SelF-Rocket : une méthode de classification de séries temporelles à noyaux de convolution aléatoires avec sélection des représentations d'entrée et de l'opérateur d'agrégation

Mouhamadou Mansour LO[1], Gildas MORVAN[2]
Mathieu ROSSI[1], Fabrice MORGANTI[1], David MERCIER[2]

[1] Univ. Artois, UR 4025, LSEE, F-62400 Béthune, France
[2] Univ. Artois, UR 3926, LGI2A, F-62400 Bethune, France

prenom.nom@univ-artois.fr

Résumé

Cet article présente une nouvelle approche basée sur MiniRocket, appelée SelF-Rocket (Selected Features Rocket), pour une classification rapide des séries temporelles. Contrairement aux approches existantes utilisant aussi des convolutions aléatoires, elle sélectionne dynamiquement le meilleur couple de représentations de la série temporelle et d'opérateur d'agrégation pendant le processus d'apprentissage, ce qui permet d'obtenir de très bonnes performances sur les jeux de données de l'Université de Californie Riverside (UCR).

Mots-clés

Classification de séries temporelles, Noyaux de convolution aléatoires, ROCKET, MiniRocket, Opérateurs d'agrégation, Sélection de caractéristiques.

Abstract

This paper introduces SelF-Rocket (Selected Features Rocket), a novel method based on random convolutions for fast time series classification. It dynamically chooses the optimal set of input representations and pooling operator during training, in contrast to current methods based on random convolution kernels. SelF-Rocket achieves state-of-the-art accuracy on the University of California Riverside (UCR) benchmark datasets.

Keywords

Time series classification, ROCKET, MiniRocket, Pooling operators, Input representation, Feature selection.

1 Introduction

Les transformations utilisant des noyaux de convolution aléatoires, ROCKET [2] et ses successeurs comme MiniRocket [3], MultiRocket [16] ou Hydra [4] ont apporté des avancées notables dans le domaine de la classification des séries temporelles (CST), offrant un compromis remarquable entre vitesse et exactitude (*accuracy*) [14].

L'idée principale des méthodes ROCKET repose sur l'utilisation d'un grand nombre de noyaux de convolution aléatoires pour transformer une série temporelle avec un ou plusieurs opérateurs d'agrégation (en anglais, *pooling operators*) en un ensemble de caractéristiques permettant une classification efficace avec un classifieur linéaire simple, comme Ridge.

À notre connaissance, toutes les transformations de type ROCKET proposées s'appuient sur un ou plusieurs opérateurs d'agrégation fixes, tels que, par exemple, la Proportion de Valeurs Positives (PPV).

Dans cet article, nous proposons une nouvelle approche, nommée SelF-Rocket (pour **Sel**ected **F**eatures Rocket), s'appuyant sur MiniRocket avec une sélection des meilleures représentations des données en entrée et du meilleur opérateur d'agrégation possible lors d'une phase d'apprentissage. Une implémentation Python 3 de SelF-Rocket, ainsi que le code nécessaire pour reproduire les résultats présentés dans cet article, est disponible sur GitHub. [1].

Cet article est organisé de la manière suivante. Les fondamentaux de la CST dont les principales méthodes utilisant les convolutions aléatoires sont d'abord rappelés dans la section 2, puis, afin de motiver l'intérêt de ce travail, nous illustrons dans la section 3 l'importance de la sélection de représentations d'entrée et d'un opérateur d'agrégation adéquat sur la base d'une étude expérimentale conduite sur les jeux de données de l'Université de Californie Riverside (UCR) [1]. Ces tests indiquent que, bien que PPV soit un opérateur d'agrégation de choix, il n'est pas toujours le plus performant dans la plupart des cas. La méthode SelF-Rocket est ensuite présentée dans la section 4 et ses performances sur les données UCR sont analysées dans

1. https://github.com/ANR-MYEL/SelF-Rocket/

la section 5. Enfin, la section 6 présente une conclusion et quelques perspectives.

2 Classification de Séries Temporelles (CST)

2.1 Vue d'ensemble

Dans le cadre de la CST, les séries temporelles à classer sont constituées de séquences d'observations collectées à intervalles de temps constants. Formellement, une série temporelle $\mathbf{X}$ est représentée par un ensemble $\mathbf{X} = \{x_1, x_2, \ldots, x_T\}$, où x_t représente la valeur de la série temporelle au pas de temps t, et T représente la longueur de la série temporelle. Cette notation est utilisée pour désigner une série temporelle tout au long de cet article.

Les méthodes traditionnelles de CST comprennent des techniques utilisant une distance comme le Dynamic Time Warping (DTW) [15], mesurant une similarité entre séries temporelles. Plus récemment, des approches d'apprentissage profond telles que InceptionTime [9] ont été proposées. Les approches hybrides comme HIVE-COTE v2.0, qui combinent plusieurs classifieurs de natures différentes, sont généralement les plus performantes mais avec un coût calculatoire très élevé [13]. Pour plus de détails, un état de l'art récent des méthodes de CST a été proposé par Middlehurst et al. [14].

Dans la section suivante, nous nous concentrons sur un type spécifique de méthodes de CST utilisant des noyaux de convolution aléatoires pour extraire des caractéristiques des données de séries temporelles.

2.2 Méthodes basées sur les convolutions aléatoires

Comme mentionné ci-dessus, ces méthodes utilisent des convolutions aléatoires pour discriminer les séries temporelles. Les noyaux de convolution peuvent être considérés comme des mini-séquences aléatoires dont la convolution avec les séries temporelles génère des cartes d'activation utilisées pour extraire plusieurs caractéristiques synthétiques qui sont ensuite introduites dans un classifieur linéaire. La phase d'apprentissage consiste à apprendre les poids appropriés attribués à chaque caractéristique, ce qui permet à ces méthodes d'être, au moment où cet article est écrit, parmi les plus rapides tout en garantissant de très bonnes performances [14].

ROCKET (*Random Convolutional Kernel Transform*) [2] est le premier algorithme de ce type à avoir été introduit. Il génère aléatoirement un grand nombre de noyaux (typiquement 10000) qu'il utilise pour créer des cartes d'activation. Ces cartes sont ensuite résumées par deux opérateurs d'agrégation : PPV (*Proportion de valeurs positives*), qui calcule le pourcentage de valeurs positives, et GMP (*Valeur d'agrégation maximale*), qui extrait la valeur maximale de la carte d'activation. Formellement :

$$PPV(Z) = \frac{1}{n}\sum_{i=1}^{n}[z_i > 0], \tag{1}$$

$$GMP(Z) = max(Z), \tag{2}$$

où Z est la sortie de convolution de X.

Pour chaque noyau, deux caractéristiques sont extraites à l'aide de PPV et GMP. Les noyaux ont une longueur choisie aléatoirement dans $\{7, 9, 11\}$. Les poids $w \sim \mathcal{N}(0, 1)$ sont choisis puis normalisés t.q. $w = W - \overline{W}$. Un biais $b \sim \mathcal{U}(-1, 1)$ est ajouté à la carte d'activation. Les dilatations sont calculées avec $d = \lfloor 2^x \rfloor$, où $x \sim \mathcal{U}(0, A)$ et $A = \log_2\left(\frac{\ell_{entrée}-1}{\ell_{noyau}-1}\right)$, où $\ell_{entrée}$ est la longueur de la série temporelle d'entrée, et ℓ_{noyau} la longueur du noyau.

MiniRocket [3] est une variante de ROCKET qui introduit plusieurs modifications clés accélérant considérablement le temps d'apprentissage sans sacrifier les performances. La longueur du noyau est fixée à 9 au lieu de $\{7, 9, 11\}$. Un seul ensemble de 84 noyaux, contenant uniquement $\{-1, 2\}$ comme valeurs, est utilisé. Cette modification majeure réduit grandement le temps de calcul. Le biais est dérivé du résultat de la convolution avec une paire noyau/dilatation. Seul PPV est utilisé (GMP n'est plus calculé).

MultiRocket [16] étend MiniRocket en introduisant plusieurs améliorations. La différence de premier ordre ($DIFF$) est utilisée comme représentation d'entrée supplémentaire. Formellement :

$$DIFF(X) = \{x_t - x_{t-1} : \forall t \in \{2, ..., T\}\} \tag{3}$$

En plus de la PPV, les opérateurs d'agrégation suivants sont utilisés : Moyenne des valeurs positives (MPV), Moyenne des indices de valeurs positives (MIPV) et Plus long intervalle de valeurs positives (LSPV) :

$$MPV(Z) = \frac{1}{m}\sum_{i=1}^{m} z_i^+, \tag{4}$$

où $Z^+ = \{z_1^+, ...z_m^+\}$ est le sous-ensemble de valeurs positives dans Z,

$$MIPV(Z) = \begin{cases} \frac{1}{m}\sum_{j=1}^{m} i_j^+ & \text{si } m > 0 \\ -1 & \text{sinon} \end{cases}, \tag{5}$$

où $I^+ = \{i_1^+, ...i_m^+\}$ indique les indices de valeurs positives, et

$$LSPV(Z) = max(j - i \mid \forall_{i \le k \le j} z_k > 0). \tag{6}$$

Hydra (*Hybrid Dictionary-ROCKET Architecture*) [4] combine des aspects des méthodes utilisant des dictionnaires et des convolutions aléatoires. L'idée générale est de regrouper tous les noyaux en groupes de

noyaux. Pour chaque série temporelle, la convolution est effectuée avec tous les noyaux d'un groupe donné. Pour chaque point de la série, le noyau conduisant à la valeur de convolution la plus élevée est sélectionné. Ainsi, pour chaque groupe, un histogramme des réponses maximales du noyau est créé, servant de caractéristiques pour le classifieur. La différence du premier ordre est également utilisée. Les noyaux sont paramétrés avec une longueur fixée à 9, des poids $w \sim \mathcal{N}(0,1)$, et aucun biais ou opérateur d'agrégation n'est utilisé après la convolution.

Les caractéristiques générées par Hydra et MultiRocket peuvent être concaténées, ce qui permet d'obtenir une classification plus précise [4].

3 Impact de la sélection de représentations et d'un opérateur d'agrégation adéquats

Pour comprendre comment les représentations d'entrée et les opérateurs d'agrégation impactent les performances de cette famille ROCKET de transformations utilisant des convolutions aléatoires, nous avons comparé sur 112 jeux de données UCR [1], MiniRocket dans sa version originale avec des versions modifiées utilisant GMP, MPV, MIPV et LSPV au lieu de PPV comme opérateur d'agrégation et la différence de premier ordre (DIFF) comme représentation d'entrée supplémentaire, conduisant à 15 transformations possibles détaillées dans la section 4.1 (5 opérateurs d'agrégations $\times$ 3 ensembles de représentations d'entrée possibles : $\{I\}, \{DIFF\}, \{I, DIFF\}$). Les classifieurs sont désignés par la notation **PO_IR** en fonction de l'opérateur d'agrégation **PO** utilisé ($PO \in \{PPV, GMP, MPV, MIPV, LSPV\}$) et de la représentation d'entrée **IR** utilisée (MIX désignant la concaténation de la différence du premier ordre DIFF et de la représentation de base). Les définitions formelles sont données dans la section 4.1. Il convient de noter que nous avons également effectué des tests avec plusieurs opérateurs d'agrégation, au lieu d'en choisir un seul comme indiqué ici, mais les résultats préliminaires nous ont montré que les performances obtenues n'étaient pas aussi bonnes que celles exposées dans la présente section.

La Figure 1 montre le nombre de jeux de données UCR pour lesquels la version modifiée de MiniRocket est la plus performante. Si PPV_MIX est la meilleure transformation en moyenne, elle est surpassée par les autres dans la plupart des cas (78.57%).

Ces expériences nous permettent de constater que le couple (représentations d'entrée, opérateur d'agrégation) conduisant à la meilleure performance varie en fonction du jeu de données. Il serait donc intéressant de pouvoir sélectionner le couple approprié à partir d'un jeu de données d'entraînement. C'est l'objectif de la méthode SelF-Rocket présentée dans la section suivante.

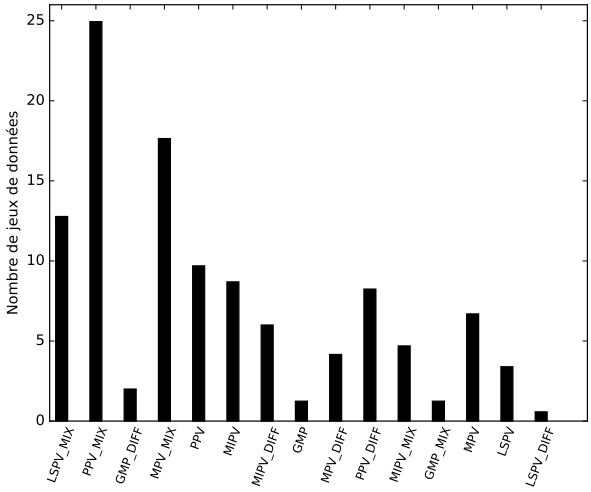


FIGURE 1 – Nombre de jeux de données UCR pour lesquels la version modifiée de MiniRocket est la plus performante

4 SelF-Rocket

Dans cette section, nous introduisons une nouvelle variante de ROCKET, appelée SelF-Rocket (**Sel**ected **F**eatures Rocket), qui vise à sélectionner dynamiquement le meilleur couple (représentation d'entrée, opérateur d'agrégation) pendant le processus d'entraînement qui est illustré sur la Figure 2. Ce dernier comporte trois étapes principales : l'étape de génération de l'ensemble des caractéristiques, détaillée dans la section 4.1, l'étape de sélection des caractéristiques, exposée dans la section 4.2 et l'étape de classification, présentée dans la section 4.3.

SelF-Rocket fonctionne de la même manière que MiniRocket ou MultiRocket. La seule différence majeure est l'ajout de cette étape de sélection du meilleur couple (représentations d'entrée, opérateur d'agrégation). Comme nous l'avons vu précédemment dans la section 3, ce couple optimal varie d'un jeu de données à l'autre, de sorte que l'utilisation d'un unique opérateur d'agrégation (PPV) comme MiniRocket ou d'un ensemble fixe de plusieurs opérateurs ({PPV, LSPV, MIPV, MPV}) comme MultiRocket peut produire de bons résultats en moyenne, mais pas optimal pour chaque cas. Pour résoudre ce problème, nous mettons en œuvre un module de sélection de ce couple avec pour objectif de sélectionner le couple le plus approprié pour chaque problème de classification.

4.1 Génération de l'ensemble des caractéristiques

Soit **IR** un ensemble de représentations d'entrée, **PO** un ensemble d'opérateurs d'agrégation, **MK** l'ensemble de tous les noyaux (contenant toutes les combinaisons noyau/dilatation) générés par MiniRocket et

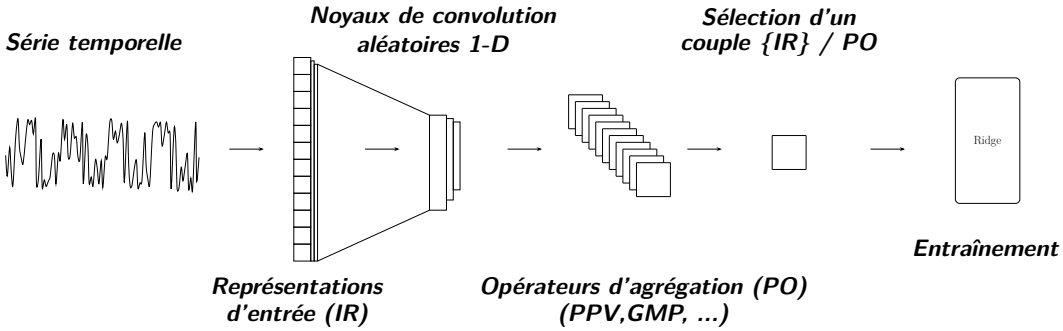


FIGURE 2 – Architecture d'entraînement de SelF-Rocket

b_κ le biais associé à un noyau $\kappa \in \mathbf{MK}$ (cf. Section 2.2). Dans l'implémentation évaluée dans cet article, nous considérons

$$\mathbf{IR} = \{I, DIFF\} \tag{7}$$

et

$$\mathbf{PO} = \{PPV, GMP, MPV, MIPV, LSPV\}, \tag{8}$$

où I est la fonction d'identité ($I(X) = X$ pour toutes les séries temporelles X) et $DIFF$ est la différence du premier ordre (cf. Équation 3).

Avec f une fonction de génération d'un ensemble de caractéristiques définie par

$$f(X, A, p) = \{p(r(X) \circledast \kappa - b_\kappa) \,|\, \kappa \in \mathbf{MK}, r \in A\}, \tag{9}$$

où X est une série temporelle, $A \subseteq 2^{\mathbf{IR}} \setminus \{\emptyset\}$, et $p \in \mathbf{PO}$, MiniRocket dans sa version originale, extrait les caractéristiques en utilisant la paramétrisation $f(X, \{I\}, PPV)$, pour n'importe quelle série temporelle X.
Le produit cartésien $2^{\mathbf{IR}} \setminus \{\emptyset\} \times \mathbf{PO}$ induit un ensemble de telles fonctions paramétrées. Dans l'implémentation évaluée dans cet article, $|2^{IR} \setminus \{\emptyset\}| = 3$ et $|PO| = 5$, ce qui conduit à $N = 3 \times 5 = 15$ paramétrisations possibles.
Soit **FV** le vecteur de caractéristiques généré. Dans la suite de l'article, pour des raisons de lisibilité, les résultats de la génération des caractéristiques sont désignés comme dans la section 3 :

- p pour $f(X, \{I\}, p)$,
- p_DIFF pour $f(X, \{DIFF\}, p)$,
- et p_MIX pour $f(X, \{I, DIFF\}, p)$,

pour tout $p \in \mathbf{PO}$.

4.2 Sélection des caractéristiques

Le module de sélection des caractéristiques, résumé dans la Figure 3 et dans l'Algorithme 1, s'appuie sur des méthodes de découpage stratifié du jeu de données d'entraînement en entraînement/test (*train/test split*). En fonction de la taille de l'ensemble d'apprentissage, nous utilisons soit une méthode stratifiée k-Fold répétée nr fois, soit une méthode stratifiée shuffle split avec $k \times nr$ découpages. Un shuffle split stratifié est utilisé si l'ensemble de données contient au moins un nombre suffisamment important de données (désigné par ***mds*** dans l'Algorithme 1). Il convient de noter que cette valeur a un impact direct sur le temps disponible pour trouver la meilleure combinaison IR-PO en limitant la taille maximale de chaque ensemble d'entraînement et de validation, et donc le temps disponible pour cette tâche. Cette stratégie garantit que, à mesure que la taille de l'ensemble de données augmente, les ensembles d'apprentissage et de validation vont varier de manière significative. Elle permet des évaluations plus fiables des performances des différentes combinaisons. Ces ensembles sont ensuite utilisés pour former $k \times nr$ mini-classifieurs pour chaque combinaison IR-PO.
Ensuite, la sélection de la combinaison IR-PO optimale sur l'ensemble des découpages de l'ensemble d'entraînement original est effectuée grâce à un système de vote à l'aide de la méthode de meilleure médiane. Il est préférable d'utiliser la médiane la plus élevée pour sélectionner un IR-PO plutôt que la moyenne la plus élevée, car elle est plus résistante aux valeurs extrêmes qui peuvent se produire. Ce vote est ensuite validé (cf Algorithme 2) avant d'être utilisé comme ensemble final de caractéristiques pour le classifieur linéaire. L'idée de cet algorithme est de vérifier si la combinaison IR-PO sélectionnée est suffisamment soutenue par les votants, sinon une combinaison par défaut est choisie. Son objectif est d'éviter de choisir une combinaison IR-PO peu généralisable, en particulier pour les petits ensembles de données. Dans notre implémentation, l'Algorithme 2 vérifie si la combinaison IR-PO sélectionnée fait partie des valeurs les plus élevées pour chaque votant (par exemple, dans les 4 les plus élevés pour chaque votant) avec un certain degré de flexibilité (par exemple une présence ≥ 0.95 pour l'ensemble des votants, implémenté comme seuil ***thresh*** dans l'Algorithme 2), sinon la combinaison IR-PO sélectionnée est remplacée par une combinaison par défaut (PPV_MIX dans notre cas).

4.3 Classification

SelF-Rocket utilise le même algorithme pour les mini-classifieurs intégrés dans le module de sélection des caractéristiques et le classifieur de fin d'étape, à savoir

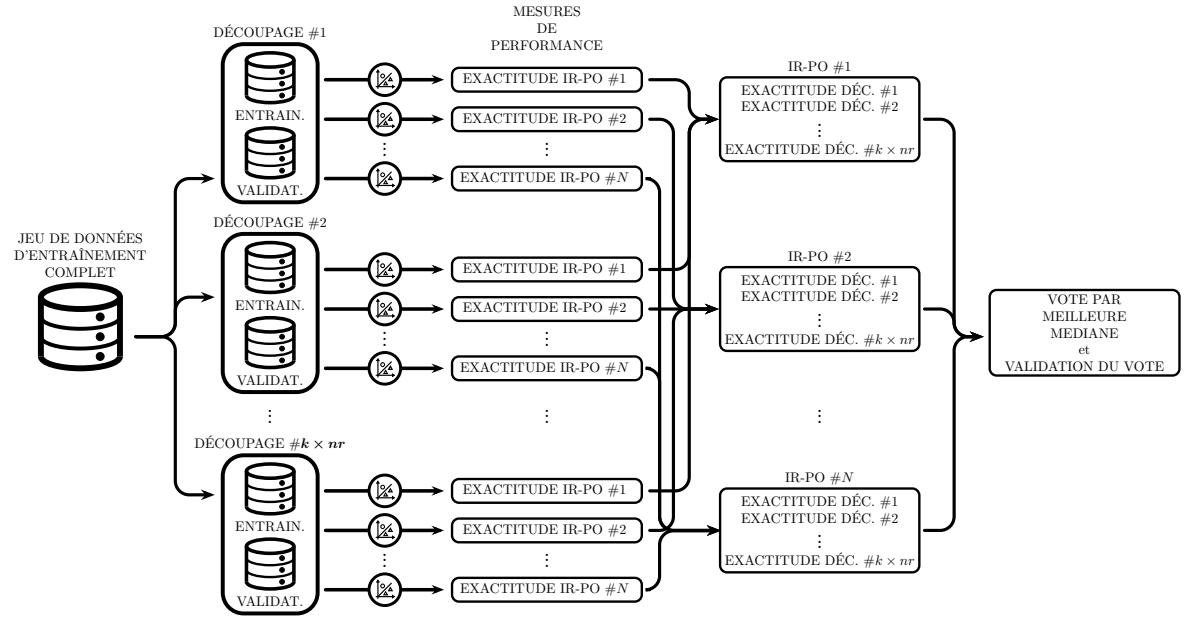


FIGURE 3 – Vue d'ensemble du module de sélection de caractéristiques de SelF-Rocket

le classifieur Ridge, tel que proposé par Dempster et al. [2]. Ce classifieur est préférable aux méthodes de descente de gradient stochastique, telles que la régression logistique, lorsque le nombre de caractéristiques est supérieur au nombre d'exemples d'apprentissage et lorsque l'on travaille avec de petits ensembles de données, par exemple lorsqu'il y a moins de 10000 exemples.

5 Expérimentations

Dans cette section, les performances de SelF-Rocket et de Hydra + SelF-Rocket (concaténation des caractéristiques de Hydra et de SelF-Rocket) sont évaluées. Nous montrons que SelF-Rocket est aussi précis que Multi-Rocket, bien qu'il ne dispose que d'un seul type de caractéristiques, et que le temps de prédiction du classifieur est relativement plus court. Nous étudions également l'influence des principaux paramètres de SelF-Rocket sur ses performances.

5.1 Protocole expérimental

Les méthodes proposées ont été évaluées sur un sous-ensemble de jeux de données de la base UCR [1], qui est largement utilisé dans ce domaine. Pour garantir la reproductibilité de nos expériences et une comparaison équitable avec d'autres algorithmes CST, nous avons utilisé les mêmes 112 ensembles de données que ceux utilisés par [14] et les mêmes 30 échantillonnages d'entraînement/test pour chacun de ces ensembles de données. Les 112 ensembles de données sont ceux des 128 ensembles originaux qui ne contenaient pas de valeurs manquantes ou de séries temporelles de longueur variable.

Les résultats des autres méthodes présentées dans les figures 4 et 5 sont issus [2] de [14].

L'implémentation originale de MiniRocket [3] a été utilisée comme modèle de base. Nous avons utilisé la version Python 3.11.4 pour notre implémentation.

Pour comparer les performances de SelF-Rocket et des autres méthodes, nous utilisons

- un diagramme de différence critique (DDC) [8], qui donne les rangs moyens de chaque méthode pour tous les ensembles de données, les lignes horizontales indiquent qu'il n'y a pas de différence statistique significative entre ces méthodes,
- et une matrice de comparaison multiple (MCM) [6] incluant le nombre de victoires/égalités/défaites entre chaque couple de classifieurs pour tous les jeux de données, et où les couleurs de la carte thermique représentent les différences moyennes de scores.

Toutes les expériences ont été menées sur un serveur de calcul équipé d'un Intel(R) Xeon(R) Gold 6434, avec 250 Go de RAM, sous Ubuntu 24.04.1 LTS.

Nous avons choisi comme paramètres par défaut pour SelF-Rocket, un nombre de blocs (folds) $k = 2$, un nombre de caractéristiques par mini-classifieur $f = 2500$, un nombre d'exécutions $nr = 10$, et pour la validation du vote : un top = 5 et un seuil de 0.9. Les autres paramètres (le nombre de noyaux, le remplissage par zéro (padding), la dilatation, le biais, la longueur

2. https://github.com/time-series-machine-learning/tsml-eval/tree/main/results/classification/Univariate
3. https://github.com/angus924/minirocket

Algorithme 1 : Module de sélection de caractéristiques

Entrée : Vecteur de caractéristiques d'entraînement $\boldsymbol{FV}$
Les labels des données d'entraînement $\boldsymbol{y_{train}}$
Paramètres : Nombre de blocs $\boldsymbol{k}$
Nombre $\boldsymbol{f}$ de caractéristiques par mini-classifieur
Nombre $\boldsymbol{nr}$ d'exécutions
La taille maximale $\boldsymbol{mds}$ allouée par découpage du jeu de données
Sortie : L'ensemble optimal des caractéristiques **S**

si $taille(y_{train}) \leq mds$ **alors**
  splt ← RepeatedStratifiedKFold($n\_splits = k,\ n\_repeat = nr$) ;
sinon
  splt ← StratifiedShuffleSplit($n\_splits = k \times nr,\ train\_size = int(mds/2),\ test\_size = int(mds/2)$) ;
fin
performances ← Liste() ;
pour $l \in [0, 1, \ldots, k \times nr - 1]$ **faire**
  ind_{train} ← splt[l][0] ; `// Index d'entraînement`
  ind_{val} ← splt[l][1] ; `// Index de Validation`
  pour $t \in [0, 1, \ldots, |FV| - 1]$ **faire**
    classifieur ← RidgeClassifier() ;
    `// Selection aléatoire des` f `indices de caractéristiques`
    ind_{feats} ← Random($[0, 1, \ldots, |FV[t]| - 1], f$) ;
    $feats_{train} \leftarrow FV[t][ind_{train}][:, ind_{feats}]$;
    $feats_{val} \leftarrow FV[t][ind_{val}][:, ind_{feats}]$;
    $y_{train} \leftarrow y_{train}[ind_{train}]$;
    $y_{val} \leftarrow y_{train}[ind_{val}]$;
    classifieur.entrainement($feats_{train}, y_{train}$) ;
    y_{pred} ← classifieur.prediction($feats_{val}$) ;
    performances.ajout(ScoreExactiture(y_{val}, y_{pred})) ;
  fin
fin
idx_{hvms} ← VoteParMediane(performances) ;
idx_{final} ← ValidationduVote(performances, $idx_{hvms},\ nb_{voters} = k \times nr$) ;
S ← $FV[idx_{final}]$;
retourner S

Algorithme 2 : Validation du Vote

Entrée : Vecteur de Performance $\boldsymbol{PV}$
Index de la combinaison IR-PO choisie $\boldsymbol{idx_{vote}}$
Nombre de votants $\boldsymbol{nb_{vot}}$
Paramètres : Index de la combinaison IR-PO par défaut $\boldsymbol{idx_{defaut}}$
La valeur du Top considéré $\boldsymbol{top}$
La valeur du seuil $\boldsymbol{thresh}$
Sortie : L'index final sélectionné $\boldsymbol{idx_{final}}$

compteur ← Liste() ;
`// Vérifie si la combinaison IR-PO choisie fait partie des meilleurs résultats de chaque votant.`
pour $vot \in [0, 1, \ldots, nb_{vot} - 1]$ **faire**
  compteur.ajout(IsInTopValue($PV[vot][idx_{vote}], PV[vot], top$)) ;
fin
si $moyenne(compteur) \geq thresh$ **alors**
  $idx_{final} \leftarrow idx_{vote}$;
sinon
  $idx_{final} \leftarrow idx_{defaut}$;
fin
retourner idx_{final}

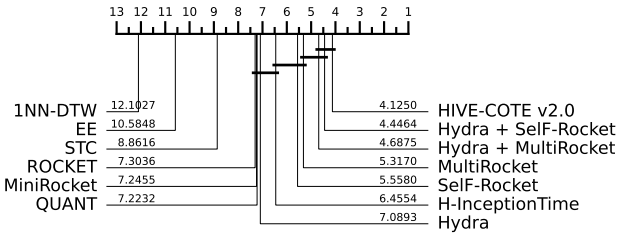


FIGURE 4 – Diagramme de différence critique de SelF-Rocket et de 11 autres classifieurs.

du noyau, les valeurs à l'intérieur du noyau) gardent les mêmes valeurs que celles fixées par défaut dans MiniRocket.

5.2 Comparaison avec les autres méthodes de classification de séries temporelles

Nous comparons les performances de SelF-Rocket et Hydra + SelF-Rocket avec 11 méthodes CST : 1NN-DTW, Elastic Ensemble (EE) [10], Shapelet transform classifier (STC) [11], HIVE-COTE v2.0 [13], H-Inceptiontime [7], MultiRocket [16], Hydra, Hydra+MultiRocket [4], MiniRocket [3], ROCKET [2] et QUANT [5].

Les performances de ces méthodes sont indiquées dans les Figures 4 et 5, sous la forme d'un DDC et d'une MCM. Pour cette dernière, nous ne retenons que les 6 classifieurs les plus précis avec SelF-Rocket. En termes de performances, Hydra + SelF-Rocket se classe juste derrière HIVE-COTE v2.0 et surpasse les autres. SelF-Rocket se classe juste après MultiRocket en termes de rang malgré une exactitude légèrement supérieure. Il n'y a pas de différence statistique significative selon le test de rang signé de Wilcoxon entre Hydra + SelF-Rocket, Hydra + MultiRocket et HIVE-COTE v2.0.

SelF-Rocket (9996 ou 19992 caractéristiques) et Hydra + SelF-Rocket (9996 ou 19992 caractéristiques + nombre de caractéristiques de Hydra) sont plus rapides dans la phase de prédiction du classifieur que MultiRocket et Hydra + MultiRocket en raison d'un nombre inférieur de caractéristiques (resp. 50000 et 50000 + nombre de caractéristiques de Hydra).

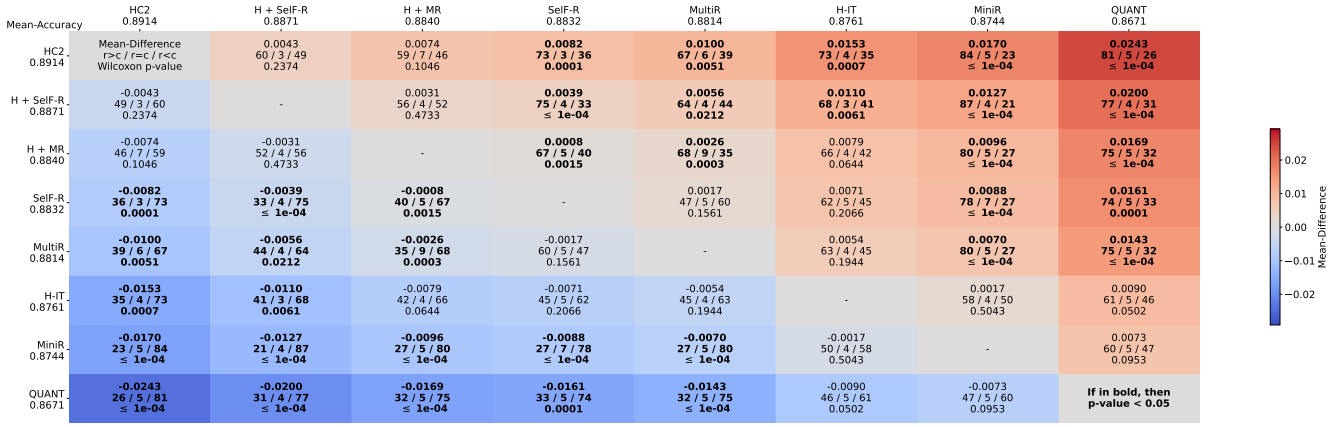


FIGURE 5 – Matrice de Comparaison Multiple (MCM) pour SelF-Rocket, Hydra + SelF-Rocket et six autres méthodes.

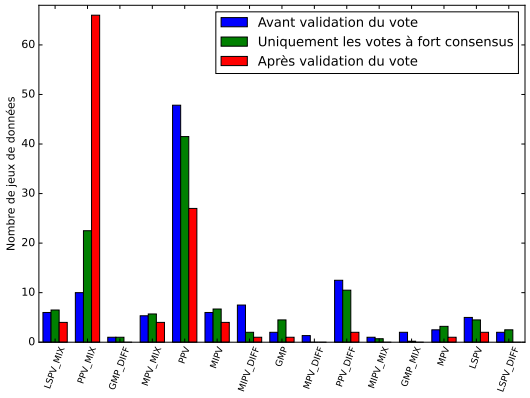


FIGURE 6 – La variation des combinaisons IR-PO les plus choisies parmi les 30 ré-échantillonnages de chacun des 112 jeux de données UCR

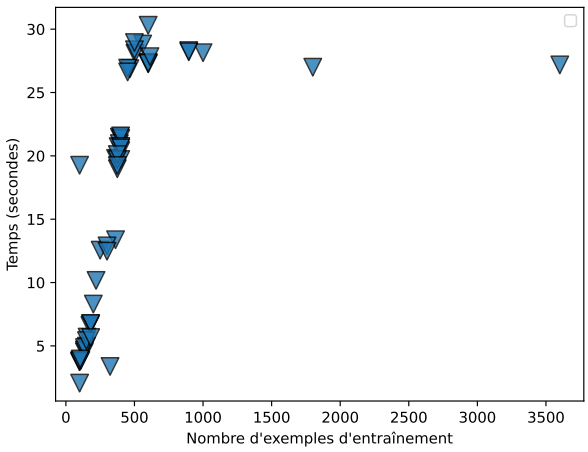
Dans la Figure 6, est donné un aperçu des combinaisons IR-PO retenues le plus souvent par SelF-Rocket avant la validation des votes, uniquement les votes à fort consensus (au moins 90% des votants doivent retrouver la combinaison choisie dans leurs Top 5), et après validation des votes. Nous pouvons remarquer qu'après la phase de validation des votes, plusieurs combinaisons IR-PO peuvent être choisies, la combinaison PPV_MIX étant la plus fréquemment choisie, cela est principalement dû à son utilisation comme combinaison par défaut lorsque le consensus est insuffisant. Avant la phase de validation des votes, plusieurs IR-PO différentes pourraient être sélectionnées, mais des tests (non donnés ici faute de place, mais présents ici [12]) montrent de plus faibles performances, d'où la nécessité de cette phase de validation des votes pour s'assurer d'un bon choix d'IR-PO. Enfin, la répartition des combinaisons IR-PO pour les votes ayant abouti à un consensus élevé des votants (les votes nécessitant un réajustement ne sont pas pris en compte ici) est la plus proche de la distribution des meilleures combinaisons IR-PO de l'Oracle, cf Figure 1.

5.3 Étude et comparaison temporelle de SelF-Rocket

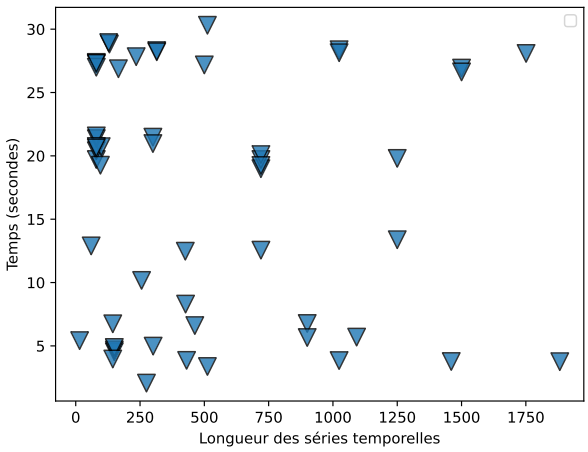
Par rapport aux autres méthodes ROCKET, SelF-Rocket inclut un module de sélection des caractéristiques. Le temps de génération des caractéristiques de SelF-Rocket et Hydra + SelF-Rocket est similaire à celui de MultiRocket et Hydra + MultiRocket respectivement, tandis que le temps d'apprentissage du classifieur Ridge de SelF-Rocket est similaire à celui de MiniRocket.

La Figure 7 montre le temps de calcul moyen pour tous les ré-échantillons et tous les ensembles de données du module de sélection des caractéristiques en fonction de la longueur de la série temporelle et du nombre d'exemples d'apprentissage. La complexité du classifieur dépend de f, le nombre de caractéristiques, et de n, le nombre d'exemples d'apprentissage. Le nombre d'exécutions nr est un multiplicateur du nombre de mini-classifieurs, et f influe sur la complexité d'un seul classifieur, de sorte que les deux paramètres ont un impact général sur le temps de calcul. Pour chaque combinaison d'IR-PO, nous instancions $k \times nr$ mini-classifieurs afin d'identifier l'ensemble optimal de caractéristiques. Un classifieur Ridge d'une complexité de $\mathcal{O}(n \cdot f^2)$ est utilisé. Nous avons une valeur de $mds = 500$ (cf. Algorithme 1), ce qui explique pourquoi il y a un plafond dans la Figure 7a sur le temps moyen du module de sélection des caractéristiques après cette valeur.

La Figure 8 illustre les temps moyens d'apprentissage et de prédiction des classifieurs considérés. Nous pouvons observer que le temps d'apprentissage supplémentaire des méthodes utilisant SelF-Rocket n'est pas très élevé (quelques secondes pour ces données), alors que leurs temps de prédiction restent rapides pour de très bonnes performances (cf Figure 4),

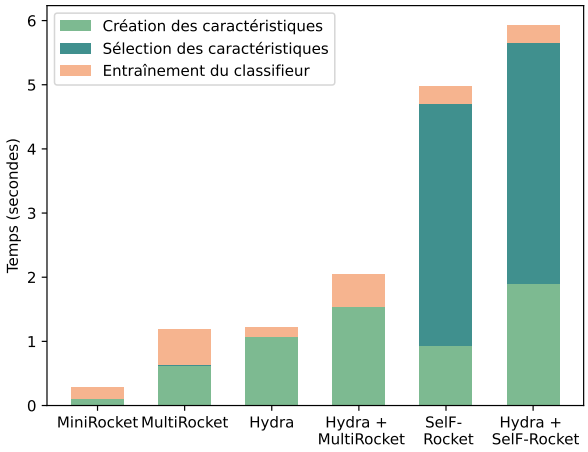


(a) Temps de calcul moyen pour tous les ré-échantillons de tous les ensembles de données pour le module de sélection des caractéristiques de SelF-Rocket ($nr = 10$, $f = 5000$) en fonction du nombre d'exemples d'entraînement

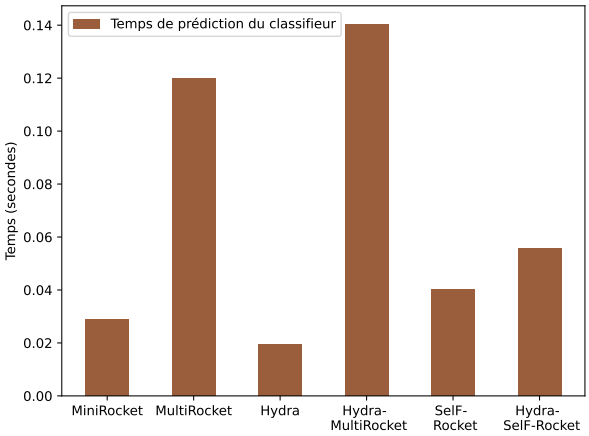


(b) Temps de calcul moyen pour tous les ré-échantillons de tous les ensembles de données pour le module de sélection des caractéristiques de SelF-Rocket ($nr = 10$, $f = 5000$) en fonction de la longueur des séries temporelles

FIGURE 7 – Temps de calcul du module de sélection des caractéristiques pour les 55 ensembles de données de développement en fonction de la longueur de la série temporelle et du nombre d'exemples d'apprentissage.



(a) Temps total d'apprentissage moyen de SelF-Rocket ($nr = 10$, $f = 2500$), MiniRocket, MultiRocket, Hydra et Hydra + MultiRocket, y compris les étapes de création des caractéristiques, de sélection des caractéristiques et d'apprentissage du classifieur Ridge.



(b) Temps de prédiction moyen de SelF-Rocket ($nr = 10$, $f = 2500$), MiniRocket, MultiRocket, Hydra et Hydra + MultiRocket

FIGURE 8 – Comparaison entre le temps moyen de prédiction du classifieur pour tous les ensembles de données et les ré-échantillons de SelF-Rocket et Hydra + SelF-Rocket avec ceux des autres méthodes Rocket.

MiniRocket restant la plus rapide en apprentissage parmi les méthodes évaluées. Plus précisément, dans la Figure 8a, nous pouvons voir le temps de calcul total moyen sur les 55 ensembles de données de développement, pour tous les ré-échantillons des méthodes ROCKET. Les temps d'entraînement (Figure 8a) et de prédiction (Figure 8b) des classifieurs SelF-Rocket et MiniRocket Ridge sont comparables. La création des caractéristiques SelF-Rocket est plus longue que celle des caractéristiques MultiRocket. Cela est dû au plus grand nombre de noyaux qui sont instanciés. MultiRocket n'instancie que 6216 noyaux pour chacune de ses représentations d'entrée (DIFF et BASE). Parmi ceux-ci, 49728 caractéristiques (6216 noyaux $\times$ 2 représentations d'entrée$\times$4 opérateurs d'agrégation) sont générées. En comparaison, SelF-Rocket génère 9996 noyaux pour chacun de ces deux types de représentation, soit un total de 99960 caractéristiques (9996 noyaux $\times$ 2 représentations d'entrée $\times$ 5 opérateurs d'agrégation). SelF-Rocket produit 9 fois plus de caractéristiques que MiniRocket. Une façon d'améliorer son temps de calcul global pourrait être de réduire le nombre de noyaux générés par SelF-Rocket ou de réduire le nombre de représentations d'entrée et d'opérateurs d'agrégation. Hydra produit $k_g \times g \times d$ noyaux : k_g (nombre de noyaux par groupe), g (nombre de groupes) et d (valeur maximale possible de la dilatation), avec 2 caractéristiques extraites par noyau (réponses maximales et minimales). Pour chaque ensemble de données, 512 ($k_g = 8$ et $g = 64$) noyaux par dilatation d sont créés avec $2^d \leq$ longueur de la série temporelle. Dans la plupart de nos cas, d sera compris entre 7 et 10, de sorte que le nombre total de caractéristiques générées sera généralement compris entre $512 \times 2 \times 7$ (7128) et $512 \times 2 \times 10$ (10240). Cela explique pourquoi Hydra est le plus rapide en termes de temps d'entraînement et de prédiction du classifieur (sans compter les parties création et sélection des caractéristiques, cf Figure 8).

5.4 SelF-Rocket avec un oracle

La sélection de la meilleure combinaison IR-PO peut s'avérer difficile, en particulier pour les petits ensembles de données. Par conséquent, la mise en œuvre de SelF-Rocket présentée dans cet article n'est pas optimale. En connaissant la meilleure combinaison IR-PO, quelles seraient les performances de SelF-Rocket ? La Figure 9 représente le rang moyen d'exactitude pour les différents classifieurs testés précédemment si nous pouvions réussir à sélectionner pour chaque ensemble de données la meilleure combinaison IR-PO. Ces résultats illustrent le potentiel de cette méthode si nous pouvions améliorer cette sélection. Les résultats détaillés de la version oracle de SelF-Rocket [4] et Hydra + SelF-Rocket [5] sont disponibles sur Github.

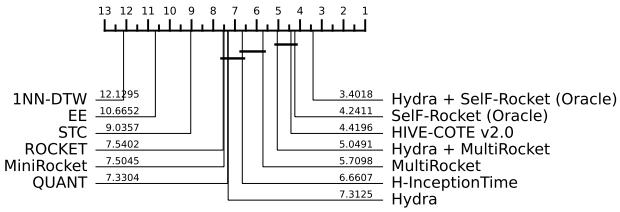


FIGURE 9 – Diagramme de rang moyen entre SelF-Rocket (Oracle), Hydra + SelF-Rocket (Oracle) et d'autres méthodes de l'état de l'art.

6 Conclusion et perspectives

Dans cet article, nous avons d'abord montré que le choix des représentations d'entrée et des opérateurs d'agrégation a un impact significatif sur les performances des méthodes de CST à noyaux de convolution aléatoires, puis que l'utilisation de plusieurs opérateurs d'agrégation, tels que MultiRocket, conduit à de bonnes performances en moyenne, mais n'est pas optimale dans des cas spécifiques. Il est toujours préférable de sélectionner un seul opérateur d'agrégation approprié.

Sur cette base, nous avons proposé un nouvel algorithme basé sur MiniRocket, SelF-Rocket, qui incorpore une étape de sélection de caractéristiques. Les expériences montrent que, dans de nombreux cas, cet algorithme sélectionne l'une des meilleures combinaisons de représentation d'entrée et d'opérateur d'agrégation, ce qui permet d'obtenir de très bonnes performances. Dans l'implémentation testée de SelF-Rocket, seules 9996 ou 19992 caractéristiques sont retenues pour entraîner le classifieur Ridge, soit beaucoup moins que les méthodes concurrentes telles que MultiRocket ou Hydra + MultiRocket, ce qui permet d'obtenir des prédictions rapides.

Dans cet article, une version minimale de SelF-Rocket a été évaluée, en utilisant 3 ensembles de représentations d'entrée, 5 opérateurs d'agrégation et une méthode simple de sélection des caractéristiques. Cette implémentation pourrait être améliorée en considérant d'autres représentations d'entrée (différence de second ordre, transformée de Hilbert, ...), et en sélectionnant automatiquement le nr correct et le seuil de vote. Au final, en échange d'un temps d'entraînement un peu plus long, SelF-Rocket permet d'obtenir des prédictions au moins du même niveau que MultiRocket, et ce plus rapidement et en utilisant $2,5$ à 5 fois moins de caractéristiques (9996 caractéristiques ou 19992 pour SelF-Rocket contre 49728 pour MultiRocket).

Par ailleurs, utiliser une méthode de sélection des caractéristiques basée sur un filtre (test χ^2 par exemple)

4. https://github.com/ANR-MYEL/SelF-Rocket/blob/main/results/Mean_perf_SR_UCR112_Oracle.csv

5. https://github.com/ANR-MYEL/SelF-Rocket/blob/main/results/Mean_perf_HSR_UCR112_Oracle.csv

au lieu du k-Fold stratifié permettrait d'accélérer l'étape de sélection des caractéristiques.

Remerciements

Ce travail de recherche, soutenu et financé par l'ANR (Agence Nationale pour la Recherche), entre dans le cadre du Labcom (Laboratoire Commun) MYEL (MobilitY and Reliability of Electrical chain Lab) associant le LSEE, le LGI2A et la société CRITTM2A (Projet ANR- 22-LCV2-0001).

Références

[1] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. The UCR Time Series Archive, 2018.

[2] Angus Dempster, François Petitjean, and Geoffrey I Webb. ROCKET : exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5) :1454–1495, 2020.

[3] Angus Dempster, Daniel F Schmidt, and Geoffrey I Webb. MiniRocket : A very fast (almost) deterministic transform for time series classification. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pages 248–257, 2021.

[4] Angus Dempster, Daniel F Schmidt, and Geoffrey I Webb. Hydra : Competing convolutional kernels for fast and accurate time series classification. *Data Mining and Knowledge Discovery*, 37(5) :1779–1805, 2023.

[5] Angus Dempster, Daniel F Schmidt, and Geoffrey I Webb. Quant : A minimalist interval method for time series classification. *Data Mining and Knowledge Discovery*, 38(4) :1–26, 2024.

[6] Ali Ismail-Fawaz, Angus Dempster, Chang Wei Tan, Matthieu Herrmann, Lynn Miller, Daniel F. Schmidt, Stefano Berretti, Jonathan Weber, Maxime Devanne, Germain Forestier, and Geoffrey I. Webb. An approach to multiple comparison benchmark evaluations that is stable under manipulation of the comparate set, 2023.

[7] Ali Ismail-Fawaz, Maxime Devanne, Jonathan Weber, and Germain Forestier. Deep learning for time series classification using new hand-crafted convolution filters. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 972–981. IEEE, 2022.

[8] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification : a review. *Data mining and knowledge discovery*, 33(4) :917–963, 2019.

[9] Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F Schmidt, Jonathan Weber, Geoffrey I Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. Inceptiontime : Finding alexnet for time series classification. *Data Mining and Knowledge Discovery*, 34(6) :1936–1962, 2020.

[10] Jason Lines and Anthony Bagnall. Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, 29 :565–592, 2015.

[11] Jason Lines, Luke M Davis, Jon Hills, and Anthony Bagnall. A shapelet transform for time series classification. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 289–297, 2012.

[12] Mouhamadou Mansour Lo, Gildas Morvan, Mathieu Rossi, Fabrice Morganti, and David Mercier. Time series classification with random convolution kernels based transforms : pooling operators and input representations matter. *arXiv preprint arXiv :2409.01115*, 2024.

[13] Matthew Middlehurst, James Large, Michael Flynn, Jason Lines, Aaron Bostrom, and Anthony Bagnall. HIVE-COTE 2.0 : a new meta ensemble for time series classification. *Machine Learning*, 110(11) :3211–3243, 2021.

[14] Matthew Middlehurst, Patrick Schäfer, and Anthony Bagnall. Bake off redux : a review and experimental evaluation of recent time series classification algorithms. *Data Mining and Knowledge Discovery*, 38(4) :1958–2031, 2024.

[15] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, 26(1) :43–49, 1978.

[16] Chang Wei Tan, Angus Dempster, Christoph Bergmeir, and Geoffrey I Webb. MultiRocket : multiple pooling operators and transformations for fast and effective time series classification. *Data Mining and Knowledge Discovery*, 36(5) :1623–1646, 2022.