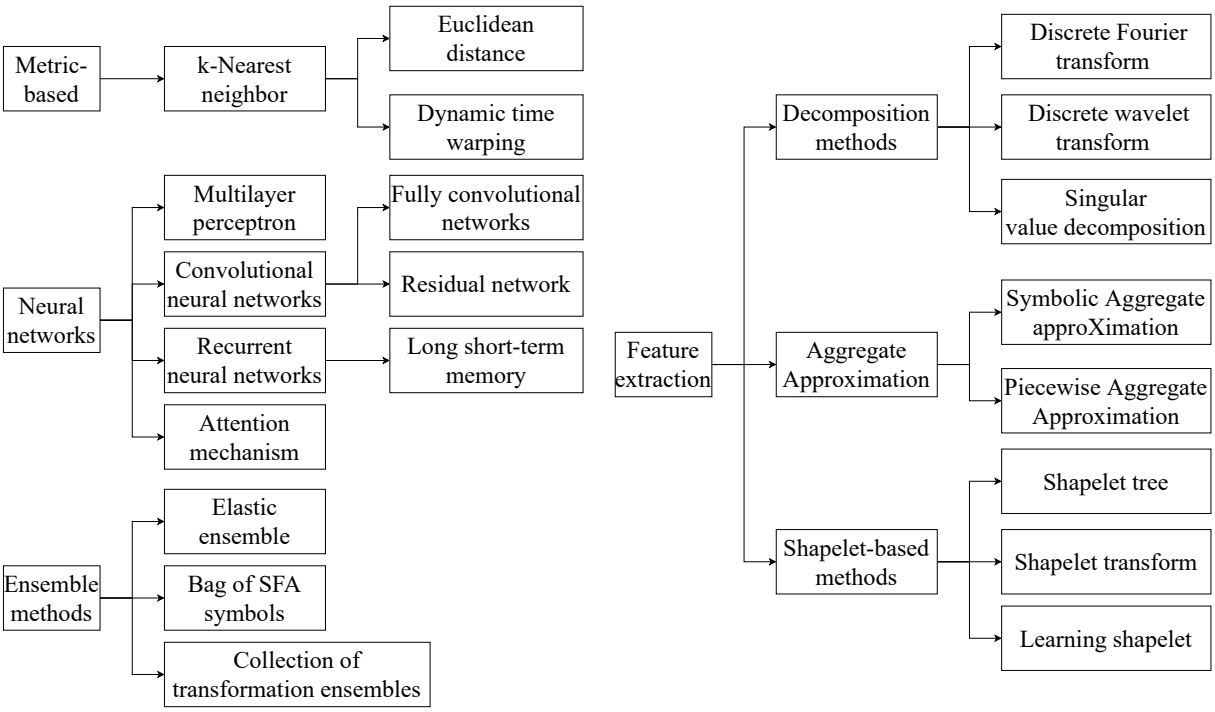


Figure 1.1 – Taxonomy of time series classification methods.

1.2 Distance-based methods

Distance-based methods measure the similarity between different time series with a dedicated distance measure. The distances, presented below, can be *rigid* (for example the Euclidean one) or *elastic* (for example the Dynamic Time Warping (DTW) one). The distances can focus on parts of the series or the entire series. Once defined, they can be incorporated into machine learning classifiers such as *Nearest neighbor* classifiers.

1.2.1 Euclidean distance

The Euclidean distance is the most common one. The pairwise Euclidean distance between two time series, Z and Z' of the same length T, is defined as:

$$D_E(Z, Z') = ||Z - Z'|| = \sqrt{\sum_{i=1}^{T} ||z^{(i)} - z'^{(i)}||^2}.$$

This distance is particularly easy to compute but, when the distance is computed on entire series, it requires the series to be of the same length which is a strong limitation in practice. Besides, it is not entirely suited for time series analysis since it does not consider the temporal scaling of the series.

1.2.2 Dynamic time warping (DTW)

Dynamic time warping (DTW) (Sakoe et al. 1978) is a commonly used distance to find the optimal alignment between two time series. The optimal path is computed based on a cross-similarity matrix, and it contains the optimal alignment information for two time series. The DTW only considers the ordering of the time series values, and the exact occurring timestamps are ignored, as shown in Figure 1.2. [1]

The DTW distance between time series Z, Z' of size n and m, respectively, is defined as an optimization problem on $\mathcal{A}(Z, Z')$, i.e., the set of all admissible paths (Tavenard 2020):

$$D_{DTW}(Z, Z') = \min_{\pi \in \mathcal{A}(Z,Z')} \sqrt{\sum_{(i,j)\in\pi} ||z^{(i)} - z'^{(j)}||^2}$$

where $\pi = [\pi_1, \pi_2, \ldots, \pi_K]$ is a sequence of $K \equiv (n + m - 1)$ index pairs, and "$\equiv$" is

1. If not specified, all the time series figures in this chapter are generated with `tslearn` package (Tavenard et al. 2020).

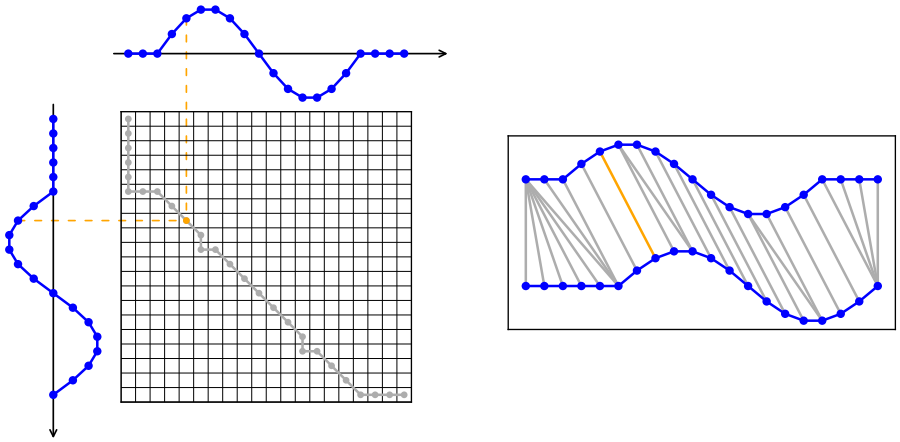


Figure 1.2 – DTW example for two series X and Y: optimal warping path (left) and alignment (right).

the identical sign that indicates the equation is always satisfied. The path $\pi_k = (i_k, j_k)$ follows such properties:

- $i_k \in [1, n]$ and $j_k \in [1, m]$
- $\pi_1 = (1, 1)$ and $\pi_K = (n, m)$
- for all $k > 1$, $\pi_k = (i_k, j_k)$ is related to $\pi_{k-1} = (i_{k-1}, j_{k-1})$ as follows:
  - $i_{k-1} \leq i_k \leq i_{k-1} + 1$
  - $j_{k-1} \leq j_k \leq j_{k-1} + 1$

DTW takes into account the time by warping the time axis of one (or both) time series to achieve a better alignment. It can be applied on time series with the same dimension but different lengths. DTW is time costly as its time complexity is quadratic ($O(mn)$). Mathematically, "DTW is not a valid metric since it satisfies neither the triangular inequality nor the identity of indiscernibles." (Tavenard 2020)

There are many DTW variants (Lines et al. 2015), and methods are developed to accelerate the DTW-based searches (Tavenard et al. 2015). However, these methods are out of the scope of this thesis.

1.3 Feature extraction methods

In machine learning, a feature is a variable that informs about observations, *e.g.*, a measurement at a certain timestamp is a feature in time series. The features can be sep-

arated into two categories: raw features (also called attributes) and informative features. Both these features contain characteristic information of the datasets. Raw features, i.e., features in their original form, can be difficult to manipulate due to the redundancy between them or their large dimensionality (Bellman 1957). Thus, they usually cannot be used to solve realistic problems. One usually needs to extract information from raw features through feature engineering techniques, and make predictions based on these informative extracted features (stored in a so-called feature vector). The space that a feature vector lies in is called the feature space.

Time series data can be extremely large in the temporal dimension, and most of the TSC methods first try to extract the representative/informative features, then train a classifier based on these features. These extracted features are preferred to be meaningful patterns, such as frequency features from Fourier transform, motifs (frequently occurring patterns), or shapelets, instead of features in latent space, such as the "feature vector" in neural networks. Traditionally, decomposition methods were popular techniques for feature extraction, including spectral methods, *e.g.* Fourier transform (Agrawal et al. 1993; Faloutsos et al. 1994), wavelet transform (Popivanov et al. 2002), and eigenvalue analysis, *e.g.* singular value decomposition (SVD) (Korn et al. 1997). We do not dive into these decomposition-based methods because they are already integrated into cited recent methods.

We now introduce the feature extraction techniques used nowadays in time series classification, namely *aggregate approximation* methods, which are simple but rapid feature extraction techniques, and *shapelet-based* methods, in which subsequences of the time series are used as discriminative patterns.

1.3.1 Aggregate approximation

In this section, we discuss symbolic representations of the time series, which are downsampling techniques that compress the time series. The aggregate approximation mainly includes two famous aggregate approximation representations (Wang et al. 2019): Piecewise Aggregate Approximation (PAA) and Symbolic aggregate approXimation (SAX). PAA is used to reduce the temporal resolution by aggregating the mean value of subsequences of the time series (Keogh et al. 2001), SAX is based on PAA by splitting the value ranges to symbolic values (Lin et al. 2003), under an assumption that the PAA values follow a Gaussian distribution (Lin et al. 2007). Aggregate approximation methods are widely applied in time series analysis.

Piecewise Aggregate Approximation (PAA)

The PAA method was firstly introduced by Keogh et al. (2001). It requires one parameter, i.e., the number of PAA segments (or, alternatively, the PAA window size which is equivalent if the time series length is fixed), to generate PAA sequences from the original time series. The PAA representation is generated in two steps:

1. Divide a time series into segments of a given window size
2. Calculate the mean value for each PAA window

The PAA representation of different window size is shown in Figure 1.3. The quality of the PAA segments depends on the number of segments: too few windows will average all the characteristics out from the time series, while too many windows will give little difference to the original series, thus making the feature extraction process meaningless.

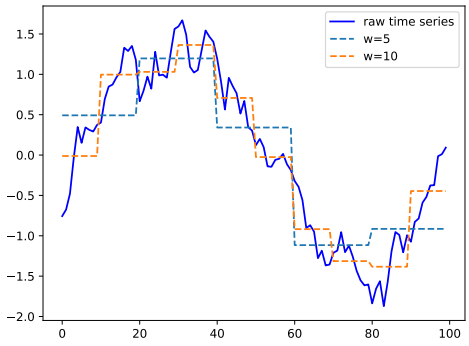


Figure 1.3 – PAA segments with different window numbers.

Symbolic aggregate approXimation (SAX) and 1d-SAX

The SAX representation is based on the PAA method. It transforms the original time series into down-sampling symbolic representation under the assumption of Gaussian distribution (Lin et al. 2007). The SAX representation is generated in two steps:

1. Compute the PAA representation for the time series
2. Quantize the PAA representation in each window into a symbol from an alphabet.

Both PAA and SAX representations are sensitive to the window size. In addition, the SAX method depends on the alphabet size (the numbers of symbols used for quantization) and on the assumption of Gaussian distribution. Malinowski et al. (2013) pointed out

that the SAX representation does not contain the trend information on the SAX segments. They proposed a 1d-SAX method to quantize both the slope and the average value of the segments. The 1d-SAX representation has better retrieval performance than the original SAX method. The SAX and 1d-SAX representations are shown in Figure 1.4.

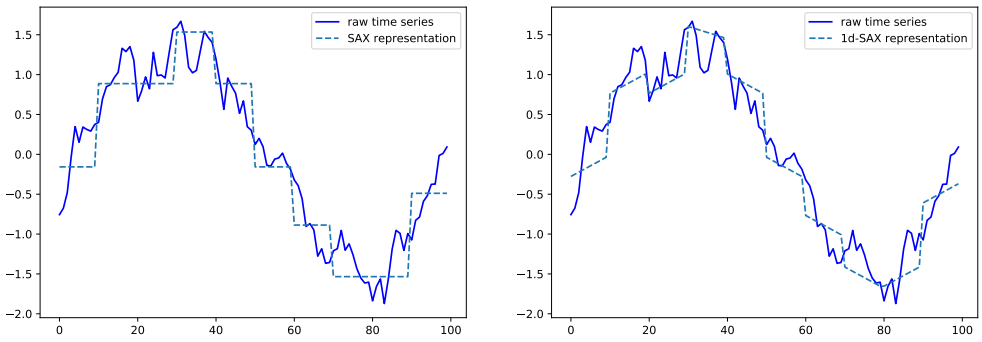


Figure 1.4 – SAX (left) and 1d-SAX (right).

1.3.2 Shapelet-based methods

Shapelets are discriminative subseries that can be either extracted a priori from a set of time series or learned to minimize an objective function, as shown in Figure 1.5. They have been introduced by Ye et al. (2009), in which a binary decision tree is built, whose nodes are shapelets and whose subtrees contain subsets of time series. In this work, shapelets are extracted from a training set of time series and building the decision tree requires to test all possible subseries from the training set, which makes the method intractable for large-scale learning with an overall time complexity of $O(n^2 \cdot T^4)$ where n is the number of training time series and T is the average length of the time series in the training set. This high time complexity has led to the use of heuristics in order to select the shapelets more efficiently.

In Fast Shapelets (FS) (Rakthanmanon et al. 2013), the authors rely on quantized time series and random projections in order to fasten the shapelet search. Note however that these improvements in time complexity are obtained at the cost of a lower classification accuracy, as reported in (Bagnall et al. 2017).

The Shapelet Transform (ST) (Lines et al. 2012) consists in transforming time series into a feature vector whose coordinates represent distances between the time series and

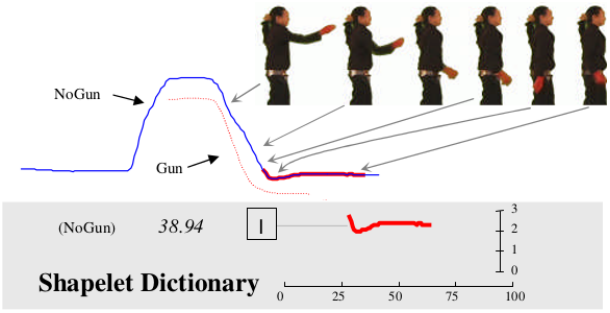


Figure 1.5 – Shapelet (in red) from Ye et al. (2009): if the distance between the shapelet and a time series is smaller than 38.94, the time series belongs to NoGun class. In practice, the shapelet performs as a sliding window on the time series, and the distance between a shapelet and a time series is a squared Euclidean distance at the best matching position.

the shapelets selected beforehand. It hence needs to select a shapelet set (as in (Ye et al. 2009)) before transforming the time series. The resulting vectors are then given to a classifier in order to build the decision function. The training time complexity for ST is also in $O(n^2 \cdot T^4)$ (Fawaz et al. 2019), which makes it unfit for large scale learning.

In order to face the high complexity that comes with search-based methods, other strategies have been designed for shapelet selection. On the one hand, some attention has been paid to random sampling of shapelets from the training set (Karlsson et al. 2016; Guillemé et al. 2019a). On the other hand, Grabocka et al. (2014) showed that shapelets could be learned using a gradient-descent-based optimization algorithm. The method, referred to as Learning Shapelets (LS) in the following, jointly learns the shapelets and the parameters of a logistic regression classifier. This makes the method very similar in spirit to a neural network with a single convolutional layer followed by a fully connected classification layer and where the convolution operation is replaced by a sliding-window local distance computation. A min-pooling aggregator should then be used for temporal aggregation.

1.4 (Deep) neural networks

A neural network is a graph of computing units (called neurons) structured in layers. Each neuron receives signals from other neurons and outputs a single value (McCulloch et al. 1943). More generally, a neuron applies a nonlinearity on a linear combination of input values, and the output of a neuron can be written as $\sigma(w^\tau x + b)$, where $x =$

$(x_1; x_2; \ldots; x_d)$ is the d-dimensional input vector of the neuron, $w = (w_1; w_2; \ldots; w_d)$ the neuron weights, b the bias, $(\cdot)^\tau$ the transpose operation, and $\sigma(\cdot)$ the activation function (nonlinear transformation). The combination of neurons with nonlinearities gives neural networks great capacity to form complex functions, and the neural network architecture depends on the way the combination is made.

The simplest (modern) NN architecture is a fully connected network (also called a multilayer perceptron (MLP)), in which all neurons of one layer are connected to all neurons of the following layer. In addition to fully connected networks, neural networks can have different architectures, *e.g.*, convolutional networks that apply convolutional operations to extract visual features (Lecun et al. 1998), recurrent networks that have the property to process sequential data. Recently, attention mechanism in the neural networks has attracted considerable attention within the scientific community. By applying attention mechanism, the neural network layers can pay attention to a specific zone of the input. Different from other methods, these neural networks can be considered as an end-to-end approach that integrates the feature engineering process in the learning problem.

In this section, we recall neural network architectures that appeared in time series classification literature with their advantages and disadvantages. We explain the neural networks in more details than other TSC methods because the contribution of this thesis is directly related to neural networks.

1.4.1 Multilayer perceptron (MLP)

Wang et al. (2017) were the first to apply neural networks to time series classification (TSC) problems. They examined different network architectures, including a multilayer perceptron (MLP), a fully convolutional neural network, and a residual network. In this section we only review the MLP proposed by Wang et al. (2017).

The output vector h of a hidden block for an input vector x of the MLP is formed as:

$$x = f_{dropout,p}(Wx),$$
$$h = ReLU(x),$$

where $ReLU$ is the rectified linear unit operation that keeps the positive parts of its arguments, i.e., $ReLU(x) = \max(x, 0)$.

This MLP has 3 hidden layers with 500 neurons for each layer. The last layer of the MLP has the same number of neurons as the number of classes K, followed by a softmax

layer, as shown in Figure 1.6.

The softmax layer calculates the probability that the input time series belongs to the class j:

$$\hat{Y}_j = \frac{\exp(a_j)}{\sum_{i=1}^{K} \exp(a_i)}$$

where the a_j is the j-th activation of the last layer before the softmax layer.

To train a neural network, we firstly initialize the parameters (also called weights) that connect the neurons randomly, then we fit the training set with the neural network by applying an optimization method to find the parameters that minimize a loss function. The usual loss function in classification is the multi-class categorical cross entropy:

$$L(Z) = -\sum_{j=1}^{K} Y_j \ln \hat{Y}_j, \tag{1.1}$$

where j is the j-th class out of K classes, $\hat{Y}_j$ the output of the model for class j, and Y_j the true label (called ground truth) of the input series Z for class j.

The MLP is not suitable for time series data, because (i) it does not take into account the temporal information as each time stamp is treated independently (Fawaz et al. 2019), (ii) the length of the first hidden block of the network is fixed thus it is not suitable for time series with different input lengths. Another inconveniences for the MLP proposed by Wang et al. (2017) is that the number of neurons in each block is fixed, thus the MLP could easily encounter generalization problems (overfitting or underfitting).

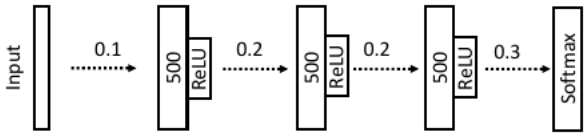


Figure 1.6 – Architecture of MLP. Each block has 500 neurons, and the values between each block are the dropout probability (Wang et al. 2017).

1.4.2 Convolutional neural networks (CNNs)

Preliminary work on convolutional networks for TSC was undertaken by Wang et al. (2017), then a large and growing body of literature has investigated (Pelletier et al. 2019; Fawaz et al. 2020; Dempster et al. 2020). We discuss TSC methods rely on convolutional

neural networks (CNNs) in this section.

Conventionally, in the machine learning domain, the convolution operation is implemented as the cross-correlation (also known as sliding dot product), which is similar to the convolution in nature but without flipping the kernel (Goodfellow et al. 2016). The convolution operation is a sliding window over the temporal dimension of the time series. It can be considered as a filtering process that generates another time series. In practice, each convolutional layer can have more than one filter [2], and each filter is called a *channel*. The convolutional layer generates multivariate time series whose dimension equals to the number of channels. The filters are learned as feature extractors that transform a time series into multiple discriminative features.

Fully convolutional network (FCN)

In the fully convolutional network (FCN) proposed by Wang et al. (2017), each hidden layer of the FCN does not have pooling operation which means that the length of the time series does not change (zero padding is applied) until the last layer. The output of a basic building block of the FCN is formed as:

$$x = BN(W * x),$$
$$h = ReLU(x),$$

where the asterisk $*$ denotes the convolution operation, BN the batch normalization operation that re-scales and re-centers the feature maps for stabilizing the training process.

The FCN has three hidden blocks, and each block has 128, 256, 128 filters with length of 8, 5, 3, respectively. The output of the last block is a global average pooling (GAP), which corresponds to an average over the whole time dimension. The third hidden block is linked to a fully connected layer followed by a softmax similar to the last layer of the MLP.

Usually, the convolutional operation for TSC can be a 1D convolution that extracts *temporal* features, while for computer vision task (where the input is a 2D image) it is a 2D convolution that extracts *spatial* features. For multivariate time series (MTS), one may use diverse convolution operations for both temporal-spatial features.

Pelletier et al. (2019) studied CNNs for satellite image time series (SITS) classification and proposed another architecture similar to FCN called temporal convolutional neural

2. A convolutional filter is also called a convolutional kernel.

networks (TempCNNs). The TempCNNs architecture replaced the pooling operation of the FCN by a flatten operation, which performs the same as the vectorization of a matrix in linear algebra that converts the feature matrix into a column feature vector, and then added an extra FC layer between the feature vector and the FC output layer to reduce the dimension smoothly. The study of Pelletier et al. (2019) is limited for SITS classification. They show that the conventional pooling operation in image classification and FCN for time series is harmful for SITS classification.

Residual networks (ResNet)

The residual networks architecture was introduced by He et al. (2016) for making neural networks deeper by adding residual blocks in the neural networks. Many object detection tasks that achieve state-of-the-art performance are based on ResNet architecture (for example (Zhang et al. 2020)).

The ResNet architecture is similar to the FCN except the basic building block of the ResNet has a residual connection:

$$\tilde{x} = BN(W * x),$$
$$h = ReLU(\tilde{x} + x)$$

The ResNet has three hidden blocks. The length of the filters for each block is set to 8, 5 and 3 respectively. The idea of residual block can help to form a much deeper network that contains a considerable amount of parameters, thus a deep ResNet can be overfitted easily on small time series datasets. The original the numbers of convolutional kernels were set to 64, 128, 128, resp., while Fawaz et al. (2019) reduced this number to 64 for each block. In Wang et al. (2017), the FCN outperforms the ResNet, but with the little modification made by Fawaz et al. (2019) the ResNet outperforms the FCN.

InceptionTime

In addition to the preliminary works on convolutional networks, Fawaz et al. (2020) proposed another CNN architecture called *InceptionTime* for TSC. The proposed architecture was inspired by Inception-v4 (Szegedy et al. 2017) for image classification problem. The InceptionTime architecture is an Inception network with residual connections. The building block of InceptionTime is an Inception module, as shown in Figure 1.7.

The Inception module has a bottleneck layer which is composed of several filters of length 1. This bottleneck layer reduces the dimension of the input multivariate time series, thus make it possible to learn longer convolutional filters (than ResNet) with roughly the same number of parameters. Convolutional layers with filters of different lengths are applied on the bottleneck outputs to extract features. In parallel, a shortcut is made by a max pooling operation, followed by another bottleneck layer in order to make the model invariant to small perturbations.

The concatenation of InceptionTime modules makes it possible to have a larger perspective field. Note that the bottleneck layer is meaningless for reducing the dimension of univariate time series, thus the bottleneck layer is skipped while applying the InceptionTime for univariate time series.

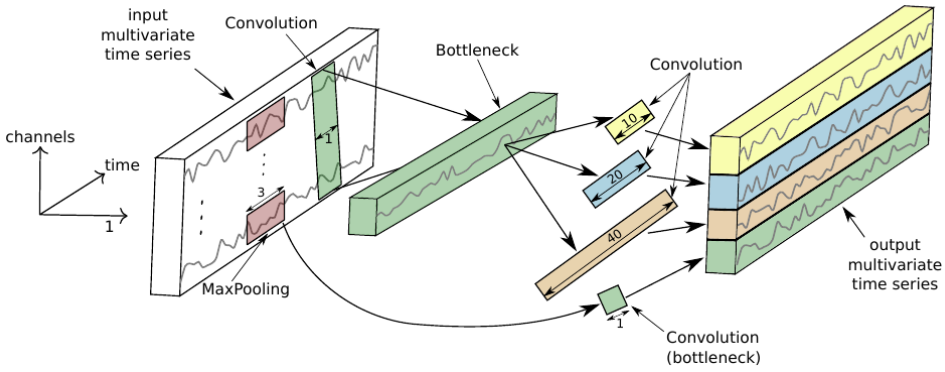


Figure 1.7 – Inception module: the building block of InceptionTime (Fawaz et al. 2020).

RandOm Convolutional KErnel Transform (ROCKET)

RandOm Convolutional KErnel Transform (ROCKET) was introduced by Dempster et al. (2020). It applies a numerous number of convolutional kernels as feature extractors, but all these kernels are randomly generalized, including the kernel lengths (selected in 7, 9, 11), the kernel weights (sampled from a normal distribution), and the kernel biases (sampled from a uniform distribution). When applying these kernels, the parameters for each kernel are also randomly chosen, including padding (applying padding will add zeros to the start and end of a time series to make the output has the same length of the input), and dilation (controls the spacing between the kernel points, sampled on an exponential scale). Stride (the step of each move of the kernels) is set to one in ROCKET.

Once these aspects of the kernels are randomly fixed, ROCKET transforms the input time series with these kernels to generate random features. The informative ones are selected by training a linear classifier on the transformed features.

1.4.3 Recurrent neural networks (RNN)

Different from feed forward neural networks, RNNs belong to another famous family of neural networks, which are applied to process and to generate sequences with different lengths. RNNs have inner loops in the networks, and these loops allow RNNs to take into account the previous information of the sequence. RNN architectures are especially useful for tasks like speech recognition and nature language processing (NLP), in which there is sequential data like time series and text.

The weights of RNN can be estimated by back-propagation through time (Rumelhart et al. 1986). However, vanilla RNN is difficult to train because of the gradient vanishing/exploding problem (Bengio et al. 1994). In addition, although RNN can take into account the previous information [3] in a sequence, it is not able to learn the connection between the information (also called long-term dependencies) if there is a huge gap between the relevant timestamps. To fix the gradient vanishing problem, as well as the arbitrary long-term dependencies, different variants of RNNs have been developed, and among them, the long short-term memory networks (LSTM) (Hochreiter et al. 1997) and gated recurrent unit (GRU) (Cho et al. 2014) were the most successful ones.

Despite the benefits of LSTM and GRU networks, they still have some limitations:

1. Although LSTM networks are easier to train (than a vanilla RNN), when the length T of an input time series is too large, the LSTM becomes a T-layer deep neural network with its recursive operations. This may still cause gradient vanishing/exploding problem, thus the long-term memory can not be infinite long.
2. The recursive nature of LSTM (and other RNNs), i.e., each hidden state of the LSTM is calculated based on the previous state, makes it impossible to train in parallel and the training process will be long.

Apart from forecasting, one rarely applies RNNs for time series classification due to these limitations which cause a poor performance (Fawaz et al. 2019). In practice, RNNs are integrated as a module in a convolutional network (see "LSTM-FCN" (Karim

3. with a bi-directional RNN architecture, we can also take into account the information in the following sequences.

et al. 2018) for example). Self-attention (as discussed in the next section) handles these problems of RNNs, and lead to a better performance in many domains.

1.4.4 Attention mechanism

Attention mechanism was introduced by Bahdanau et al. (2015) to increase the performance of an auto-encoder proposed by Cho et al. (2014) in neural machine translation. In this NLP task, the auto-encoder encodes a source sentence into a vector in latent space, and decodes the latent vector into a sentence in the target language. The attention mechanism allows the decoder to pay attention to the segments of the original sentence for each target word. This attention connects the encoder and decoder, or generally speaking, is used between different layers in a neural network.

Vaswani et al. (2017) proposed a network architecture called "Transformer" that applies a *self-attention* to the input of one layer. Different from the original attention, self-attention is applied inside a layer and calculates the correlational importance between an input sequence and each other sequence of the input sequences of this layer.

Self-attention eliminates the limitations of RNNs (as shown in Section 1.4.3): it has infinite attention size by using all-to-all comparison, and can be trained in parallel. Overwhelming variants of Transformer, called xformer (see (Tay et al. 2020) for a review), have been developed based on the self-attention. For TSC tasks, self-attention has been applied with CNNs to form new network architectures, and these self-attention-based networks have obtained state-of-the-art performance (Lin et al. 2020; Hao et al. 2020; Garnot et al. 2020).

1.5 Ensemble methods

Ensemble methods combine the predictions of different individual learners as their final prediction. This kind of machine learning models can get better generalization performance than individual learners. In this section, we recall the ensemble methods applied to time series classification. A general view of ensemble methods is available in the "non-interpretable ML models" section (Section 2.4.3).

1.5.1 Elastic ensemble (EE)

The elastic ensemble (EE) method (Lines et al. 2015) combines 11 1-nearest neighbor classifiers as its base classifiers. The 11 similarity measures for these nearest neighbor classifiers include Euclidean distance, DTW, and DTW variants.

Lines et al. (2015) conducted extensive experiments on 75 datasets. They found that the individual elastic measure-based classifiers used in the elastic ensemble method do not show significant difference among each other, but through ensemble, elastic ensemble method obtains a significantly more accurate classifier than all the nearest neighbor classifiers with a single metric.

1.5.2 Bag of SFA symbols (BOSS) ensemble

BOSS ensemble method (Schäfer 2015) used Bag of SFA symbols (BOSS) models as its base classifiers. This base classifier is essentially a bag-of-words model (Lin et al. 2012), which represents a sequence (*e.g.* a time series) as the bag of its words (SFA symbols in BOSS context), i.e., a dictionary that maps each SFA word to the counts of its occurrence.

The BOSS model transforms the time series into symbolic Fourier approximation (SFA) features [4] and compares the words in a nearest neighbor classifier based on a *BOSS distance* to make prediction. The BOSS distance between two time series $dist_{BOSS}(Z, Z')$ is a modified non-symmetrical Euclidean distance. It calculates the mean square error of the pairwise SFA word count differences, but all SFA word counts of 0 in the query Z are omitted.

The BOSS ensemble is a combination of multiple BOSS classifiers at different window lengths. Based on the SFA symbols and aggregate approximation, which naturally are low pass filters, the BOSS ensemble can filter out the high-frequency noise in the time series, thus has better performance in classification.

1.5.3 Collection of transformation ensembles (COTE)

In this section, we recall the collection of transformation ensembles (COTE) and the hierarchical Vote Collective of Transformation-Based Ensembles (HIVE-COTE).

The collection of transformation ensembles (COTE) method is a meta-ensemble, because two components of the COTE are ensemble methods (random forest and rotation

4. Similar to aggregate approximation, but uses a DFT instead of a PAA/SAX on each window, and the window size is also changeable.

forest) (Bagnall et al. 2015). COTE combines time series representations in four different domains, including the time, autocorrelation, power spectrum and shapelet domains (Bagnall et al. 2017). In total, COTE has 35 base classifiers. The final prediction is a weighted vote for all these 35 classifiers based on the cross validation accuracy of each classifier. This original COTE method is also called Flat-COTE, because the combination of the classifiers is flat.

The original HIVE-COTE (also called HIVE-COTE alpha) includes five ensembles as its modules, namely elastic ensemble (EE), shapelet ensemble, BOSS ensemble, time series forest, and spectral ensemble. It uses a hierarchical voting probability, i.e., a normalized weighted sum over these five modules (Lines et al. 2016). COTE considers each classifier as a single module, while HIVE-COTE uses each ensemble as a module. The objective of this HIVE-COTE alpha is the performance in classification, thus it does not concern the computational complexity and it is very slow.

HIVE-COTE 1.0 (HC1) (Bagnall et al. 2020) drops the elastic ensemble from the HIVE-COTE alpha because of the high computational complexity, and applies the cross-validation accuracy weighted probabilistic ensemble (CAWPE) structure (Large et al. 2019). HIVE-COTE 2.0 (HC2) (Middlehurst et al. 2021) replaces the ensemble modules in HC1 with three state-of-the-art ensemble methods. In HC2, the ensemble modules are the temporal dictionary ensemble (TDE) (Middlehurst et al. 2020b), the canonical interval forest (DrCIF) (Middlehurst et al. 2020a), and the random convolutional kernel transform (ROCKET) (Dempster et al. 2020).

Chapter 2

Interpretability in machine learning

This chapter covers the main methods used to explain or interpret the decisions of a particular machine learning model and, in particular, of a time series classifier. I will first discuss the vocabulary and explain why I decided to use the terms "interpretable models" and "interpretability" instead of the now more traditional "explainable AI" through this thesis.

2.1 Interpretability in general

> "I couldn't reduce it to the freshman level. That means we really don't understand it."
>
> – – Richard Feynman

Interpret means "to explain or to present in understandable terms [1]", and interpretability is the ability for a piece of information to be understandable by humans. Interpretability is important to a human, because we are born to be curious for explanations.

Information that a human perceives naturally in day-to-day life is considered interpretable, because of its familiarity, its ubiquitousness. However, interpretability is a subjective concept: whether a piece of information is interpretable depends on the knowledge of the perceiver. For a broad audience, visual information (images), human language (vocabularies, sound), smell, etc., are interpretable, while non-expert may not be able to interpret pressure time series or power time series which could, however, provide meaningful information to physicists.

1. Merriam-Webster dictionary, accessed 2017-02-07

2.2 What is model interpretability?

Bringing interpretability to machine learning is not a new idea. Kodratoff (1994) for example, mentions that the importance of comprehensibility or explanations about the learned models for industrial applications dates back to the 1980s. During the bygone era, people have used different words, *e.g.*, comprehensibility, understandability, transparency, justifiability, *etc.*, to denote *interpretability* and *explainability*, which are the terms used nowadays (Bibal et al. 2016).

Unfortunately, the terms in the literature are not yet standardized. Lipton (2018) claims that "interpretability is not a monolithic concept". Interpretability is a very elusive concept, due to the fact that it is a domain specific notion (Rudin 2019). Interpretability is a *mental fit*, that is related to the degree of a human to evaluate the model, while *data fit* corresponds to predictive accuracy (Bibal et al. 2016). Different users consider interpretability differently.

Generally speaking, the disagreement nowadays comes from the ambiguity between *interpretability* and *explainability*. I will try to clarify these terms by reviewing definitions given by different researchers.

Bibal (2020) defines interpretability as a property of an interpretable model: "a model can be said to be *interpretable* if, within a given time limit, the level of expertise of the user allows him to understand the model through its representation", while explainability is the capacity of the model to be explained based on "*post-hoc* explanations": the explanations of the prediction results of a complex trained model. This definition of explanations matches the commonly used definition for eXplainable Artificial Intelligence (XAI) which is the research domain that gives insights on the behavior of complex models learned by different machine learning algorithms (Gilpin et al. 2018).

Rudin (2019) agrees with the definition given by Bibal (2020), and she advocates using the term *inherently interpretable* models for interpretable models [2]. In this thesis, we call inherently interpretable models *in-situ* interpretable models. They are *self-explainable* ones for which no *post-hoc* methods is needed to understand what the model does or why it takes a particular decision. Note that there is a small disagreement between Rudin (2019) and Bibal (2020). From the definition given by Bibal (2020), only simple enough models are *in-situ* interpretable, while for Rudin (2019), a model is interpretable when a part of the model is understandable by a human.

2. In the literature on interpretability, numerous names are used for inherently interpretable models (see Section 2.2.1 for details).

Other people consider that interpretability and explainability are interchangeable (Du et al. 2019; Jacovi et al. 2020). Molnar (2019) and Miller (2019) said: "Interpretability is the degree to which a human can understand the cause of a decision in a given context." For them, interpretability is the same thing as explainability (*post-hoc* explanations) defined by Bibal (2020).

The definition of model interpretability from Doshi-Velez et al. (2017) and Biran et al. (2017) can be used both for *in-situ* or for *post-hoc*, which makes a more general definition:

Definition 3 (Model interpretability)**.** Model interpretability is the property of a model that the operations in the model are understandable by a human, "either through introspection or through a produced explanation".

In Definition 3, the term *(model) interpretability* is a more general concept, which unifies the terms interpretability and explainability of the above cited authors. In the following sections of this thesis, we will use Definition 3 as our definition of *model interpretability*, and we will **avoid** using the term *model explainability*.

In my opinion, it is the *in-situ* and *post-hoc* model interpretability (as explained later in this section) which matter the most. We will only call *in-situ* interpretable models shortly for interpretable models, because *post-hoc* model interpretability is obtained with *explanations*, which are specifically obtained with *post-hoc* interpretability techniques applied on the prediction results of black-box models in order to enhance model interpretability. The relation of these terms are shown in Figure 2.1.

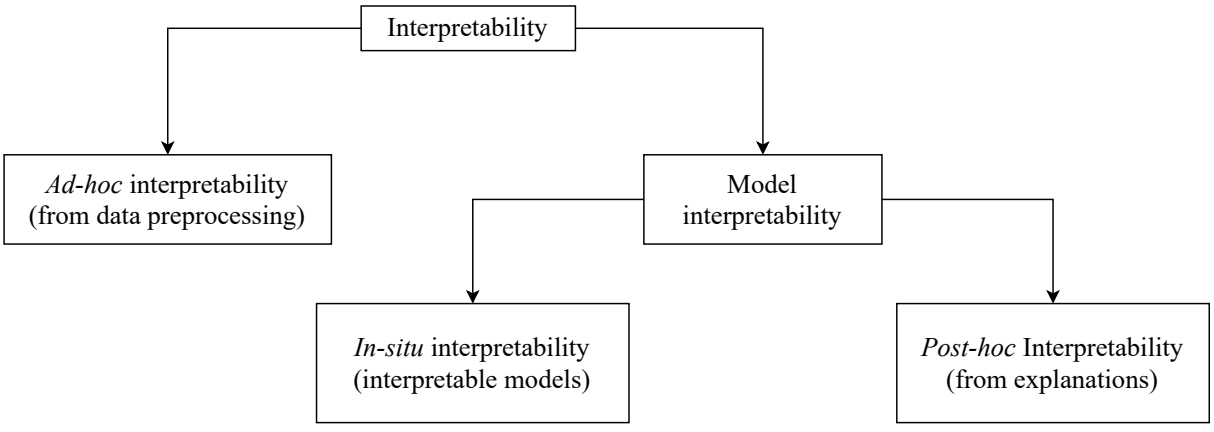


Figure 2.1 – Terms related to interpretability in this thesis.

2.2.1 Data, model or prediction?

In this section, we discuss Figure 2.1 in detail, including *ad-hoc*, *in-situ*, and *post-hoc* interpretability. They represent the interpretations during the preprocessing of the data (before training), the *self-explainability* property inside the model (during/after the training of the model), and the explanations of a prediction at inference (after training), respectively. Note that *in-situ* interpretable models are also capable to provide *post-hoc* explanations.

Ad-hoc interpretability

Ad-hoc interpretability is rarely discussed in the interpretability literature. It is the understanding of the data during the data preprocessing, which usually happens before model selection. The type of *ad-hoc* interpretation might be different depending on the data format: data are interpretable when they are expressed in low dimensions (especially for tabular data and graphs), or have intuitive features (especially for images, text, and time series), which can be more easily understandable by humans.

Ad-hoc interpretability can be achieved with dimensionality reduction techniques (such as feature selection (Doquet et al. 2019)) during the preprocessing of the data. We integrate the dimensionality reduction methods in the section dedicated to interpretable models (Section 2.4.1), because *ad-hoc* interpretability often eases model interpretability, as the model is trained on sparse or intuitive features.

In-situ interpretability

In-situ interpretability is related to interpretable models. The model is *in-situ* interpretable when the explanations can be extracted directly from the model. We can obtain model interpretability on both *global*, *modular*, and *local* level. Lipton (2018) argued that a global interpretability connotes low computational complexity, and a human should understand the model by reading it in a reasonable time. Thus, only sparse models with few parameters can be considered globally interpretable. However, for the reason of accuracy-interpretability trade-off, a simple model can not always have good performance, thus are most of the time not suitable. Global level interpretable models are reviewed in Section 2.4.1.

To overcome this, modular level interpretability becomes interesting. Molnar (2019) claimed that modular level interpretability answers the question: "how do parts of the

model affect predictions?" Lipton (2018) proposed to call modular interpretability as decomposability: the model has modular level interpretability, if a human understands the mechanism of a part, whether it is an input, a parameter, or a calculation, during the classification. These components can be originally interpretable, or can be learned with constraints. Modular level interpretable models are reviewed in Section 2.4.2.

Although we can interpret an *in-situ* interpretable model at the modular or the global level, we can still extract directly the needed information from the model to explain a particular prediction. This makes the *in-situ* interpretable model always interpretable at a local level.

Note that in the interpretability literature, an *in-situ* interpretable model is also called a white-box model, an inherently/intrinsically interpretable model (Molnar 2019; Rudin 2019), an explainable-by-design model (Wang et al. 2020; Du et al. 2019), or an *ante-hoc* interpretable model (Rojat et al. 2021).

***Post-hoc* interpretability**

Techniques are applied on the prediction results of black-box models to provide explanations, with which we obtain model post-hoc interpretability. *Post-hoc* interpretability can be obtained both on global and local levels, as Guidotti et al. (2018) and Beaudouin et al. (2020) distinguished, "global explainability means the ability to explain the functioning of the algorithm in its entirety, whereas local explainability means the ability to explain a particular algorithmic decision." In practice, local level *post-hoc* interpretability is the most important. Local level means that the explanation of the black-box is in the vicinity of an individual instance, and explanation is usually an observation of the feature importance. A review of *post-hoc* interpretability techniques is provided in Section 2.5.

2.3 When does interpretability matter?

In this section, we discuss the following two questions with examples:

— If we have a powerful accurate model, do we still care about interpretability?
— In which situations don't we need interpretability?

2.3.1 Importance of interpretability

In this section, we present some examples about the reason why interpretability matters in machine learning context. Trust is first reason for using interpretable machine learning: it is fundamental to trust a prediction from a model before using it, and one tends to trust a white-box instead of a black one. We then show an example of Uber self-driving car to explain the importance of interpretability for safety reason. To avoid safety problems, we may apply interpretability to diagnose the model. In addition, model diagnosis provides other benefits, including bias finding in the dataset and debugging the model. As the bias can be easily spotted in interpretable models, interpretable models are more ethical and obey the demands in the GDPR regulation. Finally, interpretability could be helpful for finding the causality, such as "smoking is one of the causality of the lung cancer."

Trust: correct prediction with relevant features

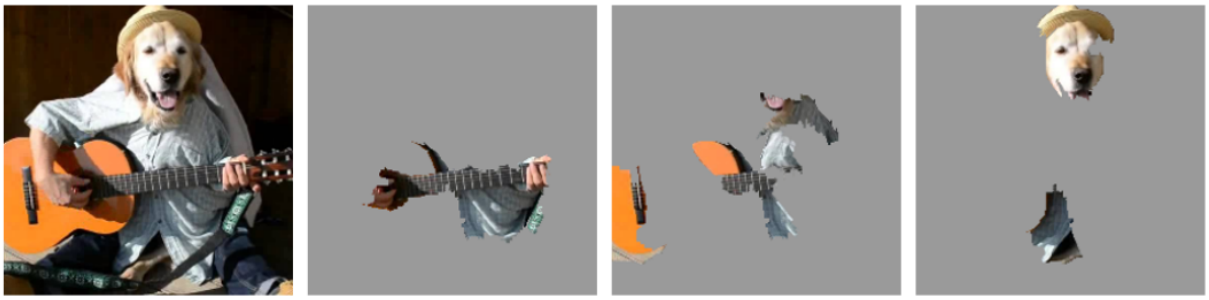
Many papers argue that trust is an important motivation for making machine learning models interpretable (Shen 2020; Lipton 2018; Ribeiro et al. 2016), because people can hardly trust the model as long as they do not understand how the model works by looking at the parameters learned from training examples. Arnold et al. (2019) proposed four pillars of trusted AI, and interpretability is a means of solving the *trust problem*.

In the point of view of interpretability, trust not only means "performance" (in many cases, the accuracy) of a model, nor the robustness that the model performs well with respect to the real scenarios. Without interpretability, one can hardly spot if the learned features make sense and thus can not trust the model.

Ribeiro et al. (2016) tested a prediction given by the Google's pre-trained Inception neural network (Szegedy et al. 2015). The reason of the good prediction of the Labrador dog is the head of the dog, as shown in Figure 2.2. This prediction along with the explanation provide trust to human, because a human can verify that the prediction is correct, and made based on right features.

Generally, trust is based on the explanations given by predictions: explanations for individual predictions is a solution to *trusting a prediction* problem, and a set of such explanations for different predictions is a solution to *trusting a model* problem [3].

3. It is called the instance level / model level interpretability in Section 2.2.1



(a) Original Image (b) Explaining *Electric guitar* (c) Explaining *Acoustic guitar* (d) Explaining *Labrador*

Figure 2.2 – LIME explanations on an image (Ribeiro et al. 2016) on the top 3 classes predicted by Google's Inception neural network: the relevant features formed through superpixels given by the interpretation provide trust to human.

Safety: do not cause damage

Sometimes, the machine learning model gives critical predictions for safety measures. In this context, we want to make sure that the system is secure and robust. However, the predictions made by black-box models are not always reliable, and this can cause serious damage. A well-known example is the self-driving Uber car which killed a pedestrian in 2018. This was believed to be the first pedestrian death caused by a self-driving car [4]. The uber self-driving vehicle software is composed with several ML models to accomplish the tasks of perception, prediction, and motion planning to control the vehicle [5]. When a pedestrian is spotted in front of the self-driving car, the object detection model in the car is supposed to make the stop prediction. In this particular example, the defect was that the self-driving vehicle did not have the capability to classify a pedestrian unless he/she was near a crosswalk [6]. The absence of the crosswalk caused finally this fatal accident. With interpretable models, in which the important features can be represented directly, engineers could have spotted the potential defects and could have avoided the accident (Shen 2020).

Model diagnosis: fix errors and find bias from dataset

Many machine learning models work well in a laboratory, but behave poorly in real life. One wants to figure out what is happening with the models by conducting model diagnosis and then improve it. Interpretable machine learning can be one approach to do

4. The New York Times: Pedestrain killed by Uber
5. Uber Machine Learning Infrastructure for Self-driving car
6. Self-driving Uber car that hit and killed woman did not recognize that pedestrians jaywalk

(a) Husky classified as wolf (b) Explanation

Figure 2.3 – Classifier failure (Ribeiro et al. 2016): a husky is classified as wolf because the classifier uses the snow from the background on the image as the discriminative feature.

this.

Ribeiro et al. (2016) provided another example about Google's Inception neural network. Given a Husky image, the model makes a wrong prediction as a wolf. For people who make a wrong prediction about dog/wolf, this may be caused by the similarity between a dog and a wolf. For this neural network, the problem comes from the snow on the background. The model takes the snow as the discriminative feature from the image, as shown in Figure 2.3. This is an example of a wrong prediction made based on wrong features from the image. With interpretable techniques, machine learners can find the reason why a model makes wrong prediction and fix bugs in the model.

Dataset plays an important role during the machine learning models' life cycle. One definitely does not want to use biased dataset to train machine learning models. A famous example of some biased data was the "tank problem" (Kanal et al. 1964). Scientists built an algorithm to detect military tanks from images. The model worked well for the tanks in test images (in the lab), but it failed with real photos in the field. Why? Because the model was focusing on wrong patterns from the images: in the training set, tanks are always emerging under clouds, and the model focused on the weather patterns, rather than the presence of tank. As long as one spots the bias from the dataset with interpretable machine learning techniques, the dataset can be improved.

Ethics: fairness and contestability

The predictions made by AI systems influence more and more in daily life, including the news we see, the loan we obtain from a bank, and even legal practices. As a widespread adopted technique in daily life, machine learning has to obey the code of ethics (Lipton 2018). New laws have been voted which impose a "right to explanation" in the General

Data Protection Regulation (GDPR). We do not give more details about regulations about explainability (see Goodman et al. (2017) and Beaudouin et al. (2020) for the legal concerns).

Medical diagnosis are often taken as an example for the importance of ethics in machine learning. Grote et al. (2020) believed that although machine learning algorithms sometimes outperform clinicians in medical diagnosis, it comes at the expense of transparency. Martens et al. (2011) pointed out that a doctor has to understand the predictions given by a machine learning model before using the model. This can only be done with interpretable models.

Pretrial risk assessment has been wildly used before the detention in the criminal justice system (Pretrial Justice Institute 2019). Recently, machine learning models has also been used in this process (Kehl et al. 2017). The purpose of the risk assessment is to mitigate judicial bias and to provide the decisions as objective as possible (Green 2020). This objectivity is called *fairness* in the justice system. However, the machine learning models can be biased (O'neil 2016; Barocas et al. 2016; Angwin et al. 2016), *e.g.*, the particular concern is that the algorithm may be discriminative against black people compared to white people, which is concerned unfair. In this context, interpretable machine learning is demanded to obtain fairness in the judicial trial.

Contestability is another important aspect of ethics, because people have the right to appeal to these decisions. Many end-to-end machine learning models make predictions based on the inputs, while the process is hidden in the black-box. When facing contestations, the black-box model cannot decompose itself into contestable components. Interpretability can help to solve this non-contestable problem by decomposing the components into a chain of reasoning (Chen 2020).

Causality: learn from the model and reuse knowledge

During the scientific research, scientists process the obtained data and explain the observed phenomenon with the data. During this process, machine learning algorithms become useful tools from chemistry (Burger et al. 2020) to geology (Audebert et al. 2019).

There are common pitfalls (not only for scientists) exist in machine learning (Riley 2019): (i) splitting data inappropriately, (ii) hidden variables (as they do not control some of the variables in the experiments), and (iii) mistaking the objective (the loss function). If scientists simply use black-box models without interpretability, they may fall into one of the pitfalls and give erroneous conclusions.

Usually, machine learning models are able to capture the correlations between variables, but not real causes. In addition to avoid the erroneous conclusions with interpretability, causal interpretable models can be even more interesting (than only interpretable models) for scientists. They are able to make causal inference by answering questions related to causality, *e.g.* "What is the specific feature that cause the prediction made by the model?". This property is helpful for scientists to understand the real causes of decisions made by machine learning algorithms (Moraffah et al. 2020). Scientists may understand better the mechanisms behind a phenomenon, and reuse the knowledge in the future research.

2.3.2 When it does not matter

Interpretability becomes less necessary when models are tested effective in real life (empirically trustworthy) and would not cause ethics or safety problems.

The first example is the recommendation system for business. A recommendation system (*e.g.* for movies) cannot cause ethics or safety problems thus the effectiveness (the performance) is more important than interpretability: the most serious problem that a recommendation system can cause is that the system is not accurate enough to recommend products/services based on the customer's flavor, and this may entail a loss of customers and of revenue. Note that interpretability is still demanded when a recommendation system is at ethics or safety issues (*e.g.* a recommendation system used to choose a key component for a satellite) and is still helpful for debugging the recommendation system.

The second example is based-on assistance dogs. They are trained to tackle the "object detection problem" in different scenarios. Sniffer dogs are trained to detect hazardous odour such as drugs or explosives. Guide dogs lead blind people to avoid obstacles. These dogs are black-box models, because their nerve system can be considered as "neural networks" with plenty of parameters. Nobody asked if the neural system of a dog is interpretable, because they are longtime used/tested in real life and empirically trustworthy. It is acceptable for a human (maybe except for neuroscientists) that the dog remains a black-box.

2.4 Interpretable vs. non-interpretable ML models

Arrieta et al. (2020) illustrated a *trade-off* between model interpretability and performance for most common machine learning models in Figure 2.4. This trade-off is caused by the complexity of the real world: sometimes simple models are not capable of approximating decision boundaries for complicated data, while complex models may have better discriminative power.

In this section, we distinguish models as *in-situ* interpretable and non-interpretable, and review the *in-situ* interpretable models both at global and modular level (cf. Section 2.2.1). Generally, regression models (linear and logistic regression), decision trees/rules are global interpretable models, nearest neighbor and Bayesian classifiers are modular interpretable models, and support vector machine, ensemble models, and deep neural networks are black-box models. We recall *post-hoc* techniques for explaining predictions of black-box models.

Note that we only categorize the models in a "general way", as the interpretability is correlated with the complexity of the model, a linear model with a thousand parameters is not more interpretable than a sparse deep neural network (Lipton 2018).

2.4.1 ML models interpretable at a global level

In this section, we discuss interpretable machine learning models at the global level. Global level interpretable models are at the right bottom corner in Figure 2.4, including linear/logistic regression, decision trees/rules. They are *usually* simple models (as long as the data does not have too many attributes) that human could understand easily.

Note that an globally interpretable model is also interpretable locally, because when we obtain a prediction for an input test instance from an interpretable model, we can also explain for this particular prediction from the model for this single input instance.

Statistical regression models: linear model and logistic model

Statistical regression analysis has been used to estimate the relationship between features and dependent variables. We discuss how statistical regression methods can help to augment interpretability in this section, including the variants of the least squares method (Legendre 1805): linear regression, multiple regression, log-linear regression, and logistic regression.

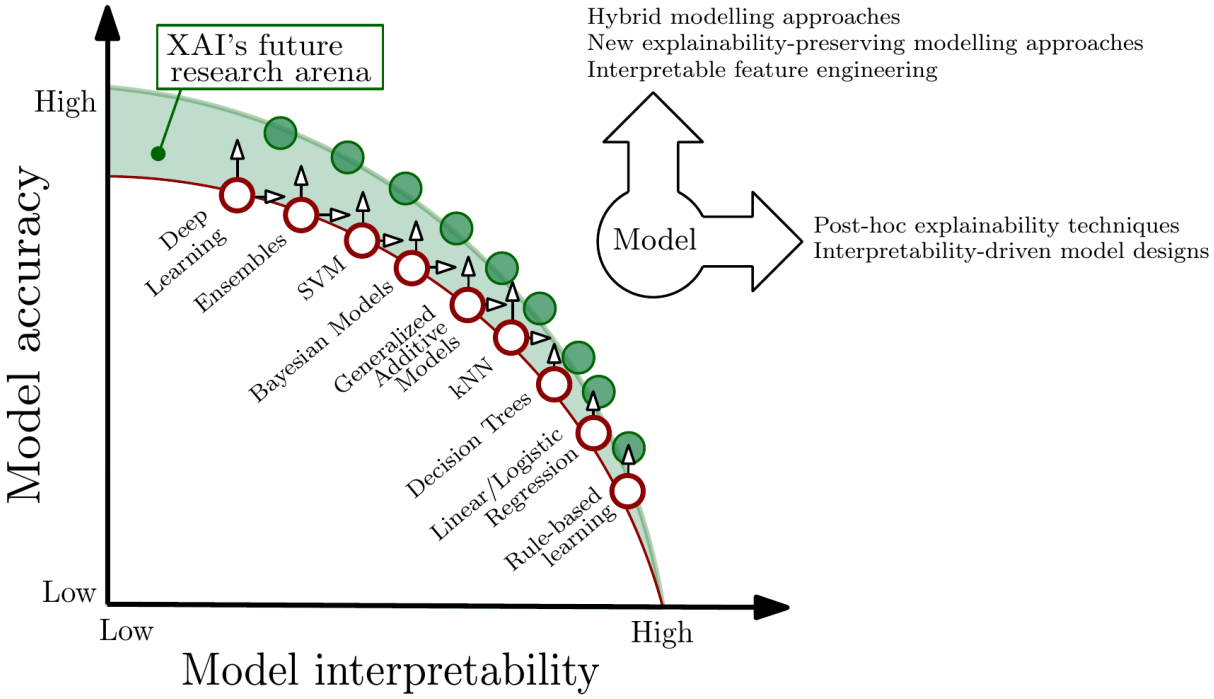


Figure 2.4 – Trade-off between model interpretability and performance (Arrieta et al. 2020).

Linear regression A linear regression model is used to draw relation between dependent variables and independent variables. It is written as:

$$y = w^\tau x + b.$$

The parameters of the linear model can be estimated with maximum-likelihood estimation by minimizing $-\log((x, y)|w)$. For linear regression with normally distributed errors, the loss function becomes a mean square error (MSE).

Molnar (2019) pointed out that linear regression models are simple enough, because they just learn a linear combination of the attributes, and the weights w describe the importance of each attribute in the prediction: with a positive w_i, it contributes positively to a prediction, and *vice versa*. A linear model is deemed interpretable because there is a direct relation between the weights and the features in the model. Note that, however, the interpretation based on the weights should be done carefully. As Lipton (2018) argued, with sufficiently high dimensionality, linear models, as well as decision trees and decision rules (as discussed later in this section) are not more interpretable than deep neural

networks.

Logistic regression We recall an adaptation of the generalized linear model to make logistic regression. A logistic sigmoid function [7], $f(z) = \frac{1}{1+e^{-z}}$ is used for logistic regression, then we have

$$y = \frac{1}{1 + \exp(-(w^\tau x + b))}$$

and

$$\ln \frac{y}{1-y} = w^\tau x + b. \tag{2.1}$$

The outcome of logistic regression y is a probability between 0 and 1. We can consider y as the predicted probability to the positive class, and the $1 - y$ the predicted probability to the negative class, then the ratio $\frac{y}{1-y}$ is called the odds. It describes the relative probability of the x to be a positive instance. When we take the logarithm of the odds, the term is called log odds (also called logit [8]).

The formula 2.1 means that the log odds from the logistic regression follows a linear model. To interpret the logistic regression model, Molnar (2019) proposed to study the impact of changing the x_j to $1 + x_j$ for the j-th element of x. We have the odds ratio

$$\begin{aligned}\frac{odds_{(x_j+1)}}{odds_{(x)}} &= \frac{\exp(w_1x_1 + w_2x_2 + \ldots + w_j(x_j+1) + \ldots + w_dx_d + b)}{\exp(w^\tau x + b)} \\ &= \exp(w_j(x_j+1) - w_jx_j) \\ &= \exp(w_j).\end{aligned}$$

Molnar (2019) pointed out that "a change in a feature by one unit changes the odds ratio (multiplicative) by a factor of $\exp(w_j)$, or in other words, a change in x_j by one unit increases the log odds ratio by the value of the corresponding weight."

Decision trees and classification rules

Decision trees are interpretable and useful models. A trained decision tree looks like a flowchart, each node is a conditional operation that compares a required feature with a cutoff value, called decision stump. When a data sample passes through, each node splits

7. Logistic sigmoid function is the most common example in the sigmoid function family, in which all the function has a sigmoid curve ("S"-shaped curve).

8. In neural networks, the logits is a vector that contains a raw (non-normalized) predictions of a classifier.

the data sample into sub-data. When the dataset cannot be split anymore, the decisions are made based on the terminal node. A simple decision tree is *in-situ* interpretable, because a human can understand the feature splitting process in a reasonable time by reading the feature with the cutoff value of each node.

Michie (1987) considered that when decision trees become too large, they can also be complex and thus less transparent. The tree is too deep or with a lot of nodes, a human cannot understand the decision process in a reasonable time.

Quinlan (1987) proposed several methods to reduce the complexity of decision trees. One of them simplifies the decision trees into decision rules. Like decision trees, the classification rules are also based on conditional operations. A terminal leaf from a simple decision tree corresponds to a decision rule:

$$\textbf{if } condition1 \wedge condition2 \wedge \ldots \wedge condition\ n \textbf{ then } classc$$

These rules can be used in different forms:

— IF-(ELSE)-THEN rules
— M-of-N rules
— Lists of rules

The rule-based classifiers are interpretable because it is readable for human beings. Fürnkranz et al. (2012) claimed that "rules offer the best trade-off between human and machine understandability".

Dimensionality reduction: feature extraction and feature selection techniques

Dimensionality reduction techniques use mathematical transforms to embed high dimensional data points into a subspace in which it is easier to calculate the distance between the data points. Among dimensionality reduction techniques, feature extraction is used to solve the curse of dimensionality, thus could be used to obtain *ad-hoc* interpretability, and feature selection is used to enhance *in-situ* interpretability by adding sparsity to the dataset and the model. Feature extraction techniques include principal component analysis (PCA) (Shlens 2014) and spectrum analysis (such as Fourier transform) (Von Luxburg 2007). For time series classification, PAA and SAX also belong to feature extraction methods. Feature selection methods include L1, lasso, and group-lasso (Yuan et al. 2006). It can also be tackled with neural networks (Doquet et al. 2019).

Note that dimensionality reduction techniques are not machine learning models, but

they are useful for *ad-hoc* interpretability and ease model interpretability (cf. Section 2.2.1). Not all dimensionality reduction techniques are useful for interpretability, i.e., technically, neural networks are also feature extraction techniques, but we consider neural networks, especially deep neural networks, non-interpretable, as discussed in Section 2.4.3.

2.4.2 ML models interpretable at a modular level

In this section, we introduce machine learning models interpretable at the modular level (cf. Section 2.2.1), *e.g.* naive Bayes classifier, k-Nearest neighbor method, and prototype-based models. The interpretation of these models are different compared to globally interpretable models. We emphasize prototype-based models in this section because it is highly related to our contribution.

Naive Bayes classifier

Based on the Bayes theorem, the class c of a given example x, can be calculated as:

$$P(c|x) = \frac{P(c)P(x|c)}{P(x)}$$

where $P(c)$ is the prior probability of class c, $P(x|c)$ is the likelihood of a sample x relative to class c. It is difficult to obtain the likelihood $P(x|c)$ because one cannot estimate this probability from all the attributes of finite training set. The naive Bayes classifier adopts the attribute conditional independence assumption, i.e., it is naive to assume that each attribute contributes independently to the classification result. The naive Bayes classifier can be written as:

$$P(c|x) = \frac{P(c)}{P(x)} \prod_{i}^{d} P(x_i|c)$$

where d is the number of attributes, and x_i is the i-th attribute from the example $\boldsymbol{x}$.

Naive Bayes classifier is interpretable because it can be interpreted on the modular level (Molnar 2019). The contribution of each attribute to a certain class is the conditional probability, and thus we can interpolate the behavior of the model from the contribution of the attributes.

k-nearest neighbor

The k-nearest neighbor (kNN) method is a common supervised learning method. The mechanism is straightforward: given a test example, the model calculate a distance to find the k nearest neighbors from the training set, and give the prediction based on these k nearest neighbors. For a classification problem, the k nearest neighbors will vote for the class. For regression, it takes the average value of these k neighbors. One does not train the model, but just store the training examples. To build the model, we only need to choose the k value and a dedicated metric.

kNN model does not have any parameters to learn, and it compares an example with its k-nearest neighbors. The model is inherently local thus it cannot be interpreted globally (Molnar 2019). The interpretability of kNN models relies on the neighbors of a given example (Moraffah et al. 2020). Note that for a kNN model, the "curse of dimensionality" (Bellman 1957) exists for the interpretation. As other interpretable models, a kNN model could become non-interpretable when the data has a thousand attributes.

Prototype-based methods

Prototypes form a minimum set of representative samples of a dataset. Traditionally, prototype-based methods can be classified into two categories in the literature: prototype selection and prototype generation (García et al. 2012; Triguero et al. 2012). Both of the methods form the prototype reduction methods, aiming at reducing the representative instances of the dataset by prototypes. Prototype selection methods belong to instance selection methods, which reduce the samples of the dataset into a small subset. Prototype generation methods generate new representation data from the existing dataset, in order to increase the generalization ability of the small amount of prototypes.

Prototype-based methods are very intuitive, because they make use of explicit representations, i.e., the prototypes of the dataset, and it compares directly the "distance" of the prototype(s) and the test instance (Biehl et al. 2016). Providing the domain experts with this small amount of prototypes, i.e., representative samples of each class, helps increase interpretability (Bien et al. 2011).

Nowadays, prototype-based classifiers learn prototypes from the dataset, and compare the "distance"[9] between class prototypes and the object to be classified. This approach is strongly linked to kNN, with a reduction in time complexity (Bien et al. 2011; Impe-

9. Can be distance, similarity, or dissimilarity (Saralajew 2020)

dovo et al. 2014; Hu et al. 2016), because once the prototypes are fixed, we can simply compute the distance between an example to be classified and the prototype(s) instead of comparing with all the instances from the dataset. An ideal prototype should have two properties: diversity and coverage (of the dataset) (Gee et al. 2019). Diversity means that the prototype(s) of one class c should consist of instances that are close to the training instances from class c, and far away from training points from other classes (Bien et al. 2011). Diverse prototypes will help focus on areas, where there are overlapping class regions, to improve classification accuracy (Gee et al. 2019). The coverage property means that the prototype(s) should also provide full coverage of the training set, i.e., every class should have at least one prototype.

As the neural networks draw great attention in the research community, prototype neural networks have been developed (Li et al. 2018; Chen et al. 2019; Gee et al. 2019). These prototypes can be learned directly from other types of black-box models, and can provide *in-situ* interpretability on the modular level.

Li et al. (2018) provided a network architecture called prototype network to learn the prototypes through an end-to-end training. The proposed network contains an auto-encoder that encodes the input vector into a latent representation; then the encoded vector is passed into a classifier, which contains a prototype layer, i.e., the layer holds the prototypes, as its first layer. The network learns the prototypes in the latent space, and as the prototypes have the same dimension of the latent representation of the input vector, users could use a decoder (from the auto-encoder) to visualize the prototypes. Gee et al. (2019) improved the prototype network (Li et al. 2018) by adding a penalty in the objective function in order to increase the prototype diversity and coverage of the data in the latent representation. This work adapts the prototype network for univariate time series classification tasks on medical data, and provides a tool to engage domain experts to fine-tune the model to increase interpretability.

Chen et al. (2019) proposed a new network architecture called prototypical part network (ProtoPNet). The spirit is similar to the work of Li et al. (2018). The difference is that, in the work of Li et al. (2018), the prototypes are in the latent space, thus should be visualized by a decoder, but in Chen et al. (2019), the prototypes are directly sub-images. During the prediction process, the ProtoPNet compares the parts on the image which are similar to the learned prototypes, and the prediction is based on the similarity between the sub-image and the prototypes. Their method uses the learned prototypes for making prediction, thus the model is considered interpretable (also called explainable-by-design

because the model is *in-situ*). The prototype-based interpretability intends to look like global representations of the training data, but *post-hoc* interpretability does not give the intuition of the relation between the training data and the end result (Gee et al. 2019).

2.4.3 Non-interpretable ML models

In this section, we discuss non-interpretable machine learning models. They are usually complex models, *e.g.* SVM, ensemble methods, and deep neural networks. As shown in Figure 2.4, there exists a trade-off between the model complexity and its interpretability. The more complex a model is, the more capacity it will have to approximate a complex function, and the more difficult it is to interpret.

Support vector machines (SVMs)

The objective of support vector machines (SVMs) is to find hyperplane(s) from the sample space of the training set to separate different classes. The hyperplane can be described by a linear equation:

$$w^\tau x + b = 0$$

where w is the normal vector of the hyperplane, which decides the orientation of the hyperplane, and b the bias, which decides the distance between the hyperplane and the origin. This hyperplane is "supported" by the vectors that are perpendicular to the hyperplane and point to the nearest samples from different classes.

However, when a dataset does not have the linear separability in the sample space, we need to modify SVM to classify the dataset. Boser et al. (1992) changed the SVM decision function into the following linear combination:

$$f(x) = w^\tau[\kappa(x, x_1); \kappa(x, x_2); \ldots; \kappa(x, x_n)] + b$$

where $\kappa(\cdot, \cdot)$ is the kernel function that maps the sample from the origin sample space to a higher dimensional space, in which the instances from different classes becomes linearly separable.

The original features could be understandable for human beings, but the features in the higher dimensional space are not. The interpretation of SVMs given by machine learners is always *post-hoc* and can be written in the form "the higher an input feature x_i is, the more likely the class is positive" (Rüping 2006).

Ensemble learning methods

An ensemble learning model is also called a multi-classifier system. Ensemble learning methods contain two steps: firstly, ensemble learning learns a group of individual learners, then it combines the predictions of the individual learners to get the final prediction (Zhou 2012). The individual learners can be homogeneous, i.e., they are the same ordinary machine learning method, or heterogeneous, i.e., individual learners are combined with different models.

Based on the generation process of the individual learner, ensemble methods can be separated into two categories. The first one is Boosting: the individual learners are trained sequentially such that each model that is added to the ensemble aims at correcting errors from the ensemble made of all previously introduced models. The second one is Bootstrap AGGregatING (bagging): the individual learners are learned independently on different subsets of the data, and the learning process can be parallelized.

Ensemble method can often get better generalization performance than individual model (Dietterich 2000). However, the interpretability of the ensemble model is sacrificed. For example, a single decision tree could be interpretable (as discussed in Section 2.4.1), but a random forest may contain a thousand trees, thus makes it hardly understandable by a human.

Deep neural networks (DNNs)

In practice, deep neural networks (DNNs) are used for better performance. The connections of DNNs can be considered as numerous non-linear functions (Telgarsky 2016). Intuitively, the multiple layer non-linear functions allow the network to learn features from different representation of the raw data to make the decision (Camburu 2020).

However, DNNs are usually used for end-to-end task, and the features learned by DNNs are sometimes not make sense to humans (Olah et al. 2018). These DNNs can neither be understood on global nor on modular level. Just like the ensemble methods, the increased number of layers and neurons makes DNNs difficult to interpret. In addition, state-of-the-art DNNs (He et al. 2016; Vaswani et al. 2017) may have millions of parameters, and in general, we do not know which parameter is more important than others.

There are two ways to make DNNs understandable: (i) Giving explanations to their decisions with *post-hoc* interpretability techniques (cf. Section 2.5), (ii) Training the DNNs with constraints (cf. Section 2.4.2).

2.5 *Post-hoc* interpretability

For non-interpretable models, we need to apply *post-hoc* interpretability techniques to explaining *a posteriori* a complex trained model. *Post-hoc* interpretability is obtained with explanations given by these *post-hoc* techniques. *Post-hoc* interpretability is used for both global and local levels. For simplicity, we will use "explanation" for local *post-hoc* interpretability.

In Section 2.5.1 we review the basic idea of the global *post-hoc* interpretability and explain why it cannot be largely applied. In Section 2.5.2, we recall local *post-hoc* explanation. Local explanation focuses on an input test instance, i.e., once a non-interpretable model is trained, we do not modify the model but apply perturbations to the input in order to observe which feature of the input is important for the prediction. Local explanation methods are largely applied in the practice because they can provide intuitive explanations to reinforce model interpretability.

2.5.1 Global *post-hoc* interpretability

Global *post-hoc* interpretability is also called global surrogate explanations. We can learn a surrogate interpretable model (cf. Section 2.4.1), with the input and output of a learned black-box model, to mimic the behaviors of the black-box. These surrogate models can be decision trees or decision rules that are extracted from black-box (Domingos 1998; Craven et al. 1995; Johansson et al. 2009) or "white-box" mathematical functions understandable by human that are expressed through a succinct symbolic function (Alaa et al. 2019). This surrogate model can neither really reveal the mechanism of the black-box, nor give similar prediction performance as the black-box, which may cause fidelity [10] problems (Rudin 2019). In practice, if a surrogate interpretable model has comparable performance as the performance of a black-box model, we could just abandon the black-box and use the interpretable model.

2.5.2 Local *post-hoc* interpretability

In this section, we review techniques that are applied to obtaining explanations for local *post-hoc* interpretability. We can divide existing methods that dedicated to obtaining local interpretability into three categories, as shown in Figure 2.5.

10. Fidelity is the ability of the explanations that reflect the behavior of the black-box model.

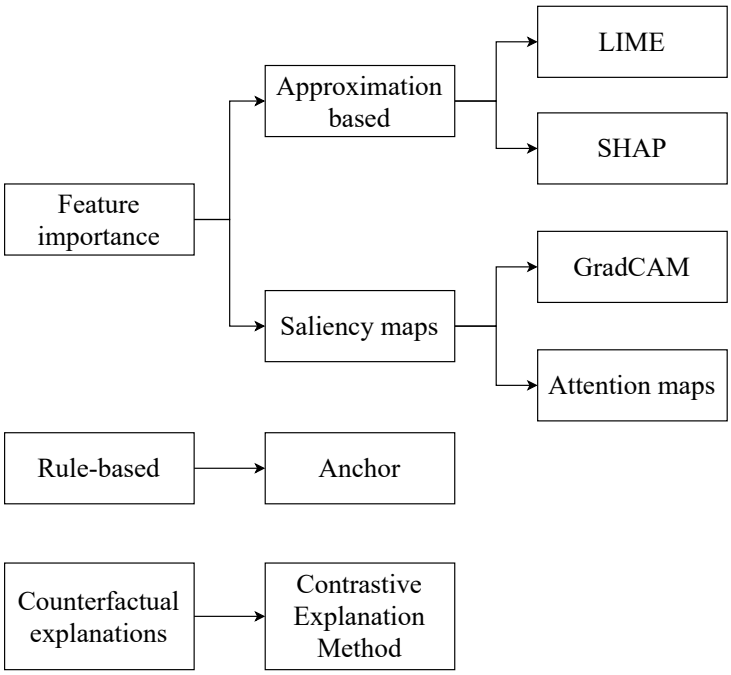


Figure 2.5 – Local *post-hoc* interpretability methods.

Among these methods, feature importance is the most widely used. For approximation techniques, a feature importance score is assigned to each feature, and for saliency maps, a heat map, that indicates the regions that are used for the prediction result, is generated on the input instance. Rule-based methods and counterfactual explanations produce intuitive explanations to humans with the rules and counterfactual examples.

Feature importance

Given the prediction of a black-box model for an input instance, feature importance methods point out which features of the input are important for the prediction. The feature importance of an input instance of a model is obtained by perturbing its attributes.

Approximation: LIME and SHAP Local Interpretable Model-agnostic Explanations (LIME) (Ribeiro et al. 2016) and SHapley Additive exPlanations (SHAP) (Lundberg et al. 2017) are two prominent local approximation techniques that give the feature importance for an input instance with a score for each feature. The local approximation gives human an intuition about the model.

LIME assigns a feature importance value for each feature for an individual prediction. LIME tries to find the most suitable interpretable model (for instance a linear model)

based on the perturbations of an original input instance, which formulates the local approximation near the input instance.

The mathematical expression of an LIME explanation ξ for a black-box f can be written as:

$$\xi(x) = \arg\min_{g \in \mathcal{G}} L(f, g, \pi_x) + \Omega(g)$$

where the model g is an explanation in the potential interpretable model family $\mathcal{G}$, L measures the unfaithfulness of the explanation, kernel $\pi_x(z) = \exp\left(-\frac{D_E(x,z)^2}{\sigma^2}\right)$ defines the locality between neighborhoods z and the instance x with a hyperparameter σ, and $\Omega(g)$ the complexity of g (the regularization term). The generated linear model has only a few non-zero coefficients.

The LIME method can be used for tabular data, text and images. LIME needs to cut the input data into interpretable components and train the linear model on these components, For tabular data, the interpretable components could be attributes, for text they could be words, and for images they could be superpixels. Once the linear surrogate model is built, the coefficients of the linear model indicates the importance of the input attributes. The original LIME method is not suitable for time series, because natural segmentation does not exist on time series for interpretable components. An adapted LIME method (called LEFTIST) for time series data is proposed by Guillemé et al. (2019b), as introduced in Section 3.2.

SHAP is a local model-agnostic explanation method based on game theory. The idea is similar to LIME (and other approximation explanation techniques), i.e. SHAP assigns to each feature an importance score for an individual prediction. Different from LIME, SHAP computes the Shapley value (Shapley 1953) as the feature importance value, i.e. Shapley values fairly distribute the prediction among the features.

Heatmap-based methods: Grad-CAM and attention map Heatmap-based methods are visualization techniques that superpose a heatmap to highlight features in an input (*e.g.* images or text). They are useful for finding the attention regions of an individual prediction. A large family of heatmap-based methods has been developed recently, including class saliency visualization (Simonyan et al. 2014), the class activation maps (CAM) family (Zhou et al. 2016; Selvaraju et al. 2017; Chattopadhyay et al. 2018), and the attention maps family (Xu et al. 2015; Choi et al. 2016; Lei 2017).

As an end user could not select one of these methods only based on the visual appeal of the image, Adebayo et al. (2018) evaluated the quality of different saliency maps

with two testes, the model parameter randomization test (randomize the parameters of a model to check a saliency map method depends on the parameters of a model), and the data randomization test (randomize the labels of the inputs to check if a saliency map method depends on the that the model was trained on). They tested on Gradient Input (Shrikumar et al. 2017), Integrated Gradients (Sundararajan et al. 2017), Guided Backpropagation (Springenberg et al. 2015), Grad-CAM / Guided Grad-CAM (Selvaraju et al. 2017), and Smooth Grad (Smilkov et al. 2017). Results show that Gradients and Grad-CAM pass the tests, while Guided BackProp and Guided Grad-CAM fail both (others fail at least one of the checks).

Among these saliency maps, the most successful saliency maps could be Grad-CAM (Gradient-weighted Class Activation Mapping) (Selvaraju et al. 2017). It is a very popular method used in computer vision to understand which parts of an original image is used by a trained neural network to make a particular classification decision. Grad-CAM and Grad-CAM++ are considered generalizations of the original CAM method (Zhou et al. 2016). The idea of Grad-CAM is to put a heat map on the original input of the neural network to highlight the regions (features) that the neural network looks at for the decision.

Attention mechanism was firstly introduced by Bahdanau et al. (2015) for the neural machine translation task, then attention networks have met a great success in natural language processing (NLP), object detection, and other tasks. Attention scores are learned to reveal the inner relation between features. As a built-in component, attention maps are available by default in attention-based networks, and could be used directly to provide the feature importance.

There are some disagreements on whether an attention map could be used as an explanation. Jain et al. (2019) pointed out that "attention is not explanation" because the relationship between attention weights and model predictions is not clear, and the attention map is not always the same as the gradient-based saliency map. However, Jain et al. (2019) were challenged by a paper entitled "attention is not not explanation" (Wiegreffe et al. 2019). They point out that the argument "attention is not explanation" is based on one's definition of "explanation" and propose different tests to prove that the attention map can be an explanation.

We believe that an explanation is nothing but an observation, and we use interpretability as a generalized term, thus, we adapt the idea of Gilpin et al. (2018) in this thesis: if an attention map of a test sample matches the attention of human, then it can be a good

explanation.

Rule-based explanations: Anchors

Sometimes, linear explanations, such as the ones given by LIME, could lead to poor performance because the decision boundary of a complex model in the vicinity of an input instance could be highly non-linear. An anchor provides an explanation with sufficient conditions (rules) for the prediction (Ribeiro et al. 2018). Anchor explanation is effective at explaining non-linear decision boundaries of the input example. More importantly, the produced rules can often be generalized to many instances other than a given input instance.

Counterfactual: Contrastive explanation method

Counterfactual explanation methods try to find a counterfactual example x' with the smallest change to the feature values of an original input x, but leading a different prediction y' instead of a predefined prediction y.

Dhurandhar et al. (2018) argued that counterfactual explanations are natural for humans and propose a contrastive explanation method (CEM) that generates pertinent positives (PP), i.e., features should be minimally and sufficiently present, and pertinent negatives (PN), i.e., features should be minimally and necessarily absent, from the input instance as its explanations. The quality of the modified x' is measured with an autoencoder by evaluating the closeness of the data manifold.

However, for an instance to be explained, counterfactual methods may find several counterfactual examples, which may contradict each other (also called Rashomon effect), and it is also possible that the counterfactual may find no example for a given tolerance value of the "small change". These situations may cause inconvenience for final users.

2.6 Summary

In this chapter, we first discussed the definition and the necessity of interpretability in the context of AI. As there exist ambiguities between the terminology of interpretability and explainability in the literature, we reviewed the different definitions and decided to use model interpretability as a general term, i.e., a model is interpretable as long as it is understandable by a human. We need interpretability not only because of the potential

harm (e.g. trust and ethic problems) that a non-interpretable machine learning system may cause, but also because of its abilities to debug the model or to make causal inference by answering questions related to causality.

The interpretability concept contains many aspects. A model can be understood in either global, modular, or local level, and based on the moment that we interpret the model, i.e., before training, during a prediction, after a prediction, we can have *ad-hoc, in-situ* and *post-hoc* interpretability. Note that although global *post-hoc* interpretability exists, it is not very helpful in practice (cf. Section 2.5.1), thus *post-hoc* interpretability is obtained with an explanation at the local level.

Global level interpretability comes with the sparsity of the model, and modular level interpretability means that parts of the model is understandable by human. Regression model, decision trees, and classification rules are interpretable at the global level. Naive Bayes, kNN, and prototype-based models are interpretable at the modular level. For black-box models, *post-hoc* methods are applied to understand them locally to an input instance of the model.

Part II

Interpretable time series classification

Chapter 3

Interpretable time series classification under a unified framework

Many time series classification (TSC) methods, especially ensemble methods and neural networks, are focused on getting better performance in accuracy. However, the state-of-the-art TSC methods are considered as black-box models (cf. Chapter 2.4) and not deemed interpretable. We need interpretable TSC, because TSC can be applied in critical decision-making process and thus cause serious problems (cf. Section 2.3).

In Section 3.1, we focus on an analytical framework to benchmark interpretable time series classification methods. The main part of this framework was proposed by Fauvel (2020). It is called a *performance-explainability framework*. The framework of Fauvel (2020) is focused on information about the explanations given by *post-hoc* techniques. Different from Fauvel (2020), we adapt it to emphasize *in-situ* interpretability for time series classification. The modified framework is called *performance-interpretability framework*. In Section 3.2 we review the context of interpretability in time series classification, specifically the *post-hoc* explainable methods for TSC models, and we instantiate the performance-interpretability framework to compare existing explanations for TSC tasks.

3.1 Performance-interpretability framework

As in Definition 3, in this thesis we emphasize *comprehensibility* as the most important property for interpretable time series classification. We are inspired by the literature in interpretability (Arrieta et al. 2020; Rojat et al. 2021), especially the framework of Fauvel et al. (2020b), and we adapt this framework to answer the following questions:

— How does the model perform in the state-of-the-art?

— How is the model interpretable?

— What is the scope of its interpretability (granularity)?
— Are the explanations faithful?
— Which kind of information do the explanations provide?
— What is the target user category of the explanations?

The details of the components that answer these questions are discussed in the following of this section. Inspired by Fauvel et al. (2020b), we present this framework in parallel coordinates plot in order to compare different methods, as shown in Figure 3.1.

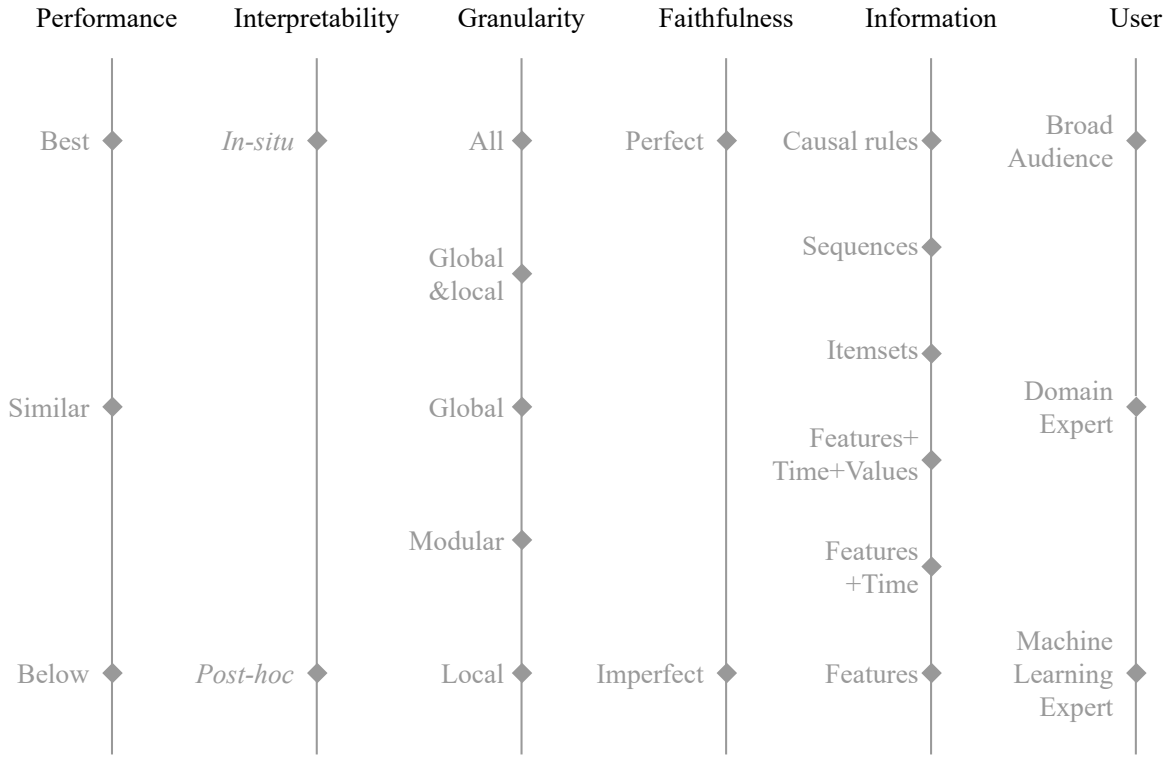


Figure 3.1 – Performance-interpretability framework for TSC: there are 6 dimensions. Every single model improves between the bottom and the top of a given dimension.

3.1.1 Performance

The level of performance of the model is the first component in the framework. Different metrics (*e.g.* accuracy, F-score, receiver operating characteristic (ROC) curve, ...) exist to evaluate the performance of a machine learning model. The choice of a metric to evaluate the performance of a machine learning model depends on the application.

In this thesis, we focus on TSC tasks and we use the accuracy score as the performance, because it is the most widely used metric for classification tasks. Note that the evaluation setting could be changed. The performance of a tested model is then categorized into 3 categories:

— Best: the tested model has a comparable performance as state-of-the-art models;
— Similar: the tested model performs similarly to the compared methods;
— Below: the performance of the tested model is below the performance of the compared methods.

3.1.2 Model interpretability

"How is the model interpretable?"

Different from the comprehensibility proposed by Fauvel et al. (2020b), a model can be *in-situ* or *post-hoc* interpretable by Definition 3: an *in-situ* interpretable model is self-explainable, and a black-box model needs *post-hoc* explanations.

3.1.3 Scope of interpretability (granularity)

Scope is also important for the quality of interpretability. Different from Fauvel et al. (2020b), we modified the granularity of an interpretation, as discussed in Section 2.2.1: (i) for in-situ interpretable models, the interpretability is at the global level when the model is simple or sparse (cf. Section 2.4.1), and is at the modular level when one can understand a part of the model (cf. Section 2.4.2); or it can also be interpretable at the local level; (ii) for black-box models that need *post-hoc* explanations, the interpretability is at the global and/or local level.

3.1.4 Faithfulness

The faithfulness is an important property of explanations. There exists a variety of terms related to faithfulness (*e.g.* fidelity (Guidotti et al. 2018), trustworthiness (Camburu 2020)). It refers to how accurately an explanation reflects the true reasoning process of the model (Wiegreffe et al. 2019). Since explanations provided by surrogate models cannot be perfectly faithful to the original model (Rudin 2019), we consider the faithfulness proposed by Fauvel et al. (2020b) good enough for us to compare explanations:

— Perfect: an explanation extracted directly from the original model is perfectly faithful;

— Imperfect: an explanation extracted from surrogate models is imperfectly faithful.

3.1.5 Information type: causality, patterns, and feature importance

"Which kind of information does the explanation provide?"

Different explainers may provide different forms of explanations (cf. Section 2.5), *e.g.* saliency maps (Selvaraju et al. 2017), linear models (LIME) (Ribeiro et al. 2016), decision rules (Ribeiro et al. 2018), counterfactual examples (Dhurandhar et al. 2018), etc. These forms of explanations provide different types of information (in the order from the best to the worst):

— Causal rules / counterfactual examples;

— Patterns: sequences and itemsets;

— Feature importance: features+time+values, features+time, and features.

Among these explanations, the most valuable information provided by an explainer are *causal rules* and *counterfactual examples* (Pearl et al. 2018). Causal rules provide the true cause of some model predictions (cf. Section 2.3.1), and counterfactual examples provide contrastive explanations (cf. Section 2.5.2). However, most of the time, machine learning models are only capable of discovering correlations, which may not provide an intuition of possible causal relations, among the data (Arrieta et al. 2020).

Besides causal rules and contractual examples, the most intuitive explanations resemble the information we perceive in day-to-day life, such as visual information (images), human natural language (text, sound), smell, etc. The information provided by an explanation can be categorized into two types: *patterns* and *feature importance*. Patterns are frequent features in a dataset, and feature importance is the relative importance of the features for a prediction.

Pattern-based explanations are better than *post-hoc* feature importance explanations. In the paper "This looks like that: deep learning for interpretable image recognition", Chen et al. (2019) proposed to learn prototypes, which are representative patterns in a dataset, to make predictions. These prototypes are constrained by training examples from the dataset. Their prototype-based method is considered *in-situ* interpretable. Two types of patterns can be given as information (Fauvel et al. 2020b):

— Itemsets: "the explanations provide patterns under the form of **unordered** groups of values, also called itemsets."

— Sequences: "the explanations provide patterns under the form of **ordered** groups of values, also called sequences." Note that shapelets are naturally subsequences of time series thus belong to sequences.

Feature importance explanations are extracted with LIME or Grad-CAM methods (cf. Section 2.5.2). For TSC task, Fauvel et al. (2020b) categorized feature importance into three groups:

— Features: the explanations only provide features relatively more important than others;

— Features+time: the explanations provide relative important features and the occurring timestamps;

— Features+time+values: the explanations provide relative important features, the occurring timestamps, and the discriminative values of a feature for each class.

3.1.6 Audience

"Is the explanation fit for broad audiences, domain experts, or machine learning experts?"

Fauvel et al. (2020b) propose to categorize the target users into three groups, from the most general to the least:

— Broad audience: non-domain experts (*e.g.* policymakers);

— Domain experts (*e.g.* researchers, physicians, doctors, electricians, traders, ...);

— Machine learning experts.

The more intuitive an explanation is, the more general the target users could be. The visual patterns (in images) could be good representations of the data for broad audiences, while a cardiologist (as a domain expert) could understand the shapelets in ECG data.

3.2 A comparison of *post-hoc* explanations for TSC

Grad-CAM and LIME are two famous methods for *post-hoc* explanations in machine learning (cf. Section 2.5). Both methods have been adapted for explaining time series classification results. We compare in the performance-interpretability framework for two *post-hoc* interpretability techniques in this chapter (namely CAM and LEFTIST), as shown in Figure 3.2.

As explained in Section 2.5.2, many *post-hoc* techniques superpose a heatmap on

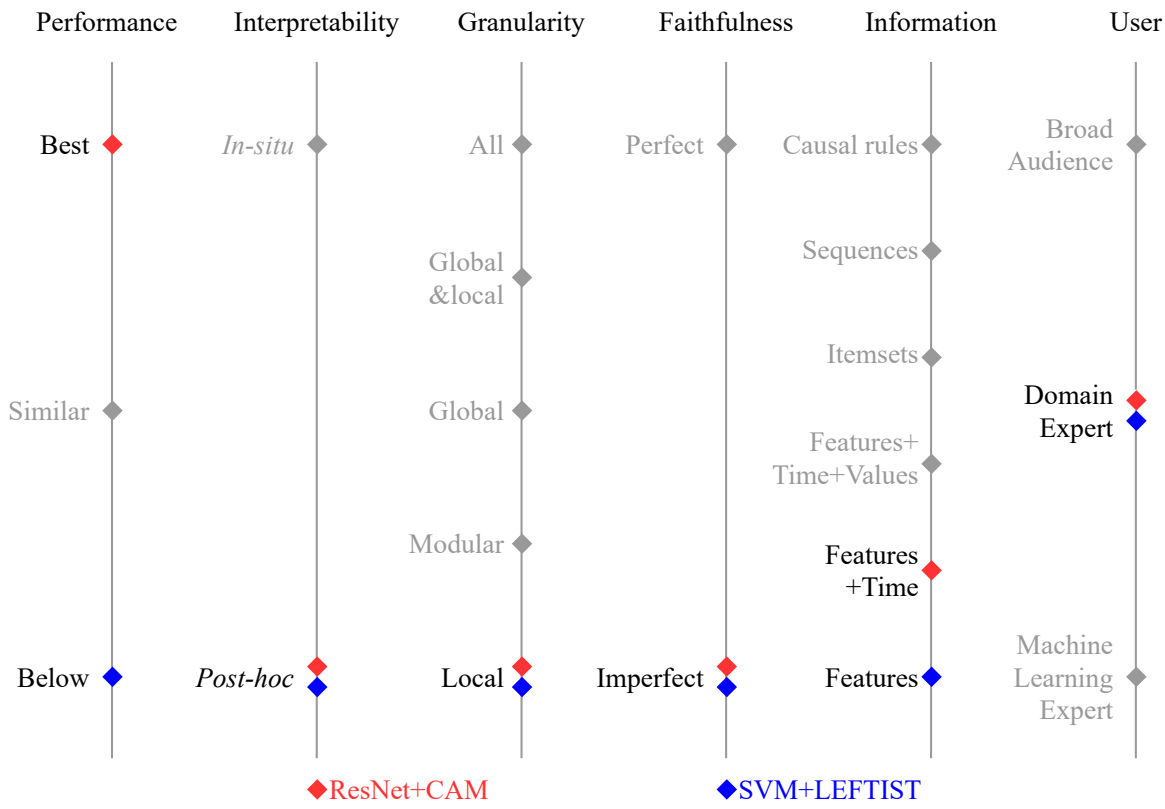
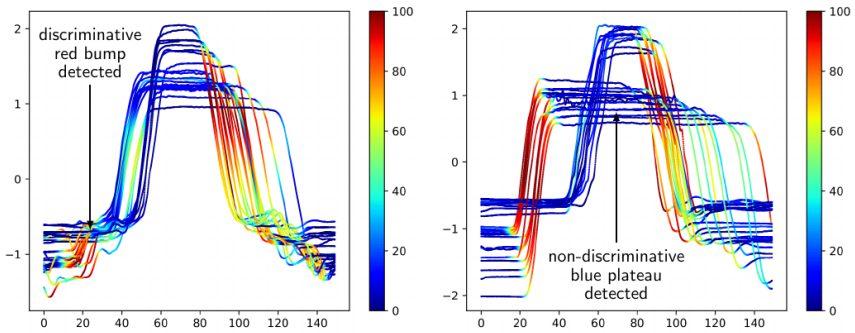


Figure 3.2 – Comparison of TSC methods with their *post-hoc* explanations for different classifiers namely ResNet (Fawaz et al. 2019) and SVM (Guillemé et al. 2019b).

the input of the model as their explanations. The idea is to highlight the important regions of an input (*e.g.* an image or a sequence of text) to show the feature importance. Heatmap-based techniques are applicable to almost all TSC problems, and they highlight the discriminative regions on the time series. Based on our framework, the discriminative regions provide the *features+time* information. Many recent works used heatmaps on the input time series as their explanations, *e.g.* (Le Nguyen et al. 2019) (the discriminative regions were highlighted), especially neural network-based methods (Fawaz et al. 2019). Figure 3.3 illustrates a CAM-based explication for ResNet classifier from Fawaz et al. (2019).

LIME is another famous method that applies perturbations on the original input of the model. As discussed in Section 2.5.2, the interpretable components (the components used as features) of LIME can be superpixels for images, attributes for tabular data, and words for text, but it can be difficult to determine the interpretable components in a



(c) ResNet on GunPoint: Class-1 (d) ResNet on GunPoint: Class-2

Figure 3.3 – CAM-based explications for a ResNet classifier for GunPoint dataset: a heatmap is superposed on each time series to highlight the discriminative parts (Fawaz et al. 2019).

time series. Guillemé et al. (2019b) adapted LIME for time series classification task and proposed a method called Local Explainer For TIme Series classificaTion (LEFTIST). The basic idea of LEFTIST is to use prefixed (both the length and the position) shapelets as the interpretable components, and provide the feature importance of each shapelet. Guillemé et al. (2019b) claimed that this idea corresponds to superpixels in computer vision. Figure 3.4 illustrates an explication given by LEFTIST Guillemé et al. (2019b).

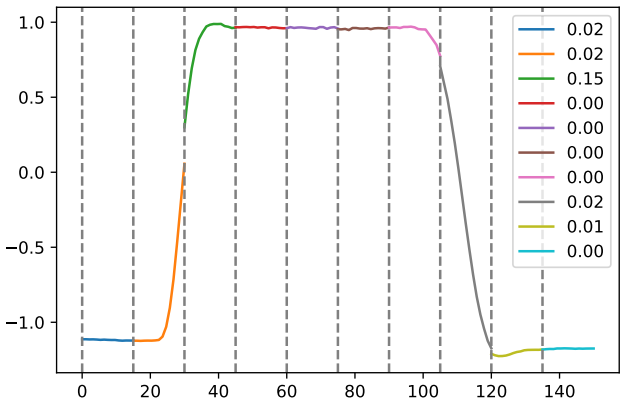


Figure 3.4 – LIME-based explication for an SVM classifier for a time series belongs to class 1 from GunPoint dataset: the time series is separated into 10 shapelet segments, and each value indicates the contribution of corresponding shapelet segment (Guillemé et al. 2019b).

However, we agree with Rudin (2019) on the fact that *post-hoc* explanations could be replaced by *in-situ* interpretability, because *post-hoc* explanation can only reflect but can not change the behavior of a model, while constrained interpretable components of the model could ensure the safety and trust of this model.

3.3 Summary

In this chapter, we adapted the *performance-explainability* framework of Fauvel et al. (2020b). We emphasized the *in-situ* interpretability as defined in Definition 3, and we regroup the granularity of the interpretability with *in-situ* and *post-hoc* interpretability. This framework can be used for benchmarking the performance of a model, the model interpretability, and the quality of explanations for time series classification. We examined two famous *post-hoc* explanation techniques for TSC tasks (namely CAM and LEFTIST) in our framework to compare them. They both provide *post-hoc* local interpretability, which is often considered unfaithful. The only difference between them is that CAM explanations provide both feature importance and the occurring time stamps, while LEFTIST provide feature importance at prefixed time stamps.

Chapter 4

In-situ INTERPRETABLE TIME SERIES CLASSIFICATION

In this chapter, we make use of a simple convolutional network to classify time series, and we show how one can leverage the principle of adversarial learning to regularize the parameters of this network such that it learns shapelets that could be more useful to interpret the classifier's decision. We detail our eXplainable-by-design CNN (XCNN) method [1] in Section 4.1, and we show quantitative and qualitative results on the usual time series benchmarks (Chen et al. 2015) in Section 4.2. Our XCNN performance are on a par with comparable state-of-the-art methods, and comparing to existing *post-hoc* explanations, we provide new types of explanations for neural networks' predictions.

4.1 Method

In this section, we present our approach to learn interpretable discriminative shapelets for time series classification.

Our base time series *classifier* is a Convolutional Neural Network (CNN). Closely related to shapelet-based methods (as stated in Section 1.3.2 and Section 3.1), variants of Convolutional Neural Networks (CNN) have been introduced for the TSC task (Wang et al. 2017). These are mostly mono-dimensional variants of CNN models developed in the Computer Vision field. Note however that most models are rather shallow, which is likely to be related to the moderate sizes of the benchmark datasets present in the UCR archive (Chen et al. 2015).

Both learning shapelets (LS) and CNN slide the shapelets on the series to compute local (dis)similarities. The main difference between the classifier of LS and that of our method is the (dis)similarity between a shapelet and a series. LS uses a squared Euclidean

1. An explainable-by-besign model is an *in-situ* interpretable model (cf. Section 2.2.1).

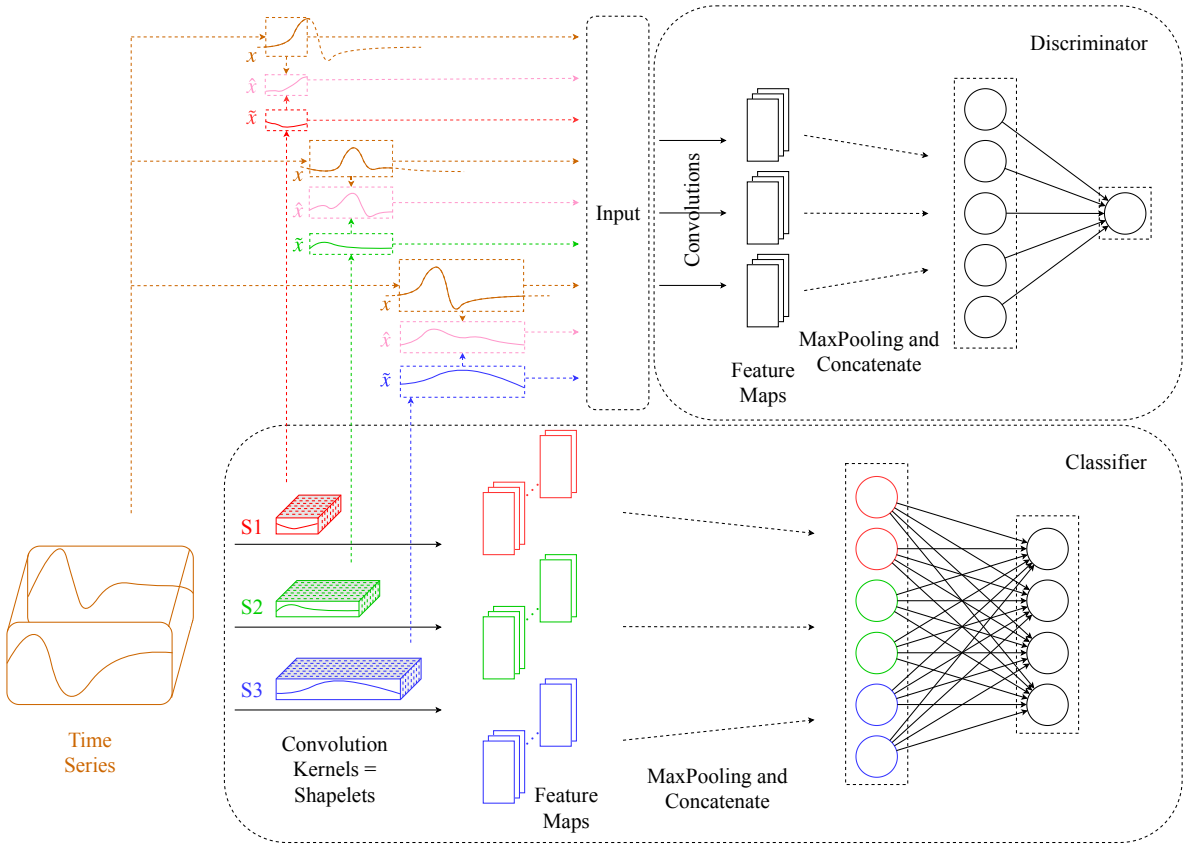


Figure 4.1 – Architecture of our proposed eXplainable-by-design CNN (XCNN)

distance between a portion of the time series Z starting at index i and a shapelet S of length L:

$$F(z_{i:i+L}, S) = \sum_{l=1}^{L} \left(z^{(i+l-1)} - S^{(l)} \right)^2. \tag{4.1}$$

The smaller this distance, the closer the shapelet is to the considered subseries. In a CNN, the feature map is obtained from a convolution, and hence encodes cross-correlation between a series and a shapelet:

$$F(z_{i:i+L}, S) = \sum_{l=1}^{L} z^{(i+l-1)} \cdot S^{(l)}. \tag{4.2}$$

Note that here, the higher $F(z_{i:i+L}, S)$, the more similar the shapelet is to the subseries. We will loosely refer to the convolution filters of a CNN classifier as *Shapelets* in the following.

Inspired by previous works on adversarial training, *e.g.* Generative Adversarial Networks (GANs) (Goodfellow et al. 2014), we make use of both a CNN classifier and an adversarial neural network (as shown in Figure 4.1) to regularize the convolution parameters of our classifier. We call this neural network architecture eXplainable-by-design CNN (XCNN).

A GAN is a combination of two neural networks: a generator and a discriminator which compete against each other during the training process to reach an equilibrium where the discriminator cannot distinguish between the generator outputs and real training data. In a GAN, the adversarial network is used to push the generator towards producing data as similar to real data as possible.

Contrarily to GANs, our adversarial architecture does not rely on a generator to produce fake samples from a latent space. Our regularization acts as a soft constraint for the classifier to learn shapelets (i.e. the convolution filters of the classifier) as similar to real pieces of the training time series as possible. As described in our framework (cf. Section 3.1), our model is self-explainable in modular level (due to the constrained shapelets), because the XCNN strategy iteratively modifies the shapelets such that they become close to subseries from the training set. To execute this strategy, the discriminator is trained to distinguish between real subseries from the training set and the shapelets. During the regularization phase, the discriminator updates the shapelets so that they become more and more similar to real subseries.

To obtain the best trade-off between the discriminative power of the shapelets (i.e. the final classification performance) and their interpretability, our training procedure alternates between training the discriminator and the classifier.

The type of data given as input to the discriminator is another major difference between a GAN and XCNN: in a GAN, the discriminator is fed with complete instances, while in XCNN, the discriminator takes subseries as input. These subseries can either be shapelets from the classifier model (denoted as $\tilde{x}$ in Figure 4.1), portions of training time series (denoted as x) or interpolations between shapelets and training time series portions ($\hat{x}$, see the following section for more details on those), as illustrated in Figure 4.2. This process allows the discriminator to alter the shapelets for better interpretability.

4.1.1 Loss Function

As for GANs, our optimization process alternates between losses attached to the subparts of our XCNN model. Here, each training epoch consists of three main steps that

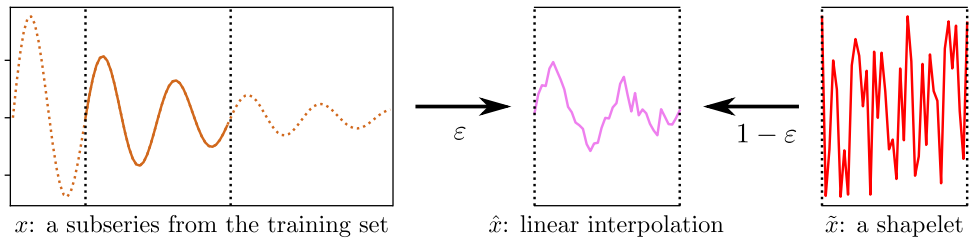


Figure 4.2 – An example of samples x, $\hat{x}$ and $\tilde{x}$ provided as input to the discriminator.

are (i) optimizing the classifier parameters for correct classification, (ii) optimizing the discriminator parameters to better distinguish between real subseries and shapelets and (iii) optimizing shapelets to fool the discriminator. Each of these steps is attached to a loss function that we describe in the following.

Firstly, a multi-class cross entropy loss is used for the classifier. It is denoted by $L_c(\theta_c)$ where θ_c is the list of all classifier parameters.

$$L_c(\theta_c) = -\sum_{c=1}^{n_{\text{cl}}} y_{o,c} \log(p_{o,c}) \tag{4.3}$$

where n_{cl} is the number of classes, y is the binary indicator if class label c is the correct classification for observation o, and p is predicted probability observation o is of class c.

Secondly, our discriminator is trained using a loss function derived from the Wasserstein GANs with Gradient Penalty (WGAN-GP) (Gulrajani et al. 2017):

$$L_d(\theta_d) = \mathop{\mathbb{E}}_{\tilde{x}\sim\mathbb{P}_S}[D(\tilde{x})] - \mathop{\mathbb{E}}_{x\sim\mathbb{P}_x}[D(x)] + \lambda \mathop{\mathbb{E}}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}\left[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2\right]$$

where $\mathbb{P}_S$ is the empirical distribution over the shapelets, $\mathbb{P}_x$ is the empirical distribution over the training subseries, and $\hat{x} = \epsilon x + (1-\epsilon)\tilde{x}$, where ϵ is drawn uniformly at random from the interval $[0, 1]$ (cf. Figure 4.2).

Thirdly, shapelets are updated to fool the discriminator by optimizing on the loss $L_r(\theta_s)$ where $\theta_s \subset \theta_c$ is the set of shapelet coefficients:

$$L_r(\theta_s) = -\mathop{\mathbb{E}}_{\tilde{x}\sim\mathbb{P}_S}[D(\tilde{x})] \tag{4.4}$$

Algorithm 1 presents the whole training procedure to update the parameters of our XCNN model. At each epoch of this algorithm, the three steps presented above are executed sequentially. Note that in the second step (lines 10–17), sampling classifier

Algorithm 1: Learning Interpretable Shapelet

Require: number of shapelets n_S
Require: random initialization for the classifier/discriminator/shapelets $\theta_c, \theta_d, \theta_s \subset \theta_c$
Require: gradient penalty coefficient λ
Require: number of epochs n_{epochs}, mini-batch size m
Require: number of classifier/discriminator/regularization mini-batches per epoch n_c, n_d, n_r
Require: optimizer (Adam) hyperparameters α, β_1, β_2

for $i = 1, \ldots, n_{epochs}$ **do**
  for $t = 1, \ldots, n_c$ **do**
    for $j = 1, \ldots, m$ **do**
      Sample a pair (Z_j, y_j) from the training set
      $\hat{y}_j \leftarrow h_{\theta_c}(Z_j)$
      $L_c^{(j)} \leftarrow \text{CrossEntropy}(y_j, \hat{y}_j)$
    end
    $\theta_c \leftarrow \text{Adam}(\nabla_{\theta_c} \frac{1}{m} \sum_{j=1}^{m} L_c^{(j)}, \theta_c, \alpha, \beta_1, \beta_2)$
  end
  for $t = 1, ..., n_d$ **do**
    for $j = 1, ..., m$ **do**
      Sample a shapelet $\tilde{x}_j$ from the set θ_s, a subseries x_j from the training set and a random number $\epsilon \sim U[0, 1]$
      $\hat{x}_j \leftarrow \epsilon x_j + (1 - \epsilon)\tilde{x}_j$
      $L_d^{(j)} \leftarrow D(\tilde{x}_j) - D(x_j) + \lambda(||\nabla_{\hat{x}} D(\hat{x}_j)||_2 - 1)^2$
    end
    $\theta_d \leftarrow \text{Adam}(\nabla_{\theta_d} \frac{1}{m} \sum_{j=1}^{m} L_d^{(j)}, \theta_d, \alpha, \beta_1, \beta_2)$
  end
  for $t = 1, \ldots, n_r$ **do**
    for $j = 1, \ldots, n_S$ **do**
      $\tilde{x}_j \leftarrow \theta_s[j]$
      $L_r^{(j)} \leftarrow -D(\tilde{x}_j)$
    end
    $\theta_s \leftarrow \text{Adam}(\nabla_{\theta_s} \frac{1}{n_S} \sum_{j=1}^{n_S} L_r^{(j)}, \theta_s, \alpha, \beta_1, \beta_2)$
  end
end

shapelets, as well as sampling subseries from the training set, is performed uniformly at random.

4.2 Experiments

In this section, we will detail the training procedure for XCNN and present both quantitative and qualitative experimental results. The source code for our experiments was made publicly available [2].

4.2.1 Experimental Setting

Competitors

We provide experiments about the quality (for explanations) of our learned shapelets as well as their quality for classification. We consider that the similar a shapelet looks like a part of a subseries, the more interpretable it is. This criterion is similar to the idea of prototype-based interpretability proposed by the paper "This looks like that: deep learning for interpretable image recognition" (Chen et al. 2019). As explained in Section 1.3.2, our most relevant competitor is Learning Shapelets (LS) from (Grabocka et al. 2014) as it also describes a shapelet-based model where the shapelets are learned and where a single model is used for classification. The quality (for explanations) of the shapelets produced by (Ye et al. 2009) and (Rakthanmanon et al. 2013) is, by design, perfect since the shapelets are true subpart of the original series so we do not compare with them but only with the shapelets learned by (Grabocka et al. 2014). However, we compare our classification performance to (Ye et al. 2009), Fast Shapelets (Rakthanmanon et al. 2013) and the recent ELIS (Fang et al. 2018).

Datasets

We use the 85 univariate time series datasets from the UCR repository for which most of our baselines results are already available (Chen et al. 2015). Note that our CNN-based method may also be suited for multivariate time series but giving "intuitive" explanations for multivariate data is far from obvious and we decided to focus only on univariate ones in this paper. The datasets are significantly different from one to another, including seven types of data with various number of instances, lengths, and classes. The splits between training and test sets are provided in the repository.

2. `https://github.com/yichangwang/XCNN`

Architecture details and parameter setting

We have implemented the XCNN model using TensorFlow (Abadi et al. 2015) following the general architecture illustrated in Figure 4.1. The classifier is composed of one 1D convolution layer with ReLU activation, followed by a max-pooling layer along the temporal dimension and a fully connected layer with a soft-max activation. The shapelets use a Glorot uniform initializer (Glorot et al. 2010) while the other weights are initialized uniformly (using a fixed range). For each dataset, three different shapelet lengths are considered, inspired by the heuristic from (Grabocka et al. 2014) but without resorting to hyper-parameter search: we consider 3 groups of $20 \times n_{\text{cl}}$ shapelets of length $0.2T$, $0.4T$ and $0.6T$, where n_{cl} is the number of classes in the dataset and T is the length of the time series at stake.

The convolution filters of the classifier, i.e. the shapelets, are given as input to the discriminator which has the same structure as the classifier, but with shorter convolution filters (100 filters of size $0.06T$, $0.12T$ and $0.18T$) and a single-neuron *tanh* activation instead of the soft-max in the last layer. For optimization, we use Adam optimizer with a standard parameterization ($\alpha = 10^{-3}$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$) and each epoch consists in $n_c = 15$ (resp. $n_d = 20$ and $n_r = 17$) mini-batches of optimization for the classifier loss (resp. discriminator and regularizer losses).

Experimental results are reported in terms of test accuracy and aggregated over five random initializations. All experiments are run for 8,000 training epochs.

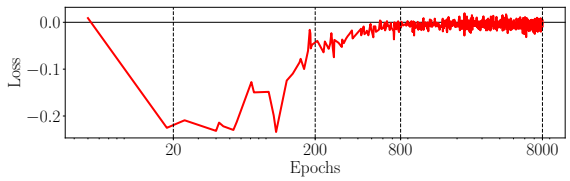
4.2.2 Qualitative results for explainability

We first describe how we compute the shapelet contributions to the classification of one (or multiple) example(s) and validate that our adversarial regularization actually ensures that shapelets are visually similar to the training data. We believe that the Euclidean distance is the most understandable distance for human eyes so all the figures that show shapelets and series will be displayed using this distance even though it is not the one optimized during XCNN training.

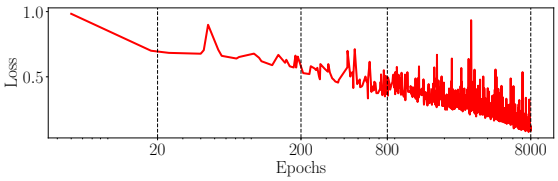
Then we show, in different ways how shapelets that look like subseries are better suited to explain decisions by comparing with standard *post-hoc* explanations from gradient-based and perturbation-based methods (cf. Section 3.2).

Training process

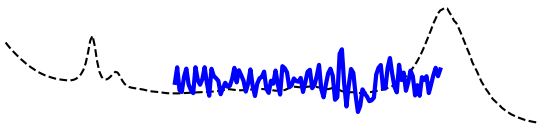
We illustrate our training process and its impact on a single shapelet in Figure 4.3. In this figure, we show the evolution of a given shapelet for the Wine dataset at epochs 20, 200, 800 and 8,000. One can see from the loss values reported in Figures 4.3a and 4.3b that these correspond to different stages in our learning process. At epoch 20, the Wasserstein loss is far from the 0 value ($L_d = 0$ corresponds to a case where the discriminator cannot distinguish between shapelets and real subseries), and this indeed corresponds to a shapelet that looks very different from an actual subseries. As epochs go, both the Wasserstein loss L_d and the cross-entropy one L_c get closer to 0, leading to both realistic and discriminative shapelets.



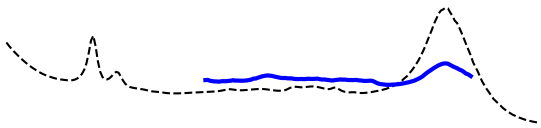
(a) Wasserstein loss L_d



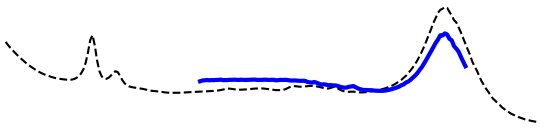
(b) Cross-entropy loss L_c



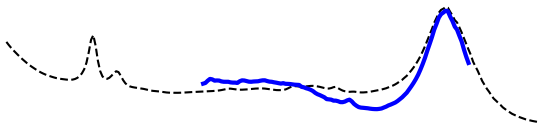
(c) Shapelet at epoch 20



(d) Shapelet at epoch 200



(e) Shapelet at epoch 800



(f) Shapelet at epoch 8000

Figure 4.3 – Illustration of the evolution of a shapelet during training (for the Wine dataset).

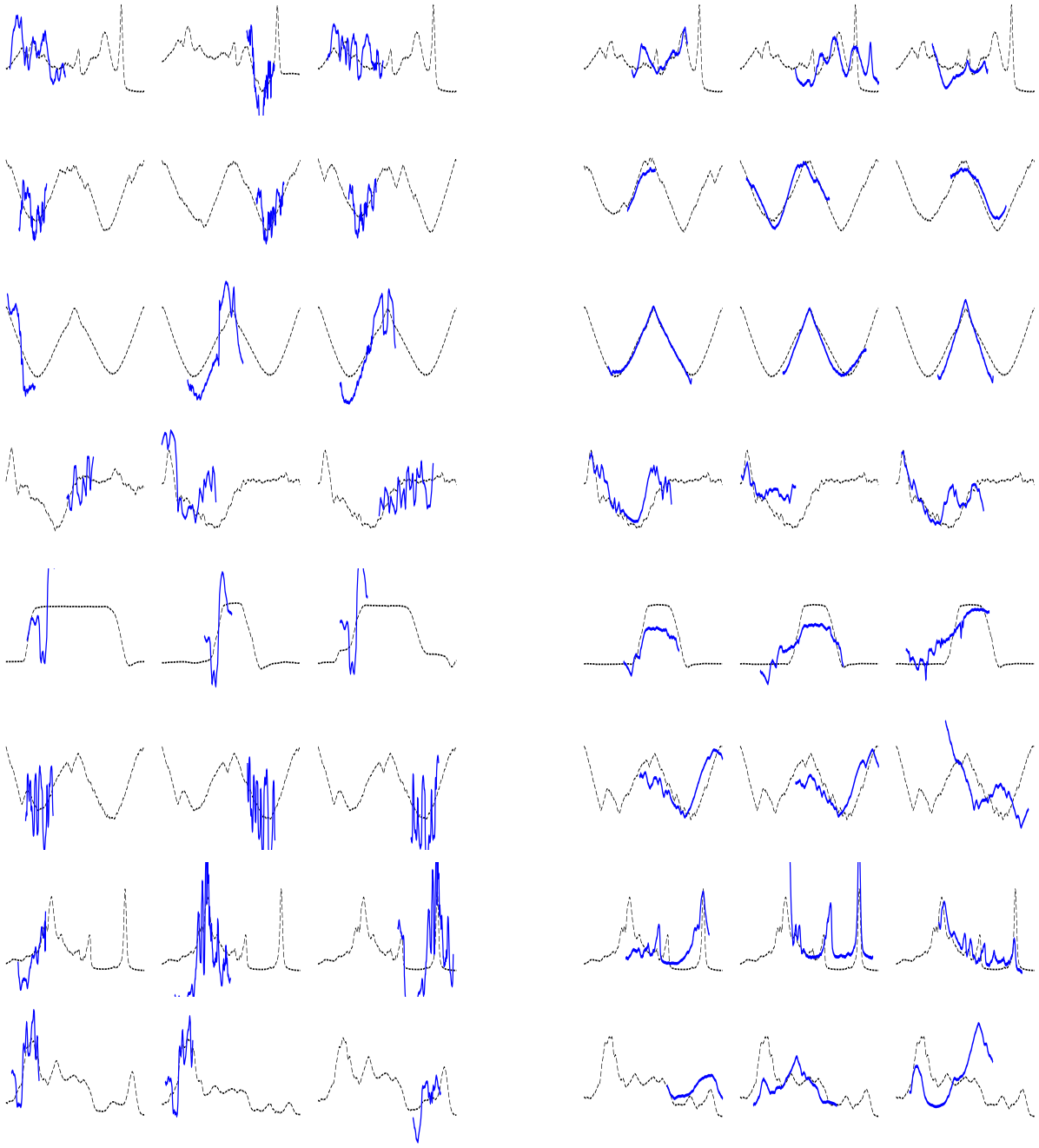


Figure 4.4 – Three most discriminative shapelets obtained for the datasets Beef, Car, DiatomSizeReduction, ECG200, GunPoint, Herring, OliveOil, Strawberry (rows 1 to 8, respectively) using (left column) Learning Shapelets or (right column) our XCNN architecture. The average discriminative power of the shapelets is evaluated using Eq. 4.5 and each shapelet is superimposed over its best matching time series in the test set.

Shapelet contributions (modular level interpretability)

The computation of the contribution of a shapelet to a decision is based on Grad-CAM. The "interesting" parts are shown using a heat map on the original image. We recall that in a convolutional neural network, a *feature map* is the output of a particular layer of neurons. It somehow (ignoring the activation function) shows the response of a given convolution filter to the output of the previous layer. Grad-CAM computes the feature importance α_k^c of the feature map A^k on the classification decision c. This is computed after the final pooling layer which transforms all spatial positions (for images) A_{ij}^k of the k^{th} feature map to a single value F^k. The filter importance weight α_k^c, for a given input image (omitted for conciseness), is calculated with:

$$\alpha_k^c = \frac{\partial y^c}{\partial F^k}$$

where y^c is the output of the network for class c.

Compared to the image classifiers used in (Selvaraju et al. 2017), in our time series classification problem (1-dimensional) we are interested in both the *positive* and *negative* contributions of each learned shapelet on the classification of the (set of) series (whereas in (Selvaraju et al. 2017) only the positive contributions matter). Those contributions are defined for a trained network and a given time series Z_i (implicitly present in the partial derivatives) as:

$$p_k(Z_i) = ReLU\left(\frac{\partial y^c}{\partial F^k}\right)$$

and

$$n_k(Z_i) = -ReLU\left(-\frac{\partial y^c}{\partial F^k}\right).$$

The positive and negative contribution of a shapelet is used for identifying whether the appearance of a shapelet is supportive or not for a prediction. Future explanations of this equation is discussed in the next subsection, as shown in Figure 4.6.

As F^k is obtained from a global max pooling ($F^k = \max_t A_t^k$), each shapelet contribution can be associated to a timestamp

$$t = \arg\max_{t'} A_{t'}^k,$$

allowing us to localize the contribution. To produce a heat map with the positive contri-

butions, we follow the same principle as in (Selvaraju et al. 2017):

$$L_{mask}(Z_i) = \sum_k p_k(Z_i)\tilde{A^k}(Z_i).$$

where $\tilde{A^k}$ is a vector of all zeros but at position $t = \arg\max_{t'} A^k_{t'}$ (where A^k_t is stored).

To obtain the **global positive contribution** of a shapelet k given a set of N time series examples, we compute

$$gp_k = \frac{1}{N}\sum_{i=1}^{N} p_k(Z_i). \tag{4.5}$$

The shapelets shown in Figure 4.4 are the 3 most contributing shapelets, according to this global criterion. In Figure 4.4, the shapelets learned by XCNN seem visually closer to the time series than the shapelets learned by LS.

We then computed the average L_2 between a shapelet and a subpart of a time series over all the shapelets learned by XCNN and by LS for a given dataset, computed at the best matching point of the closest time series in the dataset (also in terms of L_2). This L_2 distance is meaningful in our framework, because the closer a shapelet is to a time series, the more it contributes to the explainability of the model. The results are given in Figure 4.5. This scatter-plot shows that, even if the optimized distance between the shapelets and the input series in the neural network is not the L_2 one (it is the dot product), our adversarial regularization allows XCNN to obtain closer (in terms of L2) shapelets than LS which are deemed more suited for explanations.

Local and global interpretations with XCNN shapelets

Since we use a neural network classifier, we could directly benefit from the standard gradient-based explanations, as also discussed in (Fawaz et al. 2019), to show what parts of a given time series example is important for the classifier to take its classification decision.

These, nowadays standard, gradient-based explanations are interesting but do not show the inner working of the classifier and, in particular, the reason why some parts of the input series were particularly useful for the classification. We believe that our ability, with XCNN, to show the shapelets that were learned and used to make the classification gives a different type of information than the gradient-based one. To illustrate this, we overlay in Figure 4.6a and 4.6b the three most positively (resp. negatively on the right) contributing shapelets on the time series at their best matching location (using L_2

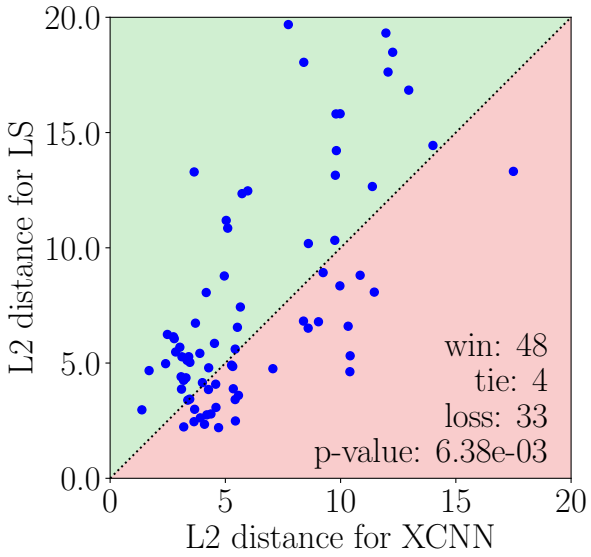


Figure 4.5 – Average over all the shapelets learned by XCNN and by LS for a given dataset, of the L_2 distances between a shapelet and a subpart of a time series at the best matching point of the closest time series in the dataset.

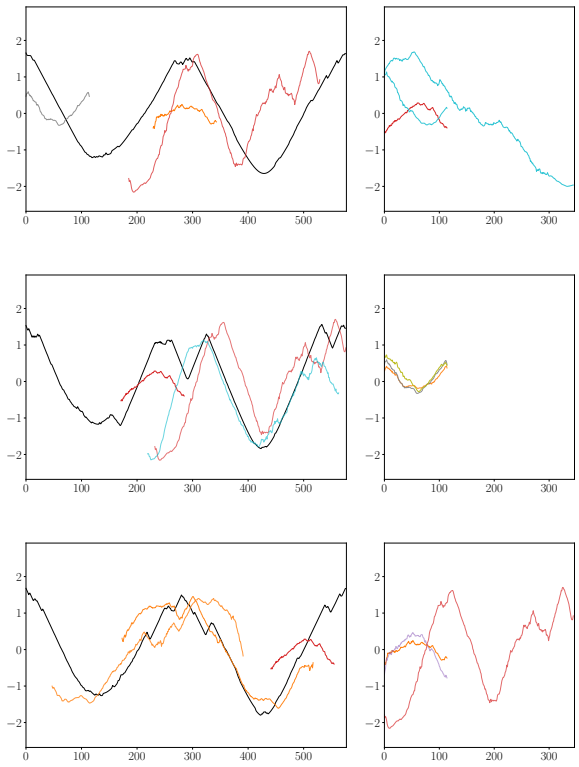
distance). Note that on the left side, the horizontal axis gives the length of the series (in black) while on the right, it gives the length of the shapelets which is at most 60% of the length of the series. We do not show the original series for the negative shapelets since, by definition, they are very far from the original series.

In Figure 4.6b there is no negative shapelet used to discriminate the series of this dataset. This is due to the fact that the series for all the classes are very similar except for very small changes in the slope of the bump or in the size of the plateau at the top of the bump. These small changes can be captured by the positive shapelets but many of them are used to succeed in discriminating the classes.

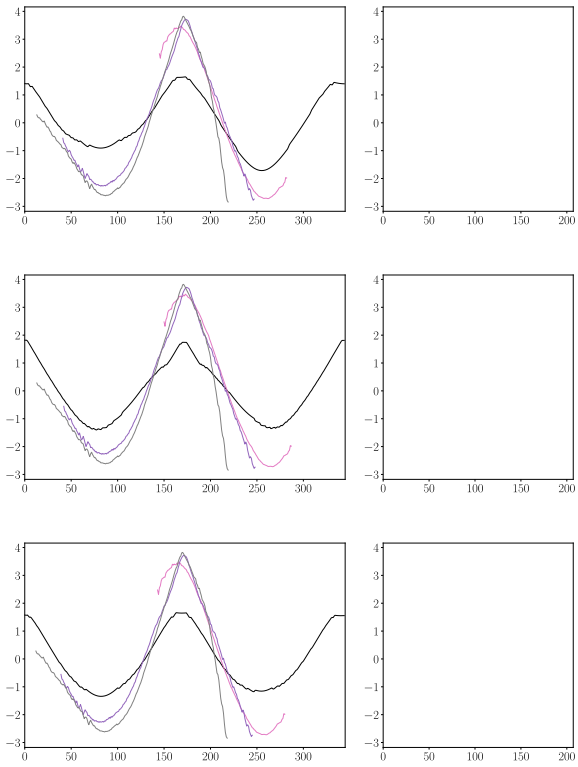
We can also use our method to show the shapelets that most contribute to the classification of all examples of a *given class*. This is useful when one wants to understand the class characteristics. The global relative positive contribution of one shapelet considering all series from a given class is:

$$rp_k(c) = ReLU\left(\frac{1}{N^c}\sum_{i=1}^{N^c}\left(p_k^c(Z_i) - \frac{1}{n_{\text{cl}}-1}\sum_{\substack{j=1\\ j\neq c}}^{n_{\text{cl}}-1} p_k^j(Z_i)\right)\right)$$

where N^c is the number of examples in class c, and n_{cl} is the total number of classes



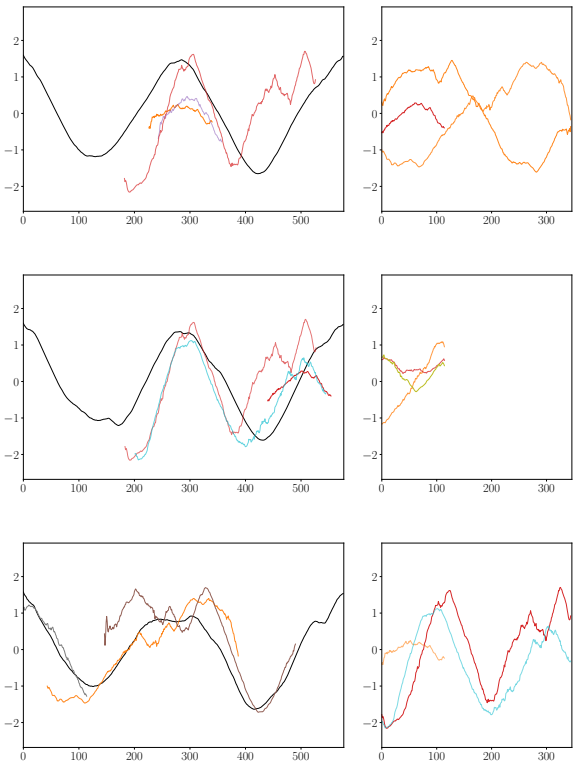
(a) Interpretations for random series of the classes (class 0, 1, 3, resp.).
Note that there are in total {92, 94, 114} positive shapelets used for this decision and {148, 146, 126} negative ones for these three predictions, resp.



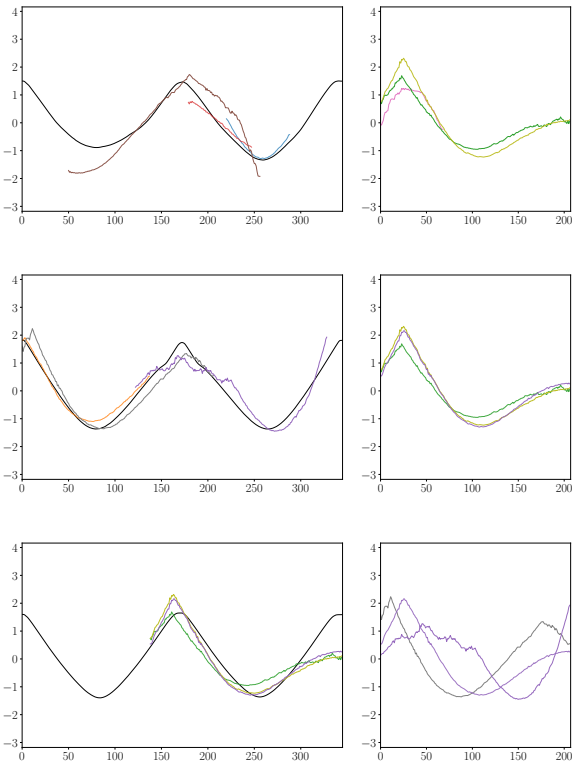
(b) Interpretations for random series of the classes (class 0, 1, 2, resp.).
Note that there are in total {239, 239, 239} positive shapelets used for this decision and {0, 0, 0} negative ones for these three predictions, resp.

Figure 4.6 – Local explanations with shapelets for random series for (a) the Car dataset and (b) the DiatomSizeReduction dataset.

in the dataset. Respectively, we can get the global relative negative contribution $rn_k(c)$ by replacing the positive contribution p_k^j with negative contribution n_k^j. The time series shown in black in Figures 4.7a and 4.7b is the average over all examples of a given class. With these figures, we can draw similar conclusions as the previous ones but for an entire class.



(a) Three most positively (left) and negatively (right) globally contributing shapelets for some of the classes (class 0, 1, 3, resp.) of the Car test set.



(b) Three most positively (left) and negatively (right) globally contributing shapelets for some of the classes (class 0, 1, 2, resp.) of the DiatomSizeReduction test set.

Figure 4.7 – Global interpretations with shapelets for some classes for (a) the Car dataset (b) the DiatomSizeReduction dataset.

Comparison with perturbation-based explanations (LEFTIST)

We compare the explanations provided by XCNN with the ones from LEFTIST (Guillemé et al. 2019b). As discussed in Section 3.2, LEFTIST prefixes the number of the interpretable components (the segments of the time series).

We firstly apply LEFTIST on a same time series with different segments on the Car dataset. As shown in Figure 4.8, each color block indicates the contribution of a segment: green and red blocks indicate positive and negative contributions, respectively, and the opacity of the color indicates the absolute value of the contribution. When segments increase from 3 to 20, the contribution of the leftmost feature from the segments changes

from positive to negative, and when there are a lot of segments (100 for instance), each segment contributes poorly to a prediction. This means that the segment number is difficult to choose, and explanations are highly dependent on different segment numbers. The lengths of shapelets in our XCNN are also prefixed, but as shown in Figure 4.6a, shapelets of different length could be used to explain a prediction result.

Figure 4.8 illustrates that for LEFTIST, once the segments are fixed, the position of the shapelets are fixed. The shapelets in our XCNN are sliding windows, which are deemed to be more flexible for giving explanations.

From Figure 4.9a we can see that, surprisingly, although the time series are from the same class, the contributions of the first and the last segments are not the same. We then increase the segments, as shown in Figure 4.9b, and find that the discriminative parts of different series from the same class become closer to what is found by XCNN.

These examples show that LEFTIST inherits the drawbacks of LIME: (i) the number of segments are difficult to set and this may cause faithfulness problems, (ii) once the number of segments is fixed, each interpretable component has a fixed position and the same length. Compared with the explanations given by our XCNN, those of LEFTIST are less flexible.

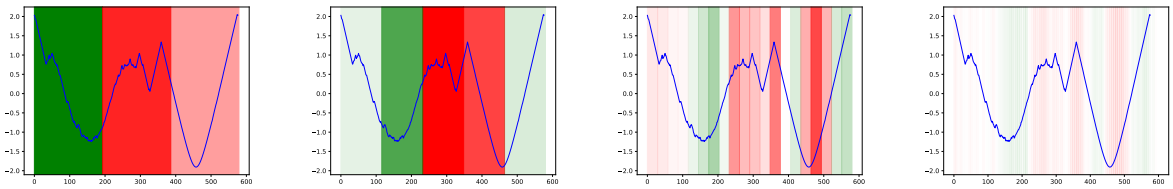
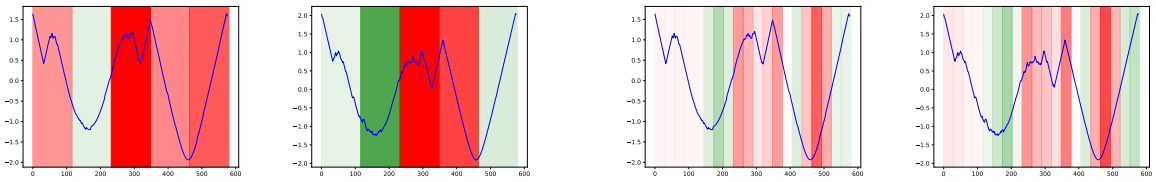


Figure 4.8 – LEFTIST explanations extracted from a same series (from class 2) with different segments (from left to right 3, 5, 20, 100, resp.) on the Car dataset.



(a) Explanations with 5 segments

(b) Explanations with 20 segments

Figure 4.9 – LEFTIST explanations extracted from different series from the same class (class 2) with same segments on the Car dataset.

4.2.3 Quantitative Results

XCNN is able to learn, by design, shapelets that are discriminative and suited for explanations. We want to quantify if this is achieved at the expense of classification accuracy and/or computation time. Our goal is to be much faster than exhaustive shapelet search methods (our baseline is Shapelets (Ye et al. 2009)), much more accurate than very fast random shapelet selection-based methods (our baseline is FS (Rakthanmanon et al. 2013)) and as accurate and as fast as single model shapelet learning methods (our baselines are LS (Grabocka et al. 2014) and ELIS (Fang et al. 2018)) with additional interpretability benefits.

Accuracy

We analyze the accuracies obtained by FS, LS, ELIS and our XCNN method on the 85 datasets using scatter plots. We compare FS versus XCNN in Figure 4.10, LS versus XCNN in Figure 4.11 and ELIS versus XCNN in Figure 4.12. We also show how a simple CNN (without the adversarial regularization) compares against LS in Figure 4.13. We indicate the number of *win/tie/loss* for our method and we provide a Wilcoxon significance test (Demšar 2006) with the resulting p-value (> 0.01: none of the two methods is significantly better than the other). The points on the diagonal are datasets for which the accuracy is identical for both competitors.

Figure 4.10 shows that, as expected, our method yields significantly better performance than FS. It gives similar results (not significantly better nor worse on average) than ELIS for 52 datasets for which ELIS terminated in 48 hours. However for 33 datasets ELIS took more than 48 hours to complete.

Compared to LS, as shown in Figure 4.11, for most datasets, the difference in accuracy is low, with a small edge (significant) for LS: on average for the 85 datasets, LS obtains an accuracy of 0.77 whereas XCNN obtains an accuracy 0.76. On three datasets (namely HandOutlines, NonInvasiveFetalECGThorax1 and OliveOil), our XCNN method and its regularization seems to be strongly positive (and detrimental on one dataset), in terms of generalization. A simple CNN that would correspond to the classifier of our XCNN alone seems to give slightly better (non significant) results than LS (and thus than our XCNN). This means that our backbone neural network architecture is a good candidate to jointly learn interpretable shapelets and classify time series with little loss on accuracy.

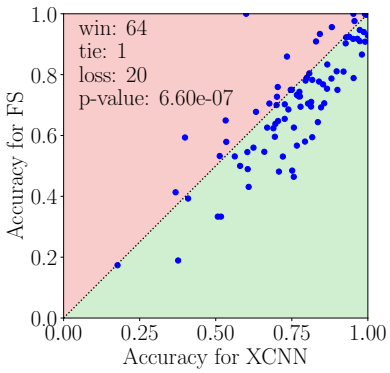


Figure 4.10 – Accuracy comparison between Fast Shapelets (FS) and XCNN on 85 datasets (each point is a dataset) of the UCR repository (Chen et al. 2015).

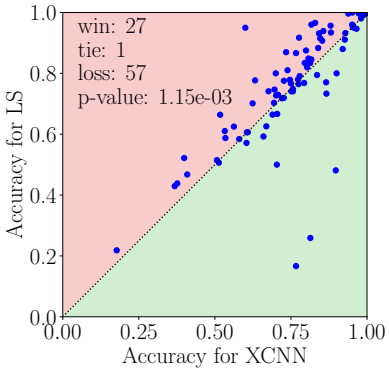


Figure 4.11 – Accuracy comparison between Learning Shapelets (LS) and XCNN on 85 datasets (each point is a dataset) of the UCR repository (Chen et al. 2015).

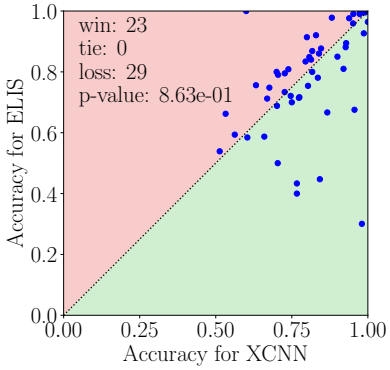


Figure 4.12 – Accuracy comparison between ELIS and XCNN on 52 datasets of the UCR repository (Chen et al. 2015).

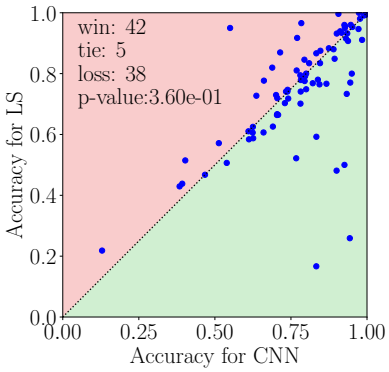


Figure 4.13 – Accuracy comparison between Learning Shapelets (LS) and a simple CNN on 85 datasets of the UCR repository (Chen et al. 2015).

Training Time

Table 4.1 – Complexity of four different shapelet-based TSC algorithms (Shapelet (Ye et al. 2009), FS (Rakthanmanon et al. 2013), LS (Grabocka et al. 2014) and XCNN). n is the number of examples in the training set, T is the average length of the time series, n_{shap} is the number of selected shapelets (if set *a priori*), and n_{cl} is the number of classes.

Shapelet	FS	LS and XCNN (per epoch)
$O(n^2 \cdot T^4)$	$O(n \cdot T^2)$	$O(n \cdot (T^2 n_{\text{shap}} + n_{\text{shap}} \cdot n_{\text{cl}}))$

We provide a theoretical complexity study (see Table 4.1) of all the baselines and of

our XCNN method. Our method is based on a classifier and a discriminator, and both of them are simple CNNs. So the complexity of our algorithm ($O(n \cdot (T^2 n_{\text{shap}} + n_{\text{shap}} \cdot n_{\text{cl}}))$) is related to training a CNN and should depend mainly on the number of examples (n), the average length of the time series (T), and the number of classes (n_{cl}), since the latter is used to decide the number of shapelets to be learned. Note that for both LS and XCNN, the provided complexity is the one for a single iteration of the algorithm since the number of iterations required for such algorithms to converge is highly data dependent.

To have a better grasp on the actual training time of all methods, we ran the methods on a single dataset (ElectricDevices) and recorded the CPU time. The experiments were conducted on a Debian Cluster using Intel(R) Xeon(R) CPU E5-2650 v4 Processor (12 core 2.20 GHz CPU) with 32GB memory. The results are averaged over five runs. The implementation code of our baselines is taken from Bagnall et al. (2017) (as for the accuracy results). As expected, the original Shapelet (Ye et al. 2009) method does not finish in 48 hours for this medium size dataset. FS finishes in 12.1 minutes, LS finishes in 2323 minutes, and our method takes 142 minutes. The theoretical complexity of LS and XCNN is identical so these results were surprising. We suspected that the JAVA implementation of LS was not well optimized and we used the implementation of LS method from tslearn (Tavenard et al. 2020) using Keras [3] with TensorFlow as backend. With this implementation, the training phase took only 71 minutes for LS on this dataset (compared to 142 for XCNN) which shows that the time difference between the two algorithms is mainly related to the implementation (and the hyper-parameters related to the number of epochs).

4.2.4 Discussion

We place our XCNN method in our performance-interpretable framework, as shown in Figure 4.14. Note that our model is interpretable at all levels.

Our main contribution uses a shapelet-based representation for TSC tasks. Our shapelets are constrained to be the discriminative parts of the time series, as the prototypes, and provide *in-situ* modular level interpretability as well as global and local level explanations. We consider shapelets, the subsequences of time series which are both important for classification and for explanations: for us, the visual quality of shapelets, i.e., the similarity to real time series, provides sequences for explaining a particular prediction

3. `https://keras.io/`

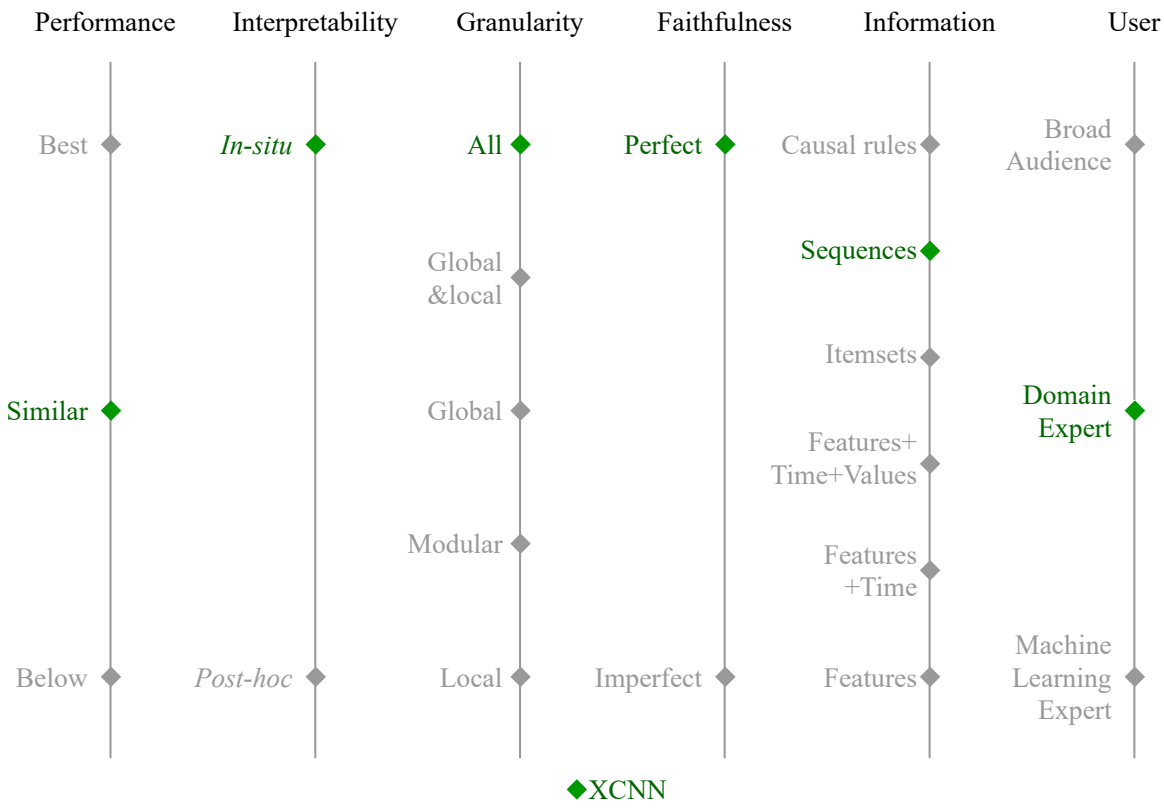


Figure 4.14 – Placement of our XCNN method in the performance-interpretability framework.

to domain experts.

As discussed in Chapter 1, the representations of time series can be in many domains (*e.g.* time domain, frequency domain, shapelet domain, etc.). Our shapelet-based method only gives explanations in the shapelet domain, i.e., the presence/absence of a shapelet leads to such prediction. However, sometimes a discriminative feature of a prediction could be found in other domain, *e.g.* the presence of a frequency in the time series leads to a particular class. In this context, our shapelet-based method is limited in the shapelet domain.

4.3 Summary

In this chapter, we have presented a new shapelet-based time series classification method that produces shapelets that are, by design, better suited to explain decisions. The method is based on a novel adversarial architecture where one convolutional neural

network is used to classify the series and another one is used to constrain the first network to learn both discriminative and meaningful shapelets. Our results show that the expected trade-off between accuracy and interpretability is satisfactory: our classification results are comparable with similar state-of-the-art methods but with shapelets that can be used in many different ways to explain the decisions. We also illustrate how the learned shapelet can be used to better understand decisions made by a classifier in terms of (i) localization of the discriminative information in the time series and (ii) visualization of the most prominent shapelets at stake in classification decisions.

We believe that the proposed adversarial regularization method could be used in many more applications where one would like to shape the features used to classify in order to better explain the decisions. This is different from what is, for example, used in Generative Adversarial Networks, where the regularization is made on the latent representation which is much less interesting in the case of time series when trying to obtain explainable decisions.

CONCLUSION AND PERSPECTIVES

Conclusion

As time series classification (TSC) and interpretability are both key issues in machine learning, in this thesis we address the problem of interpretable time series classification.

There exists a variety of terms about interpretability (*e.g.* explainability, comprehensibility), and we standardize the terminology of model interpretability, i.e., as long as a human can understand a model, either through its sparsity, through its description in a familiar language, or through a produced explanation, the model is interpretable. We emphasize the difference between *in-situ* interpretable model and *post-hoc* interpretability for a black-box model. We discuss the necessity of interpretability in AI system: interpretability is important not only for trust or ethic/regularization reasons, but also because machine learning experts could apply interpretability to understand better the data or to debug the model. Based on the life cycle of a machine learning model, we distinguish *ad-hoc*, *in-situ*, and *post-hoc* interpretability. We believe, as also pointed out by Chen et al. (2019) and Rudin (2019), that *post-hoc* explanations may cause faithfulness problems, and that *in-situ* ones might be preferable. Similar to Arrieta et al. (2020), we group different machine learning models in terms of their interpretability at a global level (*e.g.* statistical regression models, decision trees, and rule-based learning), at a modular level (*e.g.* naive Bayes and kNN), and non-interpretable models (*e.g.* SVM, ensemble methods, and deep neural networks). For those non-interpretable models, we review *post-hoc* explanation techniques to understand the prediction of a model locally, i.e., related to a specific prediction.

We adapted a framework of Fauvel et al. (2020b) (called performance-explainability framework) to put our method into perspective. Our framework is called performance-interpretability framework, and tries to answer the following questions: *How does the model perform in the state-of-the-art? How is the model interpretable? What is the scope of its interpretability (granularity)? Are the explanations faithful? Which kind of information do the explanations provide? What is the target user category of the explanations?* Different from the framework of Fauvel et al. (2020b), which focuses on the quality of

post-hoc explanations, our framework emphasize the *in-situ* model interpretability, and it combines granularity for both *in-situ* and *post-hoc* interpretability. We compare different *post-hoc* explanation techniques (namely CAM and LEFTIST) for TSC tasks in our proposed framework, and find that they both provide *post-hoc* local interpretability, which are considered unfaithful. The only difference between them is that CAM explanations provide both feature importance and the occurring time stamps, while LEFTIST provide feature importance at prefixed time stamps.

We propose a method that learns an *in-situ* interpretable model at the modular level. Our model (called eXplainable CNN (XCNN)) is based on Learning Shapelets (LS) (Grabocka et al. 2014) which for us is not interpretable, because the learned shapelets of LS are not similar to original subsequences of the time series. We regularize the shapelets with constraints, so that our shapelets would look like the subsequences. Our idea is similar to the idea of learning constrained prototypes that are similar to subparts of an image (Chen et al. 2019), and XCNN provide patterns (shapelets) that represent the characteristic of the input time series. In addition to the *in-situ* property, our model is able to provide faithful interpretations by positioning the shapelets on the time series, and with our reformulated Grad-CAM-liked *post-hoc* techniques, we provide sequences as information and our explanations are both at the local and global levels. These interpretations are helpful for domain experts to evaluate our model. In terms of accuracy, XCNN performance are on a par with comparable state-of-the-art methods, and XCNN provides new types of explanations for neural networks' predictions.

Perspectives

XCNN is a flexible and scalable method. It is interpretable at the modular level, because there are many constrained shapelets used as feature extractors in our model and each extracted feature contributes to the prediction. The sparsity of the extracted features can be useful to enhance our model interpretability to the global level. We consider using other regularization terms (*e.g.* group-lasso) to enforce this sparsity. This group-lasso (also called $l_{2,1}$ or l_1/l_2 norm) could constrain the network to use as few shapelets as possible (Bascol et al. 2016).

The second perspective is to adapt XCNN for Multivariate Time Series Classification (MTSC) tasks. Fauvel et al. (2020a) proposed an eXplainable Convolutional neural network for Multivariate time series classification (XCM). XCM extracts the temporal-

spatial information in parallel from an input time series, and makes decisions based on the extracted information. XCM performed well, but as Fauvel et al. (2020a) applied Grad-CAM for the explanations for XCM, the explanations of XCM are *post-hoc*. Different from Fauvel et al. (2019), our objective is to build *in-situ* interpretable networks. Bostrom et al. (2017) proposed shapelet-based method for MTSC tasks: Multivariate Dependent Shapelet Transform (MST_D) and Multivariate Independent Shapelet Transform (MST_I). The difference is that MST_D applies a shapelet as sliding windows for all the dimensions, while MST_I finds the closest match of a shapelet to each dimension independently (thus the best match time stamps can be different for MST_I). When we adapt our XCNN to MTSC, the shapelets can also be treated as dimension dependently or independently, and the interpretation of these shapelets on the multivariate time series should be done carefully.

As a third perspective, we would like to increase the depth of XCNN to potentially increase its performance. In a deeper network, it will not be possible to apply a regularization on the "shapelets" in our XCNN. We could regularize the "prototypes", as the prototypical part network (ProtoPNet) proposed by Chen et al. (2019). As discussed in Section 2.4.2, ProtoPNet uses several 1×1 convolutional operations as their prototypes in the prototype layer, and they constrain the prototype layer to be similar to a patch of the output of the last convolutional layer before the prototype layer. Each prototype is a latent representation of some training image patch, and the visualization of the prototypes are upsampled in the original image. Thus, we consider applying a prototype layer as ProtoPNet does, combined with our current regularization to make the XCNN deeper. This combination would also make it possible to use the network architecture on different data types than time series (images for example).

Bibliography

Abadi, Martín et al. (2015), *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*.

Adadi, Amina and Mohammed Berrada (2018), "Peeking inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)", *in*: *IEEE Access* 6, pp. 52138–52160.

Adebayo, Julius, Justin Gilmer, Michael Muelly, Ian J. Goodfellow, Moritz Hardt, and Been Kim (2018), "Sanity Checks for Saliency Maps", *in*: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pp. 9525–9536.

Agrawal, Rakesh, Christos Faloutsos, and Arun N. Swami (1993), "Efficient Similarity Search in Sequence Databases", *in*: *Foundations of Data Organization and Algorithms, 4th International Conference, FODO'93, Chicago, Illinois, USA, October 13-15, 1993, Proceedings*, vol. 730, pp. 69–84.

Alaa, Ahmed M. and Mihaela van der Schaar (2019), "Demystifying Black-Box Models with Symbolic Metamodels", *in*: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pp. 11301–11311.

Angwin, Julia, Jeff Larson, Surya Mattu, and Lauren Kirchner (2016), "Machine Bias", *in*: *ProPublica, May* 23.*2016*, pp. 139–159.

Arnold, Matthew et al. (2019), "FactSheets: Increasing Trust in AI Services through Supplier's Declarations of Conformity", *in*: *IBM Journal of Research and Development* 63.*4/5*, 6:1–6:13.

Arrieta, Alejandro Barredo et al. (2020), "Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI", *in*: *Inf. Fusion* 58, pp. 82–115.

Audebert, Nicolas, Alexandre Boulch, Bertrand Le Saux, and Sébastien Lefèvre (2019), "Distance Transform Regression for Spatially-Aware Deep Semantic Segmentation", *in*: *Computer Vision and Image Understanding* 189, p. 102809.

Bagnall, Anthony, Jason Lines, Jon Hills, and Aaron Bostrom (2015), "Time-Series Classification with COTE: The Collective of Transformation-Based Ensembles", *in*: *IEEE Transactions on Knowledge and Data Engineering* 27.*9*, pp. 2522–2535.

Bagnall, Anthony J., Michael Flynn, James Large, Jason Lines, and Matthew Middlehurst (2020), "On the Usage and Performance of the Hierarchical Vote Collective of Transformation-Based Ensembles Version 1.0 (HIVE-COTE v1.0)", *in*: *Advanced Analytics and Learning on Temporal Data - 5th ECML PKDD Workshop, AALTD 2020, Ghent, Belgium, September 18, 2020, Revised Selected Papers*, vol. 12588, pp. 3–18.

Bagnall, Anthony J., Jason Lines, Aaron Bostrom, James Large, and Eamonn J. Keogh (2017), "The Great Time Series Classification Bake off: A Review and Experimental Evaluation of Recent Algorithmic Advances", *in*: *Data Mining and Knowledge Discovery* 31.*3*, pp. 606–660.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015), "Neural Machine Translation by Jointly Learning to Align and Translate", *in*: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Barocas, Solon and Andrew D Selbst (2016), "Big Data's Disparate Impact", *in*: *California Law Review* 104, p. 671.

Bascol, Kevin, Rémi Emonet, Elisa Fromont, and Jean-Marc Odobez (2016), "Unsupervised Interpretable Pattern Discovery in Time Series Using Autoencoders", *in*: *Structural, Syntactic, and Statistical Pattern Recognition*, vol. 10029, pp. 427–438.

Beaudouin, Valérie, Isabelle Bloch, David Bounie, Stéphan Clémençon, Florence d'Alché-Buc, James Eagan, Winston Maxwell, Pavlo Mozharovskyi, and Jayneel Parekh (2020), "Identifying the "Right" Level of Explanation in a given Situation", *in*: *Proceedings of the First International Workshop on New Foundations for Human-Centered AI (NeHuAI) Co-Located with 24th European Conference on Artificial Intelligence (ECAI 2020), Santiago de Compostella, Spain, September 4, 2020*, vol. 2659, pp. 63–66.

Bellman, Richard (1957), *Dynamic Programming*, 339 pp.

Bengio, Yoshua, Patrice Y. Simard, and Paolo Frasconi (1994), "Learning Long-Term Dependencies with Gradient Descent Is Difficult", *in*: *IEEE Transactions on Neural Networks* 5.*2*, pp. 157–166.

Bibal, Adrien (2020), "Interpretability and Explainability in Machine Learning", PhD thesis, University of Namur.

Bibal, Adrien and Benoît Frénay (2016), "Interpretability of Machine Learning Models and Representations: An Introduction", *in*: *24th European Symposium on Artificial Neural Networks, ESANN 2016, Bruges, Belgium, April 27-29, 2016*.

Biehl, Michael, Barbara Hammer, and Thomas Villmann (2016), "Prototype-Based Models in Machine Learning", *in*: *WIREs Cognitive Science* 7.*2*, pp. 92–111.

Bien, Jacob and Robert Tibshirani (2011), "Prototype Selection for Interpretable Classification", *in*: *The Annals of Applied Statistics* 5.*4*, pp. 2403–2424.

Biran, Or and Courtenay Cotton (2017), "Explanation and Justification in Machine Learning: A Survey", *in*: *IJCAI-17 Workshop on Explainable AI (XAI)*, vol. 8, pp. 8–13.

Boser, Bernhard E., Isabelle M. Guyon, and Vladimir N. Vapnik (1992), "A Training Algorithm for Optimal Margin Classifiers", *in*: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pp. 144–152.

Bostrom, Aaron and Anthony J. Bagnall (2017), "A Shapelet Transform for Multivariate Time Series Classification", *in*: *CoRR* abs/1712.06428.

Burger, Benjamin et al. (July 2020), "A Mobile Robotic Chemist", *in*: *Nature* 583.*7815* (7815), pp. 237–241.

Camburu, Oana-Maria (Oct. 4, 2020), "Explaining Deep Neural Networks", PhD thesis, University of Oxford.

Chattopadhyay, Aditya, Anirban Sarkar, Prantik Howlader, and Vineeth N. Balasubramanian (2018), "Grad-CAM++: Generalized Gradient-Based Visual Explanations for Deep Convolutional Networks", *in*: *2018 IEEE Winter Conference on Applications of Computer Vision, WACV 2018, Lake Tahoe, NV, USA, March 12-15, 2018*, pp. 839–847.

Chen, Chaofan (2020), "Interpretability by Design: New Interpretable Machine Learning Models and Methods", *in*: p. 203.

Chen, Chaofan, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan Su (2019), "This Looks like That: Deep Learning for Interpretable Image Recognition", *in*: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pp. 8928–8939.

Chen, Yanping, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista (July 2015), *The UCR Time Series Classification Archive*.

Cho, Kyunghyun, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014), "Learning Phrase Representa-

tions Using RNN Encoder-Decoder for Statistical Machine Translation", *in*: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A Meeting of SIGDAT, a Special Interest Group of the ACL*, pp. 1724–1734.

Choi, Edward, Mohammad Taha Bahadori, Jimeng Sun, Joshua Kulas, Andy Schuetz, and Walter F. Stewart (2016), "RETAIN: An Interpretable Predictive Model for Healthcare Using Reverse Time Attention Mechanism", *in*: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 3504–3512.

Craven, Mark W. and Jude W. Shavlik (1995), "Extracting Tree-Structured Representations of Trained Networks", *in*: *Advances in Neural Information Processing Systems 8, NIPS, Denver, CO, USA, November 27-30, 1995*, pp. 24–30.

Dempster, Angus, François Petitjean, and Geoffrey I. Webb (2020), "ROCKET: Exceptionally Fast and Accurate Time Series Classification Using Random Convolutional Kernels", *in*: *Data Mining and Knowledge Discovery* 34.*5*, pp. 1454–1495.

Demšar, Janez (Dec. 2006), "Statistical Comparisons of Classifiers over Multiple Data Sets", *in*: *J. Mach. Learn. Res.* 7, pp. 1–30.

Dhurandhar, Amit, Pin-Yu Chen, Ronny Luss, Chun-Chen Tu, Pai-Shun Ting, Karthikeyan Shanmugam, and Payel Das (2018), "Explanations Based on the Missing: Towards Contrastive Explanations with Pertinent Negatives", *in*: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 590–601.

Dietterich, Thomas G. (2000), "Ensemble Methods in Machine Learning", *in*: *Multiple Classifier Systems*, pp. 1–15.

Domingos, Pedro M. (1998), "Knowledge Discovery via Multiple Models", *in*: *Intelligent Data Analysis* 2.*1-4*, pp. 187–202.

Doquet, Guillaume and Michèle Sebag (2019), "Agnostic Feature Selection", *in*: *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2019, Würzburg, Germany, September 16-20, 2019, Proceedings, Part I*, vol. 11906, pp. 343–358.

Doshi-Velez, Finale and Been Kim (Feb. 27, 2017), *Towards A Rigorous Science of Interpretable Machine Learning*, URL: `http://arxiv.org/abs/1702.08608` (visited on 04/04/2019).

Du, Mengnan, Ninghao Liu, and Xia Hu (Dec. 20, 2019), "Techniques for Interpretable Machine Learning", *in*: *Communications of the ACM* 63.*1*, pp. 68–77.

Faloutsos, Christos, M. Ranganathan, and Yannis Manolopoulos (1994), "Fast Subsequence Matching in Time-Series Databases", *in*: *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, USA, May 24-27, 1994*, pp. 419–429.

Fang, Zicheng, Peng Wang, and Wei Wang (Apr. 2018), "Efficient Learning Interpretable Shapelets for Accurate Time Series Classification", *in*: *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pp. 497–508.

Fauvel, Kevin (2020), "Enhancing Performance and Explainability of Multivariate Time Series Machine Learning Methods: Applications for Social Impact in Dairy Resource Monitoring and Earthquake Early Warning", PhD thesis, inria.

Fauvel, Kevin, Tao Lin, Véronique Masson, Élisa Fromont, and Alexandre Termier (2020a), "XCM: An Explainable Convolutional Neural Network for Multivariate Time Series Classification", *in*: *CoRR* abs/2009.04796.

Fauvel, Kevin, Véronique Masson, and Élisa Fromont (2020b), "A Performance-Explainability Framework to Benchmark Machine Learning Methods: Application to Multivariate Time Series Classifiers", *in*: *CoRR* abs/2005.14501.

Fauvel, Kevin, Véronique Masson, Élisa Fromont, Philippe Faverdin, and Alexandre Termier (2019), "Towards Sustainable Dairy Management - A Machine Learning Enhanced Method for Estrus Detection", *in*: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pp. 3051–3059.

Fawaz, Hassan Ismail, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller (July 2019), "Deep Learning for Time Series Classification: A Review", *in*: *Data Mining and Knowledge Discovery* 33.*4*, pp. 917–963.

Fawaz, Hassan Ismail, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean (2020), "InceptionTime: Finding AlexNet for Time Series Classification", *in*: *Data Mining and Knowledge Discovery* 34.*6*, pp. 1936–1962.

Fürnkranz, Johannes, Dragan Gamberger, and Nada Lavrac (2012), *Foundations of Rule Learning*.

García, Salvador, Joaquín Derrac, José Ramón Cano, and Francisco Herrera (2012), "Prototype Selection for Nearest Neighbor Classification: Taxonomy and Empirical Study",

in: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.*3*, pp. 417–435.

Garnot, Vivien Sainte Fare, Loïc Landrieu, Sébastien Giordano, and Nesrine Chehata (2020), "Satellite Image Time Series Classification with Pixel-Set Encoders and Temporal Self-Attention", *in*: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pp. 12322–12331.

Gauthier, Raphaël, Christine Largouët, Laurence Rozé, and Jean-Yves Dourmad (2021), "Online Forecasting of Daily Feed Intake in Lactating Sows Supported by Offline Time-Series Clustering, for Precision Livestock Farming", *in*: *Computers and Electronics in Agriculture* 188, p. 106329.

Gee, Alan H., Diego García-Olano, Joydeep Ghosh, and David Paydarfar (2019), "Explaining Deep Classification of Time-Series Data with Learned Prototypes", *in*: *Proceedings of the 4th International Workshop on Knowledge Discovery in Healthcare Data Co-Located with the 28th International Joint Conference on Artificial Intelligence, KDH@IJCAI 2019, Macao, China, August 10th, 2019*, vol. 2429, pp. 15–22.

Gilpin, Leilani H., David Bau, Ben Z. Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal (2018), "Explaining Explanations: An Overview of Interpretability of Machine Learning", *in*: *5th IEEE International Conference on Data Science and Advanced Analytics, DSAA 2018, Turin, Italy, October 1-3, 2018*, pp. 80–89.

Glorot, Xavier and Yoshua Bengio (May 13–15, 2010), "Understanding the Difficulty of Training Deep Feedforward Neural Networks", *in*: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, vol. 9, pp. 249–256.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016), *Deep Learning*.

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2014), "Generative Adversarial Nets", *in*: *Advances in Neural Information Processing Systems*, pp. 2672–2680.

Goodman, Bryce and Seth R. Flaxman (2017), "European Union Regulations on Algorithmic Decision-Making and a "Right to Explanation"", *in*: *AI Mag.* 38.*3*, pp. 50–57.

Grabocka, Josif, Nicolas Schilling, Martin Wistuba, and Lars Schmidt-Thieme (2014), "Learning Time-Series Shapelets", *in*: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 392–401.

Green, Ben (Jan. 27, 2020), "The False Promise of Risk Assessments: Epistemic Reform and the Limits of Fairness", *in*: *Proceedings of the 2020 Conference on Fairness, Ac-*

countability, and Transparency, FAT* '20: Conference on Fairness, Accountability, and Transparency, pp. 594–606.

Grote, Thomas and Philipp Berens (2020), "On the Ethics of Algorithmic Decision-Making in Healthcare", *in*: *Journal of Medical Ethics* 46.*3*, pp. 205–211.

Guidotti, Riccardo, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi (Aug. 2018), "A Survey of Methods for Explaining Black Box Models", *in*: *ACM Comput. Surv.* 51.*5*, 93:1–93:42.

Guillemé, Maël, Simon Malinowski, Romain Tavenard, and Xavier Renard (2019a), "Localized Random Shapelets", *in*: *Advanced Analytics and Learning on Temporal Data - 4th ECML PKDD Workshop, AALTD 2019, Würzburg, Germany, September 20, 2019, Revised Selected Papers*, vol. 11986, pp. 85–97.

Guillemé, Maël, Veronique Masson, Laurence Roze, and Alexandre Termier (Nov. 2019b), "Agnostic Local Explanation for Time Series Classification", *in*: *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), pp. 432–439.

Gulrajani, Ishaan, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville (2017), "Improved Training of Wasserstein GANs", *in*: *Advances in Neural Information Processing Systems (NIPS)*.

Hao, Yifan and Huiping Cao (July 2020), "A New Attention Mechanism to Classify Multivariate Time Series", *in*: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, Twenty-Ninth International Joint Conference on Artificial Intelligence and Seventeenth Pacific Rim International Conference on Artificial Intelligence {IJCAI-PRICAI-20}, pp. 1999–2005.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016), "Deep Residual Learning for Image Recognition", *in*: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 770–778.

Hochreiter, Sepp and Jürgen Schmidhuber (1997), "Long Short-Term Memory", *in*: *Neural Computation* 9.*8*, pp. 1735–1780.

Hu, Weiwei and Ying Tan (2016), "Prototype Generation Using Multiobjective Particle Swarm Optimization for Nearest Neighbor Classification", *in*: *IEEE Transactions on Cybernetics* 46.*12*, pp. 2719–2731.

Impedovo, Sebastiano, Francesco Maurizio Mangini, and Donato Barbuzzi (2014), "A Novel Prototype Generation Technique for Handwriting Digit Recognition", *in*: *Pattern Recognition* 47.*3*, pp. 1002–1010.

Jacovi, Alon and Yoav Goldberg (2020), "Towards Faithfully Interpretable NLP Systems: How Should We Define and Evaluate Faithfulness?", *in*: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 4198–4205.

Jain, Sarthak and Byron C. Wallace (2019), "Attention Is Not Explanation", *in*: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pp. 3543–3556.

Johansson, Ulf and Lars Niklasson (2009), "Evolving Decision Trees Using Oracle Guides", *in*: *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2009, Part of the IEEE Symposium Series on Computational Intelligence 2009, Nashville, TN, USA, March 30, 2009 - April 2, 2009*, pp. 238–244.

Kanal, L. N. and N. C. Randall (1964), "Recognition System Design by Statistical Analysis", *in*: *Proceedings of the 1964 19th ACM National Conference On* -, The 1964 19th ACM National Conference, pp. 42.501–42.5020.

Karim, Fazle, Somshubra Majumdar, Houshang Darabi, and Shun Chen (2018), "LSTM Fully Convolutional Networks for Time Series Classification", *in*: *IEEE Access* 6, pp. 1662–1669.

Karlsson, Isak, Panagiotis Papapetrou, and Henrik Boström (Sept. 2016), "Generalized Random Shapelet Forests", *in*: *Data Mining and Knowledge Discovery* 30.*5*, pp. 1053–1085.

Kehl, Danielle, Priscilla Guo, and Samuel Kessler (2017), "Algorithms in the Criminal Justice System: Assessing the Use of Risk Assessments in Sentencing", *in*: *Berkman Klein Center for Internet & Society*, p. 37.

Keogh, Eamonn, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra (Aug. 2001), "Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases", *in*: *Knowledge and Information Systems* 3.*3*, pp. 263–286.

Kodratoff, Yves (1994), "The Comprehensibility Manifesto", *in*: *AI Communications* 7.*2*, pp. 83–85.

Korn, Flip, H. V. Jagadish, and Christos Faloutsos (1997), "Efficiently Supporting Ad Hoc Queries in Large Datasets of Time Sequences", *in*: *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pp. 289–300.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012), "ImageNet Classification with Deep Convolutional Neural Networks", *in*: *Advances in Neural Information Processing Systems*, vol. 25.

Lan, Zhenzhong, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut (2020), "ALBERT: A Lite BERT for Self-Supervised Learning of Language Representations", *in*: *International Conference on Learning Representations (ICLR)*.

Large, James, Jason Lines, and Anthony J. Bagnall (2019), "A Probabilistic Classifier Ensemble Weighting Scheme Based on Cross-Validated Accuracy Estimates", *in*: *Data Mining and Knowledge Discovery* 33.*6*, pp. 1674–1709.

Le Nguyen, Thach, Severin Gsponer, Iulia Ilie, Martin O'Reilly, and Georgiana Ifrim (July 2019), "Interpretable Time Series Classification Using Linear Models and Multi-Resolution Multi-Domain Symbolic Representations", *in*: *Data Mining and Knowledge Discovery* 33.*4*, pp. 1183–1222.

Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998), "Gradient-Based Learning Applied to Document Recognition", *in*: *Proceedings of the IEEE* 86.*11*, pp. 2278–2324.

Legendre, Adrien-Marie (1805), *Nouvelles Méthodes Pour La Détermination Des Orbites Des Comètes*.

Lei, Tao (2017), "Interpretable Neural Models for Natural Language Processing", Massachusetts Institute of Technology, Cambridge, USA.

Leverger, Colin, Simon Malinowski, Thomas Guyet, Vincent Lemaire, Alexis Bondu, and Alexandre Termier (2019), "Toward a Framework for Seasonal Time Series Forecasting Using Clustering", *in*: *Intelligent Data Engineering and Automated Learning – IDEAL 2019*, pp. 328–340.

Li, Oscar, Hao Liu, Chaofan Chen, and Cynthia Rudin (2018), "Deep Learning for Case-Based Reasoning through Prototypes: A Neural Network That Explains Its Predictions", *in*: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th Innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pp. 3530–3537.

Lin, Huiwei, Yunming Ye, Ka-Cheong Leung, and Bowen Zhang (2020), "A Multivariate Time Series Classification Method Based on Self-Attention", *in*: *Genetic and Evolutionary Computing*, pp. 491–499.

Lin, Jessica, Eamonn Keogh, Stefano Lonardi, and Bill Chiu (2003), "A Symbolic Representation of Time Series, with Implications for Streaming Algorithms", *in*: *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pp. 2–11.

Lin, Jessica, Eamonn Keogh, Li Wei, and Stefano Lonardi (Oct. 2007), "Experiencing SAX: A Novel Symbolic Representation of Time Series", *in*: *Data Mining and Knowledge Discovery* 15.*2*, pp. 107–144.

Lin, Jessica, Rohan Khade, and Yuan Li (2012), "Rotation-Invariant Similarity in Time Series Using Bag-of-Patterns Representation", *in*: *Journal of Intelligent Information Systems* 39.*2*, pp. 287–315.

Lines, Jason and Anthony Bagnall (May 2015), "Time Series Classification with Ensembles of Elastic Distance Measures", *in*: *Data Mining and Knowledge Discovery* 29.*3*, pp. 565–592.

Lines, Jason, Luke M. Davis, Jon Hills, and Anthony Bagnall (2012), "A Shapelet Transform for Time Series Classification", *in*: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '12*, The 18th ACM SIGKDD International Conference, p. 289.

Lines, Jason, Sarah Taylor, and Anthony J. Bagnall (2016), "HIVE-COTE: The Hierarchical Vote Collective of Transformation-Based Ensembles for Time Series Classification", *in*: *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain*, pp. 1041–1046.

Lipton, Zachary C. (Sept. 26, 2018), "The Mythos of Model Interpretability", *in*: *Communications of the ACM* 61.*10*, pp. 36–43.

Lods, Arnaud, Simon Malinowski, Romain Tavenard, and Laurent Amsaleg (2017), "Learning DTW-Preserving Shapelets", *in*: *Advances in Intelligent Data Analysis XVI*, vol. 10584, pp. 198–209.

Lundberg, Scott M. and Su-In Lee (2017), "A Unified Approach to Interpreting Model Predictions", *in*: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 4765–4774.

Malinowski, Simon, Thomas Guyet, René Quiniou, and Romain Tavenard (2013), "1d-SAX: A Novel Symbolic Representation for Time Series", *in*: *Proceedings of the International Symposium on Intelligent Data Analysis*, pp. 273–284.

Martens, David, Jan Vanthienen, Wouter Verbeke, and Bart Baesens (2011), "Performance of Classification Models from a User Perspective", *in*: *Decision Support Systems* 51.*4*, pp. 782–793.

McCulloch, Warren S. and Walter Pitts (Dec. 1943), "A Logical Calculus of the Ideas Immanent in Nervous Activity", *in*: *The Bulletin of Mathematical Biophysics* 5.*4*, pp. 115–133.

Michie, Donald (1987), "Current Developments in Expert Systems", *in*: *Proceedings of the Second Australian Conference on Applications of Expert Systems*, pp. 137–156.

Middlehurst, Matthew, James Large, and Anthony J. Bagnall (2020a), "The Canonical Interval Forest (CIF) Classifier for Time Series Classification", *in*: *IEEE International Conference on Big Data, Big Data 2020, Atlanta, GA, USA, December 10-13, 2020*, pp. 188–195.

Middlehurst, Matthew, James Large, Gavin C. Cawley, and Anthony J. Bagnall (2020b), "The Temporal Dictionary Ensemble (TDE) Classifier for Time Series Classification", *in*: *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2020, Ghent, Belgium, September 14-18, 2020, Proceedings, Part I*, vol. 12457, pp. 660–676.

Middlehurst, Matthew, James Large, Michael Flynn, Jason Lines, Aaron Bostrom, and Anthony Bagnall (Apr. 15, 2021), *HIVE-COTE 2.0: A New Meta Ensemble for Time Series Classification*, URL: `http://arxiv.org/abs/2104.07551` (visited on 04/18/2021).

Miller, Tim (2019), "Explanation in Artificial Intelligence: Insights from the Social Sciences", *in*: *Artificial Intelligence* 267, pp. 1–38.

Molnar, Christoph (Mar. 24, 2019), *Interpretable Machine Learning*, 318 pp.

Moraffah, Raha, Mansooreh Karami, Ruocheng Guo, Adrienne Raglin, and Huan Liu (2020), "Causal Interpretability for Machine Learning - Problems, Methods and Evaluation", *in*: *SIGKDD Explorations* 22.*1*, pp. 18–33.

O'neil, Cathy (2016), *Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy*.

Olah, Chris, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev (Mar. 6, 2018), "The Building Blocks of Interpretability", *in*: *Distill* 3.*3*, e10.

Pearl, Judea and Dana Mackenzie (2018), *The Book of Why: The New Science of Cause and Effect*.

Pelletier, Charlotte, Geoffrey I. Webb, and François Petitjean (2019), "Temporal Convolutional Neural Network for the Classification of Satellite Image Time Series", *in*: *Remote. Sens.* 11.*5*, p. 523.

Popivanov, I. and R.J. Miller (2002), "Similarity Search over Time-Series Data Using Wavelets", *in*: *Proceedings 18th International Conference on Data Engineering*, 18th International Conference on Data Engineering, pp. 212–221.

Pretrial Justice Institute (Oct. 2, 2019), *Scan of Pretrial Practices 2019*.

Quinlan, J. Ross (1987), "Simplifying Decision Trees", *in*: *International Journal of Man-Machine Studies* 27.*3*, pp. 221–234.

Rakthanmanon, Thanawin and Eamonn Keogh (May 2013), "Fast Shapelets: A Scalable Algorithm for Discovering Time Series Shapelets", *in*: *Proceedings of the 2013 SIAM International Conference on Data Mining*, pp. 668–676.

Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin (2016), ""Why Should I Trust You?": Explaining the Predictions of Any Classifier", *in*: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, The 22nd ACM SIGKDD International Conference, pp. 1135–1144.

Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin (2018), "Anchors: High Precision Model-Agnostic Explanations", *in*: *Thirty-Second AAAI Conference on Artificial Intelligence*, p. 9.

Riley, Patrick (Aug. 2019), "Three Pitfalls to Avoid in Machine Learning", *in*: *Nature* 572.*7767*, pp. 27–29.

Rojat, Thomas, Raphaël Puget, David Filliat, Javier Del Ser, Rodolphe Gelin, and Natalia Díaz-Rodríguez (Apr. 2, 2021), *Explainable Artificial Intelligence (XAI) on TimeSeries Data: A Survey*, URL: `http://arxiv.org/abs/2104.00950` (visited on 04/20/2021).

Rudin, Cynthia (May 2019), "Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead", *in*: *Nature Machine Intelligence* 1.*5* (5), pp. 206–215.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (Oct. 1986), "Learning Representations by Back-Propagating Errors", *in*: *Nature* 323.*6088*, pp. 533–536.

Rüping, Stefan (2006), "Learning Interpretable Models", PhD thesis, Technical University of Dortmund, Germany.

Russell, Stuart J. and Peter Norvig (2009), *Artificial Intelligence: A Modern Approach*, 3rd ed.

Sakoe, Hiroaki and Seibi Chiba (1978), "Dynamic Programming Algorithm Optimization for Spoken Word Recognition", *in*: *IEEE Transactions on Acoustics, Speech and Signal Processing* 26.*1*, pp. 43–49.

Saralajew, Sascha (2020), "New Prototype Concepts in Classification Learning", Bielefeld University, Germany.

Savage, Neil (2019), "Neural Net Worth", *in*: *Communications of the ACM* 62.*6*, pp. 10–12.

Schäfer, Patrick (Nov. 2015), "The BOSS Is Concerned with Time Series Classification in the Presence of Noise", *in*: *Data Mining and Knowledge Discovery* 29.*6*, pp. 1505–1530.

Selvaraju, Ramprasaath R., Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra (2017), "Grad-Cam: Visual Explanations from Deep Networks via Gradient-Based Localization", *in*: *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pp. 618–626.

Shapley, Lloyd S (1953), "A Value for N-Person Games", *in*: *Contributions to the Theory of Games* 2.*28*, pp. 307–317.

Shen, Owen (2020), "Interpretability in ML: A Broad Overview", *in*: *The Gradient*.

Shlens, Jonathon (Apr. 3, 2014), *A Tutorial on Principal Component Analysis*, URL: `http://arxiv.org/abs/1404.1100` (visited on 04/22/2021).

Shrikumar, Avanti, Peyton Greenside, and Anshul Kundaje (2017), "Learning Important Features through Propagating Activation Differences", *in*: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, vol. 70, pp. 3145–3153.

Shumway, Robert H. and David S. Stoffer (2005), *Time Series Analysis and Its Applications (Springer Texts in Statistics)*.

Siffer, Alban, Pierre-Alain Fouque, Alexandre Termier, and Christine Largouet (2017), "Anomaly Detection in Streams with Extreme Value Theory", *in*: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '17*, The 23rd ACM SIGKDD International Conference, pp. 1067–1075.

Silver, David et al. (2016), "Mastering the Game of Go with Deep Neural Networks and Tree Search", *in*: *Nature* 529.*7587*, pp. 484–489.

Simonyan, Karen, Andrea Vedaldi, and Andrew Zisserman (2014), "Deep inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", *in*:

2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Workshop Track Proceedings.

Smilkov, Daniel, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg (June 12, 2017), *SmoothGrad: Removing Noise by Adding Noise*, URL: `http://arxiv.org/abs/1706.03825` (visited on 10/02/2020).

Springenberg, Jost Tobias, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller (2015), "Striving for Simplicity: The All Convolutional Net", *in*: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*.

Sundararajan, Mukund, Ankur Taly, and Qiqi Yan (2017), "Axiomatic Attribution for Deep Networks", *in*: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, vol. 70, pp. 3319–3328.

Szegedy, Christian, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi (2017), "Inception-v4, Inception-Resnet and the Impact of Residual Connections on Learning", *in*: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pp. 4278–4284.

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich (2015), "Going Deeper with Convolutions", *in*: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pp. 1–9.

Tavenard, Romain (2020), "Apprentissage statistique et séries temporelles", HDR thesis, Laboratoire LETG, UMR CNRS 6554.

Tavenard, Romain and Laurent Amsaleg (Jan. 2015), "Improving the Efficiency of Traditional DTW Accelerators", *in*: *Knowledge and Information Systems* 42.*1*, pp. 215–243.

Tavenard, Romain, Simon Malinowski, Laetitia Chapel, Adeline Bailly, Heider Sanchez, and Benjamin Bustos (Sept. 2017), "Efficient Temporal Kernels between Feature Sets for Time Series Classification", *in*: *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery*, pp. 528–543.

Tavenard, Romain et al. (2020), "Tslearn, a Machine Learning Toolkit for Time Series Data", *in*: *Journal of Machine Learning Research* 21.*118*, pp. 1–6.

Tay, Yi, Mostafa Dehghani, Dara Bahri, and Donald Metzler (Sept. 16, 2020), *Efficient Transformers: A Survey*, URL: http://arxiv.org/abs/2009.06732 (visited on 09/29/2020).

Telgarsky, Matus (2016), "Benefits of Depth in Neural Networks", *in*: *Proceedings of the 29th Conference on Learning Theory, COLT 2016, New York, USA, June 23-26, 2016*, vol. 49, pp. 1517–1539.

Triguero, Isaac, Joaquín Derrac, Salvador García, and Francisco Herrera (2012), "A Taxonomy and Experimental Study on Prototype Generation for Nearest Neighbor Classification", *in*: *IEEE Trans. Syst. Man Cybern. Part C* 42.*1*, pp. 86–100.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin (2017), "Attention Is All You Need", *in*: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008.

Von Luxburg, Ulrike (2007), "A Tutorial on Spectral Clustering", *in*: *Statistics and Computing* 17.*4*, pp. 395–416.

Wang, Lin, Faming Lu, Minghao Cui, and Yunxia Bao (2019), "Survey of Methods for Time Series Symbolic Aggregate Approximation", *in*: *Data Science*, pp. 645–657.

Wang, Yichang, Rémi Emonet, Élisa Fromont, Simon Malinowski, and Romain Tavenard (2020), "Adversarial Regularization for Explainable-by-Design Time Series Classification", *in*: *32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020, Baltimore, MD, USA, November 9-11, 2020*, pp. 1079–1087.

Wang, Zhiguang, Weizhong Yan, and Tim Oates (May 2017), "Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline", *in*: *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017 International Joint Conference on Neural Networks (IJCNN), pp. 1578–1585.

Wiegreffe, Sarah and Yuval Pinter (2019), "Attention Is Not Not Explanation", *in*: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pp. 11–20.

Xu, Kelvin, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio (2015), "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", *in*: *Proceedings of the 32nd Inter-*

national Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015, vol. 37, pp. 2048–2057.

Ye, Lexiang and Eamonn Keogh (2009), "Time Series Shapelets: A New Primitive for Data Mining", *in*: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 947–956.

Yuan, Ming and Yi Lin (Feb. 2006), "Model Selection and Estimation in Regression with Grouped Variables", *in*: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68.*1*, pp. 49–67.

Zhang, Heng, Élisa Fromont, Sébastien Lefèvre, and Bruno Avignon (2020), "Localize to Classify and Classify to Localize: Mutual Guidance in Object Detection", *in*: *Computer Vision - ACCV 2020 - 15th Asian Conference on Computer Vision, Kyoto, Japan, November 30 - December 4, 2020, Revised Selected Papers, Part IV*, vol. 12625, pp. 104–118.

Zhou, B., A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba (June 2016), "Learning Deep Features for Discriminative Localization", *in*: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2921–2929.

Zhou, Zhi-Hua (2012), *Ensemble Methods: Foundations and Algorithms*, 222 pp.

Zuo, Jingwei, Karine Zeitouni, and Yehia Taher (2021), "SMATE: Semi-Supervised Spatio-Temporal Representation Learning on Multivariate Time Series", *in*: *CoRR* abs/2110.00578.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre : Classification interprétable de séries temporelles

Mot clés : Séries temporelles, interprétabilité, réseaux de neurones convolutifs, apprentissage antagoniste, intelligence artificielle explicable

Résumé : Nous étudions différentes méthodes pouvant être utilisées pour expliquer les décisions prises par les modèles de classification des séries temporelles. Nous supposons que, dans le cas des séries temporelles, les meilleures explications doivent prendre la forme de sous-séries (également appelées shapelets) puisqu'il s'agit d'un "langage" intelligible et expressif pour un utilisateur s'intéressant à ce type de séries.

Bien que certaines méthodes de l'état de l'art permettent d'apprendre des shapelets discriminantes automatiquement, nous constatons qu'elles ne sont pas toujours similaires aux morceaux d'une série réelle existante. Il est donc difficile de les utiliser pour expliquer la décision du classificateur à un utilisateur qui pourrait être dérouté par ce langage d'explication éloigné des séries qu'il connait. Nous proposons une méthode innovante qui permet, grâce à un réseau convolutif simple, de classer des séries temporelles et nous introduisons une régularisation antagoniste pour contraindre le modèle à apprendre des shapelets interprétables.

Nos résultats de classification sur de nombreux jeux de données de séries temporelles univariées, sont comparables, en terme de précision, aux meilleurs résultats obtenus par les algorithmes de classification basés sur les shapelets. Cependant, nous montrons, en comparant avec d'autres méthodes d'explication sur des modèles de type "boîte noire", que notre régularisation antagoniste permet d'apprendre des shapelets qui sont, par conception, mieux adaptées pour expliquer les décisions et cela pour plusieurs niveaux d'explication.

Title: Interpretable time series classification

Keywords: Time series, interpretability, convolutional neural networks, adversarial training, explainable artificial intelligence

Abstract: In this thesis, we will study different existing methods that can be used to explain decisions taken by time series classification models. We argue that, in the case of time series, the best explanations should take the form of sub-series (also called shapelets) since it is "pattern language" familiar to a time series user.

We review state-of-the-art classification methods that can jointly learn a shapelet-based representation of the series in the dataset and classify the series according to this representation. However, although the learned shapelets are discriminative, they are not always similar to pieces of a real series in the dataset. This makes them difficult to use to explain the classifier's decision. We make use of a simple convolutional network to tackle the time series classification task and we introduce an adversarial regularization to constrain the model to learn meaningful shapelets.

Our classification results, on many univariate time series benchmark datasets, are comparable with the results obtained by state-of-the-art shapelet-based classification algorithms. However, we show, by comparing to other black box explanation methods that our adversarially regularized method learns shapelets that are, by design, better suited to explain decisions.