Apprentissage en Profondeur pour la Classification des Séries Temporelles à l'aide de Nouveaux Filtres de Convolution Créés Manuellement

Ali Ismail-Fawaz, Maxime Devanne, Jonathan Weber, Germain Forestier

Apprentissage en Profondeur pour la Classification des Séries Temporelles à l'aide de Nouveaux Filtres de Convolution Créés Manuellement

Ali Ismail-Fawaz Maxime Devanne Jonathan Weber Germain Forestier

IRIMAS, Universite Haute-Alsace, Mulhouse, France

{ali-el-hadi.ismail-fawaz,maxime-devanne,jonathan.weber,germain.forestier}@uha.fr

Résumé

Ces dernières années, l'apprentissage profond a suscité un intérêt croissant pour la classification de séries temporelles. Dans ce domaine, les architectures les plus performantes s'appuient sur des réseaux de neurones convolutifs. Ces derniers permettent l'apprentissage de filtres de convolution unidimensionnels afin de capturer des motifs permettant de discriminer différentes classes. Ces filtres sont initialisés de manière aléatoire puis optimisés tout au long de l'entraînement du modèle. Dans cet article, nous explorons des filtres créés manuellement dans le but de capturer des motifs spécifiques aux séries temporelles (pente ascendante, pente descendante, pics). Pour cela, nous proposons un ensemble de filtres dont les valeurs sont fixes et non modifiées lors de l'étape d'apprentissage. La pertinence de l'ajout de ces filtres est étudiée en les utilisant au sein d'architectures existantes. Pour cela nous les avons testés avec plusieurs architectures, allant de la plus simple (Fully Convolutional Network (FCN)), à la plus performante de l'état de l'art (InceptionTime). Les expériences révèlent que l'ajout de nos filtres, créés manuellement, augmente la précision des prédictions sur une majorité des 128 jeux de données de l'archive UCR. Nous montrons également que l'utilisation conjointe de nos filtres créés manuellement avec des filtres appris, permet d'obtenir les meilleurs modèles. Ce travail est la première étape afin de proposer un catalogue de filtres génériques et fixes qui pourraient être utiles pour une large gamme d'applications pour améliorer la précision des modèles profonds pour la classification de séries temporelles.

Mots Clef

Classification des Séries Temporelles, Réseaux de Neurones, Apprentissage Profond, Filtres de Convolution Créés manuellement

1 Introduction

Ces dernières années, la classification des séries temporelles (*Time Series Classification* TSC) a connu un intérêt croissant dans la communauté scientifique, surtout après la publication de l'archive UCR [5], la plus grande archive d'ensembles de données TSC univariées. La TSC a

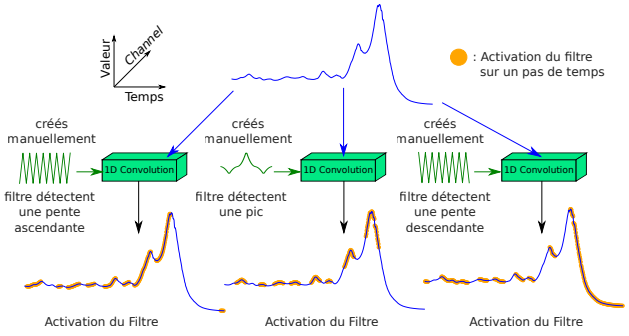

FIGURE 1 – Trois filtres créés manuellement détectant : (1) les pentes ascendantes, (2) les pentes ascendantes et (3) les pics d'une série temporelle. Les points orange indiquent sur quels pas de temps les filtres sont activés après avoir été convolués avec une série temporelle d'entrée de l'ensemble de données Meat de l'archive UCR.

également été utilisée pour la prédiction des compétences chirurgicales [16], l'évaluation de la réadaptation physique [9] et la télédétection [26]. Dans des travaux récents, des approches d'apprentissage profond ont été utilisées pour l'analyse des séries temporelles. Ces approches comprennent la classification [30, 18, 10], le clustering [21, 28], la distillation des connaissances [1], les attaques adversariales [25, 11], l'augmentation des données [15, 24], le *self-supervised* [14], etc. Dans presque tous les cas, il est démontré que les réseaux de neurones à convolution (Convolutional Neural NEtworks CNNs) peuvent être plus performants que d'autres approches pour capturer des caractéristiques pertinentes sur des séries temporelles. L'extraction de caractéristiques consiste à trouver des combinaisons linéaires entre des intervalles consécutifs d'une taille fixe. Plus le modèle est profond, plus il augmente son champ réceptif. Ce dernier représente l'espace d'entrée dont dépend un point situé à une certaine profondeur du réseau. Plus le champ réceptif est grand, plus il est bénéfique pour le modèle.

En vision par ordinateur, de nombreux travaux ont été proposés pour expliquer les caractéristiques détectées par les CNN sur les images. Il a été montré que la plupart du

temps, les premières couches de convolution permettent de détecter les contours d'une image. Les couches plus profondes sont alors capables de détecter des caractéristiques plus complexes telles que les changements de texture, d'arrière-plan et d'orientation. Cela a motivé les chercheurs à créer des filtres de convolution manuellement, au lieu de les apprendre, afin de détecter ces caractéristiques [4, 12, 13].

Dans ce travail, nous abordons le problème du TSC et proposons des filtres créés manuellement dédiés aux séries temporelles, ce qui, à notre connaissance, n'a jamais été fait auparavant. Nous proposons trois types de filtres, les filtres de détection de pente ascendante et descendante et le filtre de détection de pic. Un résumé des filtres créés manuellement proposés dans cet article peut être vu dans la Figure 1. Étant donné que dans le domaine des séries temporelles, nous travaillons dans un axe unidimensionnel les premiers types de modèles qui peuvent être utiles à la classification des séries temporelles sont les pentes ascendantes et descendantes. En outre, certains types d'ensembles de données présentent de nombreuses oscillations (hausses et baisses) à une fréquence élevée. Cela pourrait perturber la classification si un modèle ne prend en compte que les pentes ascendantes et descendantes. C'est pourquoi nous considérons également un autre type de motif qui peut être détecté, les pics.

Les modèles d'apprentissage profond utilisent la rétropropagation afin de minimiser une fonction de perte sur une entrée donnée. L'objectif de ce dernier algorithme est la propagation des erreurs de couche en couche dans le réseau de neurones. En d'autres termes, si une erreur se produit dans la dernière couche du réseau, l'erreur qu'elle génère est propagée à la première couche. Le modèle est entraîne sur un seul jeu de données, par conséquent, le modèle peut avoir du mal à apprendre un filtre général qui satisfait tous les jeux de données. Pour résoudre ce problème, une approche triviale consisterait à ajouter davantage de filtres à apprendre. Cela pourrait conduire à d'excellentes performances, mais aussi encourager le sur-apprentissage dans certains cas. Un problème supplémentaire avec cette solution triviale est son coût en mémoire et sa complexité en temps. Pour surmonter ces problèmes, d'autres approches ont été employées comme l'application d'une régularisation aux filtres ou l'utilisation d'un dropout pour mieux généraliser et être plus robuste. Dans ce travail, nous proposons une approche originale en utilisant des filtres créés manuellement, en parallèle des filtres appris par le modèle d'apprentissage profond. De cette façon, le réseau se concentre davantage sur les types de filtres qui n'ont pas été créés manuellement.

Pour évaluer l'impact des filtres créés manuellement sur les performances de TSC, nous proposons deux modèles adaptés d'architectures précédentes, un Réseau Entièrement Convolutif (Fully Convolutional Network FCN) [30] et InceptionTime [18]. Nous évaluons ainsi deux modèles hybrides et montrons qu'avec l'aide des filtres génériques créés manuellement, ces modèles peuvent se concentrer sur d'autres types de filtres et ne pas recalculer les mêmes filtres (ceux créés manuellement). Nous tenons à souligner que les filtres proposés dans cet article sont indépendants du jeu de données considéré. Ils agissent donc comme des filtres génériques capables d'extraire des caractéristiques similaires pour toute série temporelle d'entrée. Cela n'est pas vrai pour les filtres appris par un réseau de neurones convolutifs (CNN) étant donné que les filtres varient d'un jeu de données à l'autre en raison de la phase d'apprentissage.

Nos principales contributions dans ce travail sont :

- La proposition de nouveaux filtres créés manuellement pour détecter les pics et les pentes ascendantes et descendantes.
- La construction de deux modèles hybrides d'architectures existantes en y incorporant des filtres créés manuellement.
- L'évaluation de ces modèles hybrides sur l'archive UCR démontrant l'impact important des filtres créés manuellement sur les performances.

2 Travaux Connexes

La TSC vise à associer une série temporelle donnée à une étiquette de classe correspondante en utilisant un classifieur. Ce classifieur peut être par exemple une régression linéaire, un arbre de décision, l'algorithme des k-plus proches voisins ou un réseau de neurones profond. Les paragraphes suivants passent en revue certaines des approches existantes les plus pertinentes en matière de TSC.

2.1 Sur les données brutes

Une première série de méthodes de TSC a travaillé sur la définition d'une métrique calculant la similarité entre deux séries temporelles brutes. Ainsi, la métrique Déformation Temporelle Dynamique (Dynamic Time Warping DTW) a été largement utilisée car elle permet de comparer deux séries temporelles brutes indépendamment de leurs distorsions temporelles. Par exemple, dans l'article [2], les auteurs ont utilisé la métrique DTW avec un classifieur du Plus Proche Voisin (Nearest Neighbour NN-DTW). Les auteurs de [32] ont ensuite proposé une adaptation de DTW, shapeDTW, qui évite l'alignement point par point entre deux séries temporelles. La métrique obtenue, lorsqu'elle est utilisée avec l'algorithme du plus proche voisin (NN-ShapeDTW), a permis d'améliorer de manière significative les performances de TSC, établissant ainsi une nouvelle référence sur l'archive UCR. De plus, les auteurs de [23] ont introduit une façon de calculer la moyenne de séries temporelles en utilisant la métrique DTW. Cet algorithme aligne point par point deux séries temporelles et fait la moyenne sur les pas de temps alignés. Cette méthode a été évaluée dans un problème de TSC en utilisant NN-DTW.

2.2 Réseaux de neurones profonds

L'apprentissage profond s'est révélé performant sur les images depuis [20]. Plusieurs adaptations ont été proposées pour les séries temporelles [33, 30, 22]. Un Réseau Entièrement Convolutif (*Fully Convolutional Network* FCN) a été proposé par [30], composé de trois couches de convolution suivis d'une normalisation par lot et d'une activation ReLU. Les auteurs ont également adapté le réseau incluant des connections résiduelles (ResNet) pour les séries temporelles. Les auteurs ont utilisé trois blocs résiduels, chacun composé d'un FCN avec un nombre différent de filtres. Plus récemment, dans [18], les auteurs ont utilisé l'idée de connexion résiduelle mais au lieu de blocs de convolution réguliers, ils ont proposé plusieurs couches de convolution en parallèle afin de capturer des motifs à différentes échelles en utilisant différentes tailles de filtres. Cette architecture plus complexe, appelée InceptionTime, est actuellement l'approche basée sur l'apprentissage profond la plus performante pour la TSC. Par ailleurs, alors que des architectures d'apprentissage profond de plus en plus complexes ont été construites pour la TSC, d'autres travaux se sont concentrés sur l'allègement de cette complexité croissante. Par exemple, dans [1], la distillation des connaissances a été utilisée afin de réduire la taille du modèle FCN sans trop diminuer les performances de TSC.

2.3 Convolutions

Comme expliqué précédemment, les approches d'apprentissage profond les plus efficaces sont basées sur les convolutions. Ceci est souligné dans [17] où plusieurs approches d'apprentissage profond sont comparées. Cette revue montre que l'utilisation des convolutions permet de capturer des caractéristiques pertinentes sur des séries temporelles. Au lieu d'apprendre ces convolutions, les auteurs [6] ont présenté ROCKET, une nouvelle approche utilisant des filtres de convolution aléatoires. Certaines variations de ROCKET ont été introduites dans d'autres travaux tels que MINIROCKET [7] et MultiRocket [27].
Le succès de ces méthodes nous a incité à explorer l'utilisation de filtres de convolution créés manuellement pour la TSC. Ces filtres créés manuellement ont été largement utilisés dans le traitement d'images, comme les filtres de *Sobel* [4, 12] pour la détection des contours. En particulier, ces filtres de *Sobel* ont été utilisés avec succès pour améliorer la classification d'images dans [13]. Dans ce travail, nous avons cherché à construire des filtres de convolution manuels dédiés aux séries temporelles et à explorer leur impact sur la TSC lorsqu'ils sont combinés avec des filtres de convolutions appris à l'aide de la rétro-propagation.

3 Méthode

3.1 Définitions

Avant de décrire la méthode que nous proposons, nous introduisons des définitions importantes qui seront utilisées dans le reste de cet article.

Séries temporelles univariées Soit $\mathbf{x}$ une série temporelle univariée de longueur L, une séquence de points de données également séparés dans le temps.

Jeu de données de séries temporelles univariées Un jeu de données $\mathcal{D} = \{(\mathbf{x}_0, y_0), ..., (\mathbf{x}_N, y_N)\}$ est un ensemble de N séries temporelles univariées de longueur L associées à une étiquette y.

Convolution unidimensionnelle Opération utilisant un filtre $\mathbf{w}$ de longueur k sur une série temporelle $\mathbf{x}$ pour obtenir $\mathbf{s} = \mathbf{x} * \mathbf{w}$ comme suit :

$$\forall t \in [0; L-1] \quad \mathbf{s}[t] = \sum_{i=0}^{k-1} \mathbf{x}[t+i].\mathbf{w}[i] \tag{1}$$

Activation du filtre Lorsque l'opération de convolution donne lieu à une réponse positive, le filtre est considéré comme activé. Dans cet article, une telle activation est représentée par un point orange dans les figures suivantes.

Pente ascendante Une sous-séquence d'une série temporelle $\mathbf{x}$ dont les valeurs sont strictement croissantes dans le temps.

Pente descendante Une sous-séquence d'une série temporelle $\mathbf{x}$ où les valeurs sont strictement décroissantes dans le temps.

Tendance stationnaire Une sous-séquence d'une série temporelle $\mathbf{x}$ où la variations des valeurs est inférieure à ϵ.

Pic Une sous-séquence d'une série temporelle $\mathbf{x}$ où les valeurs évoluent avec une grande variation de manière croissante puis décroissante.

3.2 Filtres créés manuellement pour les séries temporelles

Dans cet article, nous proposons des filtres créés manuellement adaptés aux séries temporelles afin de détecter des modèles spécifiques. Ces filtres sont décrits dans les paragraphes suivants.

Filtre de détection des pentes ascendantes. Pour qu'un filtre puisse détecter une pente ascendante, ce dernier doit détecter la différence de valeurs entre les pas de temps. Par conséquent, nous définissons un filtre de pente ascendante de longueur k comme suit : $\mathbf{w_{I_k}} = [(-1)^{(i+1)}$ pour $i \in \{0, ..., k-1\}$. Pour s'assurer qu'aucun pas de temps n'est laissé intact lors de l'application de la convolution, k doit être un nombre pair. Par exemple, le filtre de détection de tendance croissante de longueur 12 est défini comme suit :

$$\mathbf{w_{I_{12}}} = [-1, 1, -1, 1, -1, 1, -1, 1, -1, 1, -1, 1] \tag{2}$$

L'opération de convolution résultante avec un filtre de détection de pente ascendante de taille $k = 16$ est visible sur la Figure 2. Nous pouvons observer que les points d'activation en orange sont principalement localisés dans les parties croissantes de la série temporelle.

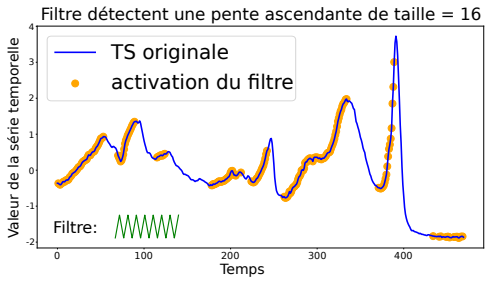

FIGURE 2 – Filtre créé manuellement de détection de pente ascendante de longueur $k = 16$ appliqué sur le jeu de données Beef de l'archive UCR. Le pas de temps où le filtre est activé (en orange) est uniquement sur les intervalles croissants de la série temporelle (en bleu).

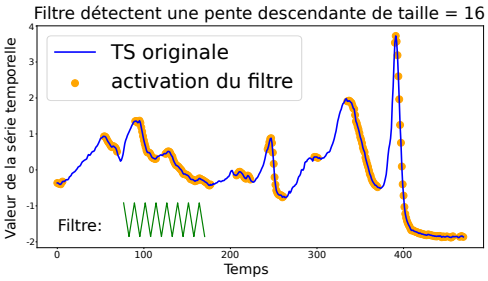

FIGURE 3 – Filtre créé manuellement de détection de pente descendante de longueur $k = 16$ appliqué sur le jeu de données Beef de l'archive UCR. Le pas de temps où le filtre est activé (en orange) est uniquement sur les intervalles décroissants de la série temporelle (en bleu).

Filtre de détection des pentes descendantes. De la même manière que le détecteur de pente ascendante, le filtre de détection de pente descendante a pour but de détecter la différence de valeurs entre les pas de temps. Cependant, il doit être activé lorsqu'il est appliqué sur un intervalle décroissant. Par conséquent, nous définissons le filtre de détection de pente descendante de longueur k comme suit : $\mathbf{w_{D_k}} = [(-1)^i$ pour $i \in \{0, ..., k-1\}]$. Pour la même raison que le filtre de détection de pente ascendante, k doit être un nombre pair. Par exemple, le filtre de détection de pente descendante de longueur 12 est défini comme suit :

$$\mathbf{w_{D_{12}}} = [1, -1, 1, -1, -1, 1, -1, -1, 1, -1, 1, -1] \quad (3)$$

La figure 3 montre l'application du filtre de détection de pente descendante de taille $k = 16$ sur une série temporelle. Ce filtre est principalement appliqué sur les parties décroissantes de la série temporelle, comme le montrent les points orange.

Filtre de détection des pics. Nous considérons un pic comme une pente ascendante suivie d'une pente descendante avec une grande variation entre les deux. Pour capturer un tel pic dans une série temporelle, il faut détecter un changement de convexité. Pour ce faire, nous proposons d'imiter la forme de la dérivée seconde négative de la fonction gaussienne illustrée dans la Figure 4. Une telle forme peut être définie en utilisant la fonction parabolique au carré $f(x) = x^2$.

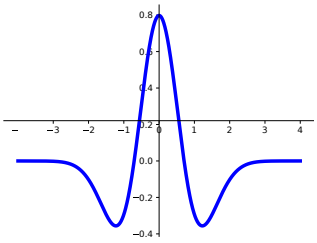

FIGURE 4 – Dérivée seconde inversée gaussienne. Cette fonction est imitée à l'aide d'un polynôme du second ordre. Par cette approche, nous avons créé manuellement le filtre de détection de pic. Nous avons retenu l'utilisation du filtre gaussien pour éviter d'avoir des hyperparamètres correspondant à la moyenne et la variance à utiliser.

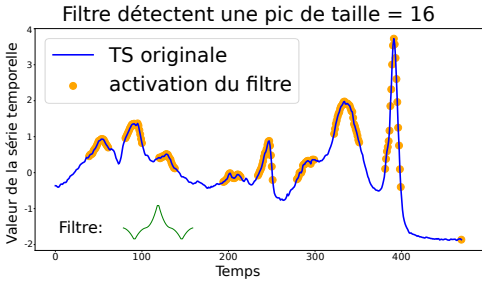

FIGURE 5 – Filtre créé manuellement de détection de pic de longueur $k = 48$ appliqué sur le jeu de données Beef de l'archive UCR. Le pas de temps où le filtre est activé (en orange) est uniquement sur les pics de la série temporelle (en bleu).

Comme notre objectif est de créer des filtres fixes et génériques qui peuvent être utilisés pour n'importe quel jeu de données, nous n'utilisons pas directement le filtre gaussien afin d'éviter le caractère aléatoire du choix de la moyenne et de la variance dans les filtres. Par conséquent, nous proposons de créer ce type de filtre en le divisant en trois parties de longueur égale comme suit : (1) une partie parabolique négative détectant la première pente ascendante, (2) une partie parabolique positive détectant le pic, (3) une partie parabolique négative détectant la pente descendante. Par exemple, le filtre de détection de pic de longueur 12 est défini comme suit :

$$\mathbf{w_{P_{12}}} = [-0.25, -1, -1, -0.25, 0.5, 2, \\ 2, 0.5, -0.25, -1, -1, -0.25] \quad (4)$$

L'opération de convolution utilisant le filtre de détection de pic de taille $k = 48$ est illustrée dans la Figure 5.

Afin de capturer les motifs de longueur variable, nous utilisons dans cet article un ensemble de n variations des filtres de détection des pentes ascendantes, descendantes et de pic créés manuellement. Une fois que les filtres créés

manuellement sont définis, notre objectif est de les utiliser comme extracteurs de caractéristiques génériques dans les architectures d'apprentissage profond pour la TSC. Dans ce travail, nous considérons deux architectures différentes : le Réseau Entièrement Convolutif (FCN) et InceptionTime. Ces architectures et leur adaptation incorporant des filtres créés manuellement sont décrites dans les deux sous-sections suivantes.

3.3 FCN Hybride (H-FCN)

Bien que les filtres créés manuellement permettent de détecter trois types de motifs différents, nous pensons que le modèle peut trouver d'autres motifs pertinents. C'est pourquoi nous avons conçu le modèle **Hybrid FCN** (H-FCN), dont un exemple peut être vu dans la Figure 6. Au lieu de remplacer l'intégralité de la première couche de convolution par des filtres créés manuellement, nous proposons de l'améliorer. Pour ce faire, les caractéristiques extraites à l'aide de nos filtres créés manuellement sont concaténées aux caractéristiques extraites par la première couche de convolution entraînable du FCN original. Le FCN original proposé dans [30] est composé de trois couches de convolution avec 128, 256 et 128 filtres respectivement. Dans ce travail, nous pensons qu'avec l'aide de nos filtres créés manuellement, il est suffisant de limiter le nombre de filtres entraînables à 64, 128 et 64, respectivement. Nous notons que, comme nos filtres de convolution créés manuellement n'incluent pas de biais, nous supprimons également le biais des couches de convolution entraînables.

3.4 L'adaptation de Inception

Inception Hybride (H-Inception). Nous considérons maintenant l'architecture plus complexe Inception [18] car c'est actuellement le modèle d'apprentissage profond le plus performant pour la TSC sur l'archive UCR. Afin d'incorporer nos filtres créés manuellement dans Inception, nous proposons le modèle Inception hybride (H-Inception). Comme pour H-FCN, nous concaténons les caractéristiques capturées à l'aide de nos filtres créés manuellement avec celles capturées par le premier bloc d'Inception. Le reste de l'architecture reste identique à l'original. Nous notons que contrairement à H-FCN, nous ne réduisons pas le nombre de filtres entraînables par couche de convolution car il est déjà faible dans le modèle Inception original.

Hybrid InceptionTime. Dans [18], un ensemble de cinq modèles d'Inception est également proposé. Nous adaptons la même idée avec le modèle **H-InceptionTime** qui est un ensemble de cinq modèles de H-Inception différents. L'architecture détaillée d'un modèle H-Inception est présentée dans la Figure 7.

4 Évaluation expérimentale

4.1 Ensembles de données et détails de mise en œuvre

Pour évaluer les modèles proposés, nous utilisons l'archive UCR 2018 [5], composée de 128 jeux de données de séries temporelles univariées étiquetées. Pour une comparaison équitable avec les approches existantes, une *z-normalization* est appliquée à chaque ensemble de données. Nous avons entraîné chaque modèle à l'aide de l'optimiseur Adam [19] avec un taux d'apprentissage décroissant. Le meilleur modèle obtenant la meilleure perte sur le jeu d'entraînement est conservé pour être évalué sur l'ensemble de test. Afin d'être moins dépendant de l'initialisation aléatoire, l'ensemble du processus est répété cinq fois. Par conséquent, les résultats présentés dans le reste de l'article sont une moyenne sur cinq initialisations différentes. Toutes les expériences ont été réalisées sur une NVIDIA GeForce GTX 1080 avec 8 GO de mémoire. Le code est disponible ici : `https://github.com/MSD-IRIMAS/CF-4-TSC`.

4.2 Résultats sur les architectures adaptées

Nous avons comparé les performances de nos architectures adaptées utilisant des filtres créés manuellement avec les modèles originaux. Pour une paire de modèles, nous avons comparé la précision obtenue sur chaque jeu de données et calculé le nombre de victoires, d'égalités et de pertes. Ces résultats comparatifs sont rapportés à l'aide de graphiques One-VS-One montrant le nombre de victoire/égalité/perte, comme le montrent les figures 8 et 9. Chaque point dans les graphiques représente un seul jeu de données de l'archive UCR. Les axes montrent la précision de classification de chaque classifieur (moyenne sur cinq initialisations) entre 0 et 1.0. De plus, afin d'évaluer la relevance de la comparaison, le test statistique des rangs signés de Wilcoxon [31, 8, 3] est effectué pour chaque paire de classifieurs. La mesure statistique résultante, la P-Valeur, est indiquée dans la légende de chaque graphique. Si la P-Valeur entre deux classifieurs est inférieure à un seuil, cela signifie que nous n'avons pas assez de jeux de données pour trouver une différence statistiquement significative. Habituellement, le seuil de cette P-Valeur pour ce type de décision est de 0.05.

H-FCN. Nous avons utilisé, pour l'entraînement de notre modèle H-FCN, la même taille de lot et le même nombre d'époques que pour le modèle FCN original, de sorte que la comparaison entre les deux modèles soit le plus juste possible. Pour les filtres de détection des pentes ascendantes et descendantes, nous avons utilisé 6 variations de longueurs $[2^i$ pour $i \in \{1, ..., 6\}]$. Pour les filtres de détection des pics, nous avons utilisé 5 variations de longueurs $[6, 12, 24, 48, 96]$.
Dans la Figure 8, nous pouvons voir que le modèle H-FCN obtient des performances significativement meilleures que le modèle FCN original (en haut à gauche) mais aussi que

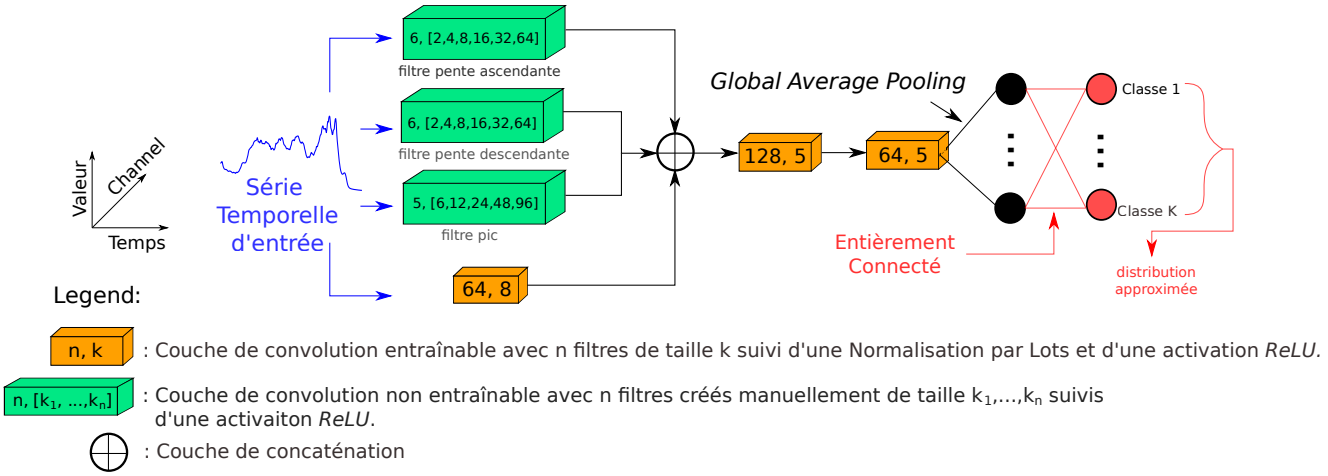

FIGURE 6 – Un exemple de l'architecture FCN Hybride (H-FCN). La série temporelle d'entrée est envoyée à un bloc de convolution (en orange, en bas à gauche) et aux filtres créés manuellement (en vert). Les caractéristiques sont ensuite concaténées et transmises au reste de l'architecture, comprenant deux couches de convolution en série (en orange) chacune suivie d'une normalisation par lots et d'une activation ReLU. Une couche de *Golbal Average Pooling* est ensuite appliquée sur la troisième couche de convolution (connexions noires) suivie d'une couche de classification entièrement connectée (connexions rouges).

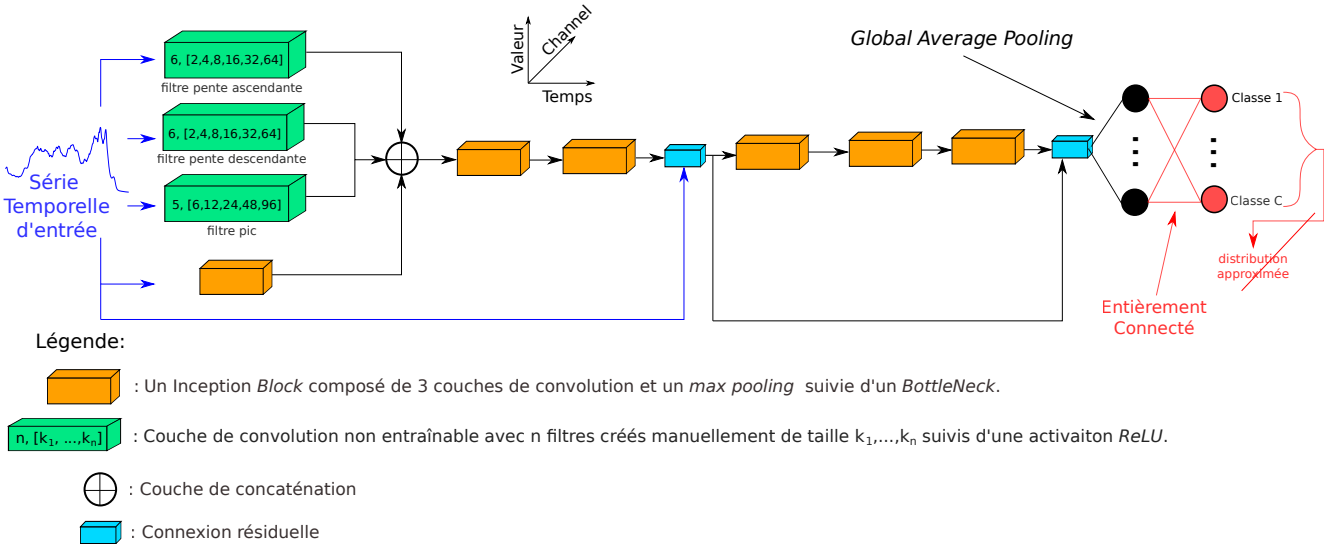

FIGURE 7 – Un exemple de l'architecture Inception Hybride (H-Inception). La série temporelle d'entrée est transmise aux filtres créés manuellement non appris et à un bloc Inception. Les caractéristiques sont ensuite concaténées et transmises au reste de l'architecture composée de 5 blocs Inception avec une connexion résiduelle comme dans le modèle d'origine. Une couche *Global Average Pooling* est ensuite appliquée sur le deuxième bloc résiduel (connexions noires) suivie d'une couche de classification entièrement connectée (connexions rouges).

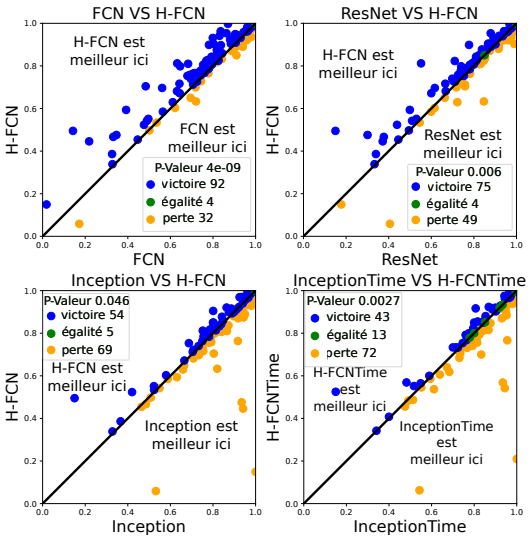

FIGURE 8 – Graphique One-VS-One montrant le nombre de victoire/égalité/perte et la P-Valeur lors de la comparaison du H-FCN proposé avec les modèles originaux FCN, ResNet, Inception et InceptionTime sur les 128 jeux de données de l'archive UCR.

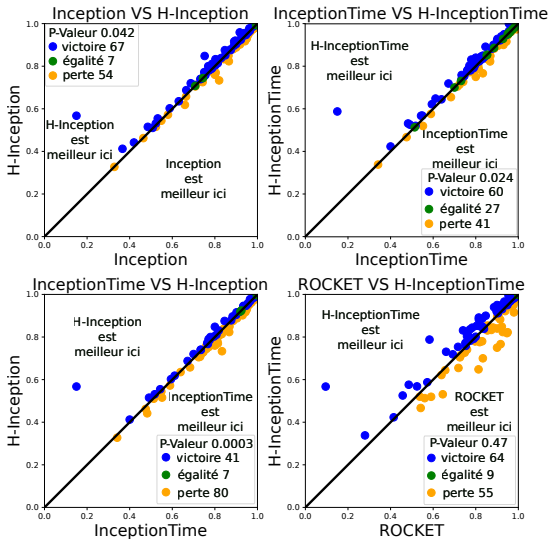

FIGURE 9 – Graphique One-VS-One montrant le nombre de victoires/égalités/pertes et la P-Valeur lors de la comparaison des modèles H-Inception, H-InceptionTime proposés avec les modèles Inception, InceptionTime et ROCKET d'origine sur les 128 jeux de données de l'archive UCR.

ResNet (en haut à droite). Cependant, en comparant H-FCN au modèle plus complexe Inception (en bas à gauche), on constate que H-FCN obtient des résultats légèrement inférieurs. En outre, nous avons comparé notre modèle H-FCN à l'état de l'art InceptionTime. Afin de comparer ces modèles, nous avons également créé un ensemble de modèles H-FCN résultant en H-FCNTime. Les résultats présentés dans la Figure 8 (en bas à droite) montrent qu' InceptionTime est significativement meilleur que H-FCNTime. Cela suggère que même si les filtres créés manuellement permettent d'améliorer les performances du FCN, les limitations de l'architecture FCN empêchent la capture de caractéristiques plus complexes pour être compétitif avec des architectures plus profondes comme InceptionTime.

H-InceptionTime. Afin d'évaluer la contribution de nos filtres créés manuellement dans une architecture plus profonde, nous avons considéré le modèle InceptionTime [18]. Comme dans le travail original [18], nous avons utilisé une taille de lots de 64 et entraîné le modèle pour 1500 époques en utilisant un optimiseur Adam avec un taux d'apprentissage décroissant. Pour les filtres de détection de pente ascendante et descendante, nous avons utilisé 6 variations de longueurs $[2^i$ pour $i \in \{1, ..., 6\}]$. Pour les filtres de détection des pics, nous avons considéré 5 variations de longueurs $[6, 12, 24, 48, 96]$. Nous avons comparé nos modèles adaptés H-Inception et H-InceptionTime avec les modèles de l'état de l'art Inception, InceptionTime et ROCKET. Les résultats, calculés sur une moyenne de cinq exécutions, sont présentés dans la Figure 9 à l'aide de diagrammes victoire/égalité/perte.

Nous pouvons d'abord observé que H-Inception a plus de victoires que Inception (en haut à gauche) avec une P-Valeur inférieure à 0.05, ce qui signifie que les deux modèles sont significativement différents sur l'archive UCR. De plus, la comparaison entre H-Inception et InceptionTime (en bas à gauche) montre que InceptionTime, grâce à l'ensemble, est significativement meilleur que notre modèle H-Inception. Néanmoins, après avoir comparé les performances des méthodes d'ensemble H-InceptionTime et InceptionTime (en haut à droite), nous pouvons constater que notre modèle H-InceptionTime surpasse significativement la méthode InceptionTime originale. Enfin, en comparant H-InceptionTime avec ROCKET (en bas à droite), nous pouvons remarquer que H-InceptionTime bat ROCKET dans le compte des gains/égalités/pertes sur les 128 jeux de données de l'archive UCR. Cependant, la P-Valeur suggère qu'il n'y a pas de différence significative entre ces deux classifieurs sur les 128 jeux de données de l'archive UCR.

4.3 Filtres Entraînables contre Non-Entraînables

Afin d'évaluer la pertinence de nos filtres créés manuellement, nous avons proposé de les comparer avec les filtres appris dans les modèles originaux. Nous avons également proposé d'évaluer l'impact des filtres créés manuellement incorporés dans le modèle H-FCN. Nous avons comparé les 64 filtres appris par la première couche du H-FCN et les 128 filtres appris par la première couche du FCN original. Nous avons également incorporé nos filtres détectant des pentes ascendantes et descendantes créés manuellement dans la comparaison. Une fois la distance DTW calculée pour chaque paire de filtres, nous les projetons dans un espace bi-dimensionnel en utilisant la méthode *T*-

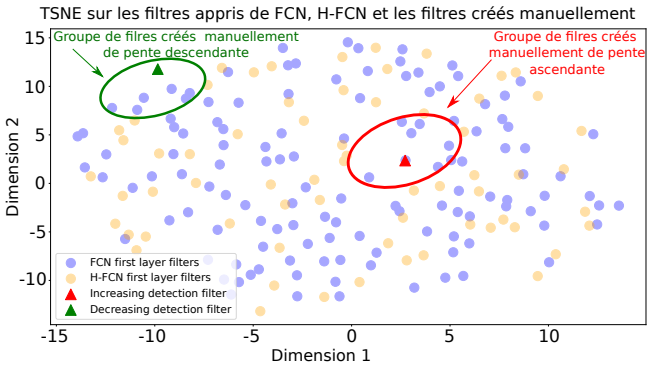

FIGURE 10 – Projection bi-dimensionnelle T-SNE des 128 filtres appris par la première couche du FCN original et des 64 filtres appris par la première couche du H-FCN sur le jeu de données CricketY de l'archive UCR. Les deux filtres de détection de pente ascendante et descendante créés manuellement sont également projetés dans l'espace bi-dimensionnel. Nous avons utilisé la métrique DTW comme distance pour le T-SNE.

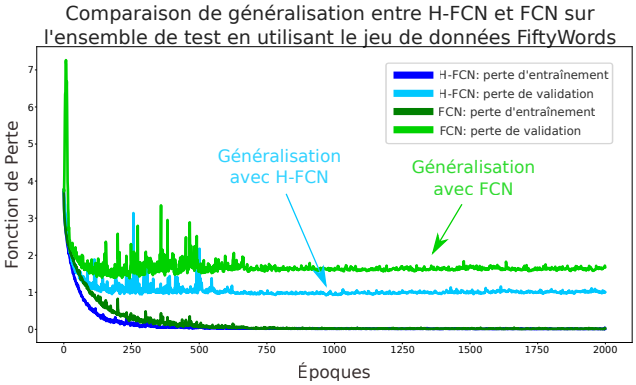

FIGURE 11 – Phase d'apprentissage des architectures FCN et H-FCN sur le jeu de données FiftyWords tout en surveillant la perte de validation sur le jeu de test. La perte d'entraînement de H-FCN (en bleu) converge plus rapidement que la perte d'entraînement de FCN (en vert). De plus, la perte de validation de H-FCN (en cyan) est toujours inférieure à la perte de validation de FCN (en vert clair). Cela montre que H-FCN généralise mieux que FCN sur ce jeu de données.

distributed Stochastic Neighbor Embedding (T-SNE) [29], comme le montre la figure 10 sur le jeu de données CricketY. Nous pouvons observer que les filtres appris par H-FCN (en orange) sont assez similaires aux filtres appris par FCN (en bleu). Cependant, nous pouvons remarquer quelques zones vides dans la distribution des filtres H-FCN, mises en évidence par des ellipsoïdes rouges et verts. Ces zones correspondent au filtre détectant une pente ascendante (triangle rouge) et au filtre détectant une pente descendante (triangle vert) créés manuellement. Ces filtres créés manuellement peuvent alors être considérés comme des prototypes représentatifs de plusieurs filtres appris du modèle FCN. Ces filtres créés manuellement permettent de réduire le nombre de filtres appris dans H-FCN tout en laissant le modèle se concentrer davantage sur l'apprentissage d'autres filtres significatifs.

4.4 Généralisation avec l'aide des Filtres Créés Manuellement

Les résultats expérimentaux de la section 4.2 ont démontré que l'incorporation de filtres créés manuellement dans les modèles d'apprentissage profond augmente les performances. Cependant, nous avons montré dans la section 4.3 que les filtres créés manuellement dans le modèle H-FCN sont similaires à certains filtres appris dans le modèle FCN original. Ces observations soulèvent la question suivante : Si le modèle FCN peut apprendre par lui-même les filtres créés manuellement, pourquoi l'incorporation de filtres créés manuellement dans H-FCN aide-t-elle à être plus précis ? Nous pensons que cela peut s'expliquer par la meilleure capacité des filtres créés manuellement à généraliser aux séries temporelles non vues. En effet, sans filtres créés manuellement comme dans FCN, le modèle optimise les poids des filtres afin de capturer les motifs pondérés qui correspondent presque parfaitement à l'ensemble d'apprentissage. Il est alors plus susceptible d'être sur-ajusté sur l'ensemble d'entraînement, un problème bien connu dans les approches basées sur l'apprentissage profond pour la TSC. À l'inverse, dans H-FCN, l'incorporation de filtres génériques créés manuellement permet de capturer des modèles généraux qui sont plus susceptibles d'être également pertinents pour des séries temporelles non vues pendant la phase de test.

Afin de valider cette hypothèse, nous avons proposé de comparer les courbes d'entraînement et de validation pour les modèles FCN et H-FCN sur le jeu de données FiftyWords, comme le montre la Figure 11. Nous notons que la perte de validation a été calculée sur l'ensemble de test car l'archive UCR ne fournit que des ensembles d'entraînement et de test. Cela a été fait uniquement pour surveiller le comportement de la généralisation et non pour affiner les hyper-paramètres. Nous pouvons clairement observer que la perte de validation de H-FCN converge vers une valeur beaucoup plus faible que la perte de validation de FCN. Cela montre que H-FCN généralise mieux sur le jeu de données FiftyWords et cela explique pourquoi il obtient une précision supérieure de 15% sur l'ensemble de test par rapport à FCN.

5 Conclusion

Dans cet article, nous avons abordé le problème de la classification des séries temporelles à l'aide d'architectures de réseaux neurones convolutifs. Nous avons proposé d'améliorer ces architectures en combinant les filtres appris avec des filtres créés manuellement. En particulier, nous avons conçu trois filtres génériques pour détecter les pentes ascendantes, les pentes descendantes et les pics. Nous avons

évalué l'impact de l'incorporation de ces filtres créés manuellement dans les architectures d'état de l'art en utilisant les jeux de données de l'archives de l'UCR. Les résultats ont montré des améliorations significatives non seulement pour l'architecture FCN de base mais aussi pour l'architecture plus complexe InceptionTime. Nous pensons que ce travail est une première étape dans la découverte de filtres de convolution génériques permettant non seulement d'améliorer les performances des architectures profondes mais aussi de réduire le nombre de paramètres à entraîner. Dans le cadre de travaux futurs, nous souhaitons étudier de nouveaux filtres créés manuellement en explorant d'autres motifs génériques ou en combinant plusieurs.

Remerciements

Ce travail a été soutenu par le projet ANR DELEGATION (subvention ANR-21-CE23-0014) de l'Agence Nationale de la Recherche française. Les auteurs tiennent à remercier le Centre de Calcul Haute Performance de l'Université de Strasbourg pour avoir soutenu ce travail en fournissant un support scientifique et un accès aux ressources de calcul. Une partie des ressources de calcul a été financée par le projet Equipex Equip@Meso (Programme Investissements d'Avenir) et le CPER Alsacalcul/Big Data. Les auteurs tiennent également à remercier les créateurs et fournisseurs de l'archive UCR.

Références

[1] E. Ay, M. Devanne, J. Weber, and G. Forestier. A study of knowledge distillation in fully convolutional network for time series classification. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2022.

[2] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off : a review and experimental evaluation of recent algorithmic advances. *Data mining and knowledge discovery*, 31(3) :606–660, 2017.

[3] Alessio Benavoli, Giorgio Corani, and Francesca Mangili. Should we really use post-hoc tests based on mean-ranks ? *The Journal of Machine Learning Research*, 17(1) :152–161, 2016.

[4] Victor Bogdan, Cosmin Bonchiş, and Ciprian Orhei. Custom extended sobel filters. *arXiv preprint arXiv :1910.00138*, 2019.

[5] 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. *IEEE/CAA Journal of Automatica Sinica*, 6(6) :1293–1305, 2019.

[6] 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.

[7] 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.

[8] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7 :1–30, 2006.

[9] M. Devanne and S. M. Nguyen. Multi-level motion analysis for physical exercises assessment in kinaesthetic rehabilitation. In *IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 529–534. IEEE, 2017.

[10] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Transfer learning for time series classification. In *2018 IEEE international conference on big data (Big Data)*, pages 1367–1376. IEEE, 2018.

[11] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Adversarial attacks on deep neural networks for time series classification. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

[12] Wenshuo Gao, Xiaoguang Zhang, Lei Yang, and Huizhong Liu. An improved sobel edge detection. In *2010 3rd International conference on computer science and information technology*, volume 5, pages 67–71. IEEE, 2010.

[13] Nhat-Duc Hoang and Quoc-Lam Nguyen. Fast local laplacian-based steerable and sobel filters integrated with adaptive boosting classification tree for automatic recognition of asphalt pavement cracks. *Advances in Civil Engineering*, 2018, 2018.

[14] Ali Ismail-Fawaz, Maxime Devanne, Jonathan Weber, and Germain Forestier. Enhancing time series classification with self-supervised learning. In *15th International Conference on Agents and Artificial Intelligence : ICAART 2023*, pages 1–8. INSTICC, SciTePress, 2023.

[15] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Data augmentation using synthetic data for time series classification with deep residual networks. In *ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data*, 2018.

[16] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Evaluating surgical skills from kinematic data using convolutional neural networks. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 214–221. Springer, 2018.

[17] 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.

[18] 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.

[19] Diederik P Kingma and Jimmy Ba. Adam : A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[21] Baptiste Lafabregue, Jonathan Weber, Pierre Gançarski, and Germain Forestier. End-to-end deep representation learning for time series clustering : a comparative study. *Data Mining and Knowledge Discovery*, 36(1) :29–81, 2022.

[22] Arthur Le Guennec, Simon Malinowski, and Romain Tavenard. Data augmentation for time series classification using convolutional neural networks. In *ECML/PKDD workshop on advanced analytics and learning on temporal data*, 2016.

[23] François Petitjean, Alain Ketterlin, and Pierre Gançarski. A global averaging method for dynamic time warping, with applications to clustering. *Pattern recognition*, 44(3) :678–693, 2011.

[24] G. Pialla, M. Devanne, J. Weber, L. Idoumghar, and G. Forestier. Data augmentation for time series classification with deep learning models. In *Advanced Analytics and Learning on Temporal Data (AALTD)*, 2022.

[25] Gautier Pialla, Hassan Ismail Fawaz, Maxime Devanne, Jonathan Weber, Lhassane Idoumghar, Pierre-Alain Muller, Christoph Bergmeir, Daniel Schmidt, Geoffrey Webb, and Germain Forestier. Smooth perturbations for time series adversarial attacks. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 485–496. Springer, 2022.

[26] M Rußwurm, C Pelletier, M Zollner, Sébastien Lefèvre, and M Körner. Breizhcrops : A time series dataset for crop type mapping. *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 43 :1545–1551, 2020.

[27] 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*, pages 1–24, 2022.

[28] Tsegamlak Terefe, Maxime Devanne, Jonathan Weber, Dereje Hailemariam, and Germain Forestier. Time series averaging using multi-tasking autoencoder. In *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1065–1072. IEEE, 2020.

[29] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[30] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks : A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*, pages 1578–1585. IEEE, 2017.

[31] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer, 1992.

[32] Jiaping Zhao and Laurent Itti. shapedtw : Shape dynamic time warping. *Pattern Recognition*, 74 :171–184, 2018.

[33] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J Leon Zhao. Exploiting multi-channels deep convolutional neural networks for multivariate time series classification. *Frontiers of Computer Science*, 10(1) :96–112, 2016.