


Internship report M2 (IM)

Deep Self-Supervised Learning for Time Series Classification

HABIB Mario

Internship carried out from 30/01/2023 to 21/07/2023

Master 2 Informatique et Mobilité

Laboratory : IRIMAS

Internship tutors : FORESTIER Germain

Referent teacher : LEPAGNOT Julien

Éducational institution : Université de Huate-Alsace / FST

UNIVERSITÉ HAUTE-ALSACE

Formulaire d'information sur le plagiat

Dans le règlement des examens validé par la CFVU du 2 octobre 2014, le plagiat est assimilé à une fraude. Cette information est présentée à chaque étudiant de l'Université de Haute Alsace susceptible de rédiger un document long de type thèse, mémoire ou rapport de stage. Le formulaire signé devra obligatoirement être intégré au document.

Le plagiat consiste à reproduire un texte, une partie d'un texte, des données ou des images, toute production (texte ou image), ou à paraphraser un texte sans indiquer la provenance ou l'auteur.

Le plagiat enfreint les règles de la déontologie universitaire et il constitue une fraude. Le plagiat constitue également une atteinte au droit d'auteur et à la propriété intellectuelle, susceptible d'être assimilé à un délit de contrefaçon.

En cas de plagiat dans un devoir, dossier, mémoire ou thèse, l'étudiant sera présenté à la section disciplinaire de l'université qui pourra prononcer des sanctions allant de l'avertissement à l'exclusion.
Dans le cas où le plagiat est aussi caractérisé comme étant une contrefaçon, d'éventuelles poursuites judiciaires pourront s'ajouter à la procédure disciplinaire.

Je soussigné(e) ..... Mario Habib Fathi .....

Etudiant(e) à l'Université de Haute Alsace en : ..... 2022 / 2023 .....

Niveau d'études : ..... Master M2 .....

Formation ou parcours : ..... Informatique et Mobilité .....

Reconnaît avoir pris connaissance du formulaire d'information sur le plagiat.

Fait à ..... Mulhouse ..... Le ..... 6/7/2022 ..... Signature : Mario Habib

Acknowledgments

I want to thank M. FAWAZ Ali for his support, help, and for guidance during my internship and introducing me to the life of a researcher.

I want to also thank M. FORESTIER Germain for accepting me to do this internship, for his supervision of my work, and for his help and articles that he regularly provided to increase my knowledge in the field.

Also, there is M. ABDULLAYEV Javidan, who helped me by giving advice when I faced any problem.

Also, I would like to thank Mme. LAHOUD Christine from the UNIVERSITÉ FRANÇAISE D'EGYPTE, without whom, I would not be able to have this opportunity to do my Master M2 or to do a research internship in France.

There is also Mme. MOUSSA Shrein from the UNIVERSITÉ FRANÇAISE D'EGYPTE, who supervised me from Egypt and kept in touch to know the updates.

Finally, I want to thank everyone in the team of MSD for their support and for creating an encouraging environment and for having a good time with them.

Abstract

This report covers my internship at the lab of MSD at the Université de Haute-Alsace. It was focused on time series classification and the aspects related to it. It starts by explaining the structure of the lab at the university, then the basic knowledge that anyone should be aware of in order to work in the field of time series classification, and also what the lab's contributions were in this field. After explaining the basic knowledge, our contribution to the lab was in two subjects: ROCKET, which is a fast method used for time series classification, and Jigsaw puzzle, which is a technique used in images for data augmentation, and we tried to adapt it for time series classification. Finally, we discussed the conclusion of this internship, its benefits, and our contribution to the lab.

Key words: Time series Classification, Data augmentation, Deep learning, convolution, Jigsaw puzzle, Rocket.

Contents

1 Introduction: **1**

2 LAB: **1**

3 Internship: **4**

3.1 Background: . . . 4

3.1.1 Scientific Background: . . . 4

3.1.2 Deep learning: . . . 5

3.1.3 Traditional ways: . . . 8

3.2 ROCKET: . . . 11

3.2.1 Introduction: . . . 11

3.2.2 Tasks . . . 15

3.3 Jigsaw: . . . 17

3.3.1 Introduction . . . 18

3.3.2 Jigsaw in images: . . . 18

3.3.3 Model Phases: . . . 19

3.3.4 Background: . . . 20

3.3.5 Autoencoder: . . . 20

3.3.6 Number of segments: . . . 20

3.3.7 Permutation list: . . . 21

3.3.8 Number of copies: . . . 23

3.3.9 Bottleneck: . . . 25

3.3.10 Gaps: . . . 25

3.3.11 Number of tests: . . . 26

3.3.12 Example: . . . 27

3.3.13 Fine Tuning: . . . 28

3.3.14 Architectures: . . . 28

3.3.15 Comparison Criteria: . . . 30

3.3.16 Results: . . . 30

4 Conclusion: **36**

A Algorithm used for optimizing list generation **40**

1 Introduction:

I have chosen to do my internship in a lab to help me gain insight into the life of a researcher, to know how to do the correct research, and to have the necessary information about the life of a researcher and whether to pursue in doing a PhD or not.
I also wanted to do my internship in time series because they exist in every aspect of our life, and one of the main reasons that made me do research in it is that I want to create a program that helps in the prediction of market price changes in the near future, because of the inflation and its impact on people, and one of the main aspects of this program is the analysing of the prices, which are represented in time series.
And I have found that IRIMAS is also interested in analysing and classification of time series, so that is the reason that I applied for an internship at the lab of IRIMAS in the team of MSD.

2 LAB:

My internship was in the lab of IRIMAS, which stands for L'Institut de Recherche en Informatique, Mathématiques, Automatique et Signal, which is translated into -The Institute for Research in Computer Science, Mathematics, Automation and Signal-.
IRIMAS is a research lab that is part of Université de Haute-Alsace -UHA- in Mulhouse, France, founded in January 2018. This laboratory focuses on multiple research aspects that are related to our modern day, which affects the lives of a lot of people like in computer science and AI.

This laboratory consists of three departments, which are:

1. Informatique
2. MATHÉMATIQUES
3. AUTOMATIQUE, SIGNAL ET IMAGE

And each one of those departments is divided into multiple teams -eight in total- and we can see the clear hierarchy in the following figure 1.
I was doing my internship in the team MSD, which stands for Modélisation et Science de Données, and it is part of the Department of Informatique. This team's main area of research is in modelling and data science, and one of their main topics in research is time series classification and analysis.

The team has access to different resources starting from five different local servers to do the large tasks and research, especially if the research requires the training of a model over several days, there is also another server which is more powerful, called Mésocentre of

[1] https://www.irimas.uha.fr/index.php/organigramme/

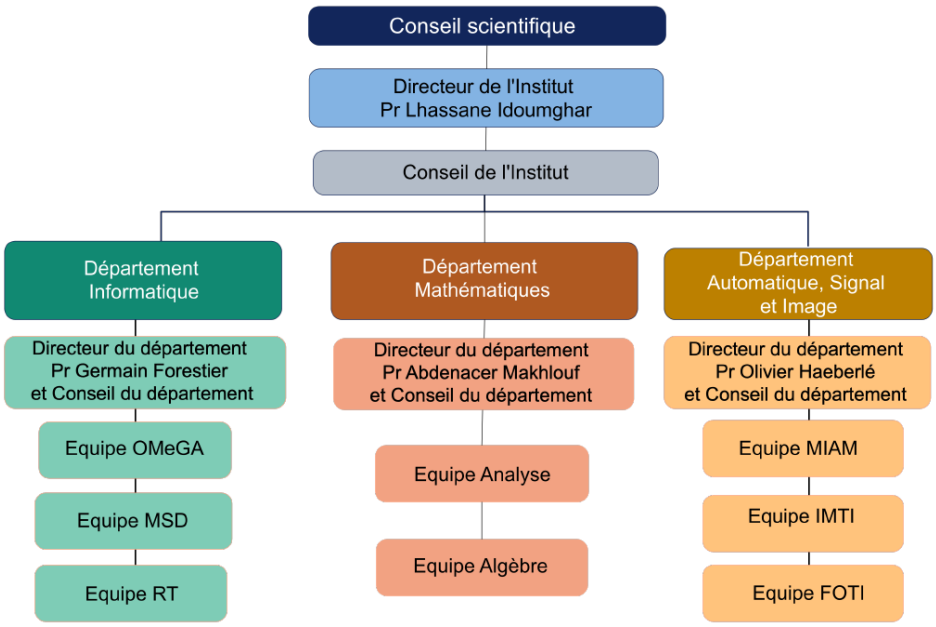


Fig. 1: This is the current structure for IRIMAS as it is explained by their website [1].

Strasbourg, and finally, there is also a robot which is called 'Reachy' it is a humanoid robot, with a main purpose of teaching students various skills. It is still in the development stage, but it can do various tasks, like reading facial expressions and telling if the person is happy, sad, etc. Also, it is capable of grabbing small items and various other tasks.

Also, each person in the lab has a PC that is operated by Linux, and most of these PCs have a GPU, it is not a strong GPU to do large experiments like in the servers, but it has a purpose for doing small experiments.

Doing research in the lab usually consists of three stages. The first stage is launching an experiment on a very small subset of datasets of the experiment, and it is usually done on the PC, and it has the purpose of knowing only the general insight like if the program is working correctly, if there were any bugs and also to be able to see the results, and decides whether this research topic could lead to good results so we would continue to do it on the server or to stop and find another research topic.

In the second stage, they launch the experiment on a server because it has better resources and can be launched for several days. This stage helps to give a general behaviour about the data. It even helps in creating some hypotheses and assumptions that can be either confirmed or rejected. This stage is also critical because if the results are different from what we expected them to be or are shown to lead to worst results, then this can lead to the abandonment of this research topic in order to save resources. This stage usually takes around 10 to 20 % of the available datasets of the experiment.

In the third and final stage, the experiments are launched on the server for all the available datasets for this experiments, and after having all the results after several days. The analysing of those results starts in order to confirm or reject the assumptions that we had in the previous stage and also to have a better understanding of behaviour in those results,

and it could also lead to new research topics that the lab can find and work on them, and if possible, a paper may be written on these results and then being published which could also lead change how the research community think about this subject and open a whole new topic for research.

In my team, we were 11 persons in the room divided into three categories:

1. Three persons were PhD students
2. Four persons were research engineers
3. Four persons were doing their internship

There were other PhD students, but unfortunately, I did not have the chance to meet them. We were working on various activities, but this did not prevent our work from being intertwined on some occasions, and there were three main categories our team worked on, most of us worked on time series classification and Human motion acts, and there were only two persons that were working on the robot.
There was also something called the journal club, which was held every two weeks on Thursday at around 2 pm. It was a voluntary presentation where anyone could present a topic or a paper that he finds interesting and share what he understood and if it was possible to apply it to our research. And it was an enrichment experiment where it had helped me to gain additional knowledge, and some of the topics that I found to be interesting were:

1. DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation by (Ruiz et al. 2023)
2. A Similarity-based Time Series Source Dataset Selection Method for Transfer Learning by (Zhang et al. 2023)

Those were some of the papers that I have found to be interesting. Unfortunately, I could not participate in this experiment because I was unable to find any interesting papers to share.
Those papers were interesting to me because they gave me a lot of wondering and raised a few questions to be able to search for, like for DreamBooth, How could we be able in the future to differentiate between AI-generated images and original images? What techniques would be able to use it? What will be the legality consequences for such techniques, especially if it was used in crimes like what has happened in China[2]? And how to prevent it? Those were just some of the questions that got to me when I heard this presentation.
Finally, there was also the IRIMAS seminar that was held on the 1st of June. And it was important for me because it has given me insight into what other teams were doing and their research which helped to wider my point of view and find the intersection and similarities between the different teams and how they complete each other.

[2]https://www.reuters.com/technology/deepfake-scam-china-fans-worries-over-ai-driven-fraud-2023-05-22/

3 Internship:

During the internship, which was focused on time series classification, before starting any work, I had to familiarise myself with basic concepts that are necessary to understand, which would be essential to use during the rest of the internship.

3.1 Background:

In this subsection, we will talk about the first month of the internship, which was focused on knowing the basics of time series classification. This month was essential to be able to understand the critical concepts.

3.1.1 Scientific Background:

For time series classification, we must mention some important scientific definitions, but first, we must define what a time series is.

Definition 1 Time series: A sequence of recorded values at specific timestamps, usually with a well-defined time interval between each timestamp. This period is usually in seconds and minutes. However, it could also be in days and months. And it can be divided into two categories: Univariate time series -UTS- and Multivariate time series -MTS-.

Definition 2 UTS: It is the simplest form in time series. It has an ordered set according to the recorded values at timestamps, and it has the following equation: $X = [x_1, x_2, ..., x_n]$ Where X is the time series, it has a length of n, and the x_i is the value recorded at the i^{th} timestamp.

Definition 3 MTS: It is a collection of different UTS, and it has the following form: $M = [X^1, X^2, ..., X^N]$ Where M is the time series, and it has a length of N UTS, and the X^i is the i^{th} UTS in this MTS.

Definition 4 Dataset: It is a collection of similar data -samples- in a specific domain, and these data are usually separated into classes, and each class has a similar distinctive pattern which makes it unique from other classes. For the time series domain, a dataset can be constructed from UTS or MTS, and it can be represented as follows: $D = [(X^1, Y^1), (X^2, Y^2), ..., (X^N, Y^N)]$ Where D is the dataset, and it has N number of samples, each sample composes of a pair of (X^i, Y^i) where X^i can either be a UTS or an MTS and its corresponding class is the Y^i.

Definition 5 Time series classification -TSC-: It is the domain that specialises in finding new methods and algorithms to help classify time series in more accurate and faster ways. Also, it has the purpose of associating a given time series with the corresponding label.

Furthermore, we also like to further explain these definitions by giving an example:
An example of UTS is a sensor that reads data and saves the readings at regular intervals.
An example of MTS is to have multiple sensors that simultaneously read multiple different data -i.e. Temperature and Pressure- and save those readings at regular intervals.

For datasets, there is the UCR archive (Dau et al. 2019). It is an open repository specialising in UTS, and it has 128 different datasets covering different domains, and they are grouped into 15 different domains. Also, it is considered the largest repository of UTS datasets available, and many researchers use it as a benchmark to test their approach.
There is another archive called UEA (Bagnall, Dau, et al. 2018). It is an archive that specialises in MTS datasets and has 30 datasets.

3.1.2 Deep learning:

After knowing these basics, we had to know the different state of the arts -SOTA- used in deep learning for TSC, and according to (Ismail Fawaz, Forestier, et al. 2019), which was a review that compared the best architectures, at that time, used for TSC, and it had found that the best architectures were:

1. Fully Convolutional Neural Network -FCN-
2. Residual Network -ResNet-

Both of those architectures were mentioned in (Wang et al. 2017), and then there was another research (Ismail Fawaz, Lucas, et al. 2020) that created another architecture that is called InceptionTime that has outperformed the ResNet architecture.

We are going to explain each one of those architectures in detail.

FCN:

The FCN consists of three consecutive convolution blocks. Each block consists of three layers. The first layer is a convolution layer, followed by a batch normalisation layer, followed by a rectified linear unit -ReLU- as the activation function. They used batch normalisation to speed up the convergence speed, while the ReLU is used in order not to activate all the neurons at the same time because if the output of the previous layer is less than 0, then this neuron will be deactivated. And all of these are in order to help in generalisation.
Those three blocks have a filters of the following sizes {128,256,128} and a 1-D kernels of sizes {8,5,3}
We can also represent a block by the equation 1:

$$\begin{aligned} y &= W \otimes x + b \\ s &= BN(y) \\ h &= ReLU(s) \end{aligned} \tag{1}$$

Then after those three blocks, we have a global average pooling -GAP- layer followed by a softmax layer.

ResNet:

The ResNet consists of three residual blocks. A residual block can be represented using equation 2. Each one of those residual blocks is composed of three consecutive convolution blocks, with each being represented by equation 1, which is a convolution layer followed by a batch normalisation layer followed by a ReLU activation function, but what makes ResNet different is the existence of shortcuts between each residual block. These shortcuts are linear, and they link the input of a residual block to the output of the second convolution blocks inside the same residual block -equation 2- which makes it easy for the gradient to pass through these connections, which helps to solve the vanishing gradient problem.

$$\begin{aligned} h_1 &= Block_1(x) \\ h_2 &= Block_2(h_1) \\ y &= W \otimes h_2 + b \\ s &= BN(y) + BN(x) \\ \hat{h} &= ReLU(s) \end{aligned} \tag{2}$$

Here $Block_i$ refers to the block which is represented by equation 1 -FCN Block- while for the third block, it is a special case because the shortcut is related to it, so after doing the convolution, the s is the sum of both batch normalisation for the input -denoted x- which is fed through the shortcut and the output y of the convolution then they are fed to ReLU activation function. Finally, we have a GAP layer followed by a softmax layer.
For the number of kernels, they are 1-D kernels of sizes {8,5,3} per residual block, but the number of filters is fixed across all the convolution blocks for the same residual block, which are {64,128,128}.

InceptionTime:

The InceptionTime was first proposed by (Ismail Fawaz, Lucas, et al. 2020). It is an ensemble of five models, and each one of them has multiple cascading inception modules. All the models have the same architecture but with different randomly initialised weight values. A model consists of two different residual blocks, and each one of those blocks consists of three inception modules. Also, there is a shortcut, a linear connection that transfers the block's input to be added to its output, which reduces the gradient effect. After those two residual blocks, there is a GAP layer followed by a softmax layer.
Inside the inception module, which is shown in figure 2 (Ismail Fawaz, Lucas, et al. 2020), the input can be either MTS or UTS -in case it was the first layer that is connected to the input layer, and the input was UTS-. So, after having the input, there are two parallel operations. In the first operation, the first layer is a bottleneck layer with m filters with a stride of 1 and length of 1, which has the purpose of sliding these filters. In the general case -MTS- this layer will significantly reduce dimensionality, reducing the overfitting problem. After the bottleneck layer, multiple convolution filters with different lengths are sliding simultaneously on the same data -output of the previous layer-. For the second parallel operation, which they have decided to add to make the model less invariant to small perturbations, is a

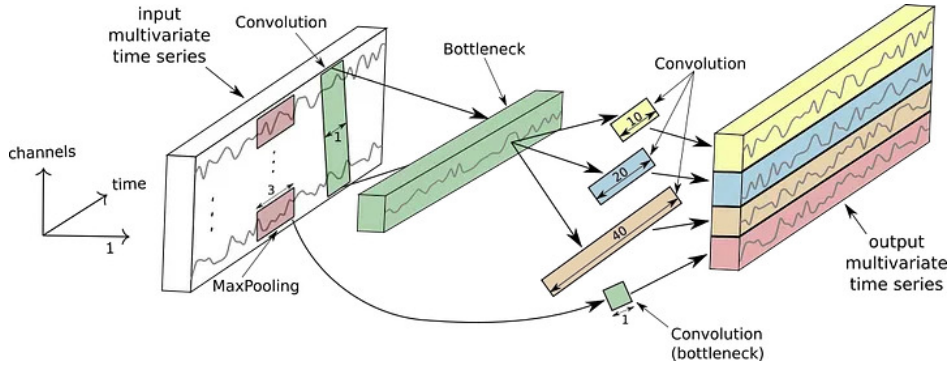


Fig. 2: The inception module as explained in Ismail Fawaz, Lucas, et al. 2020.

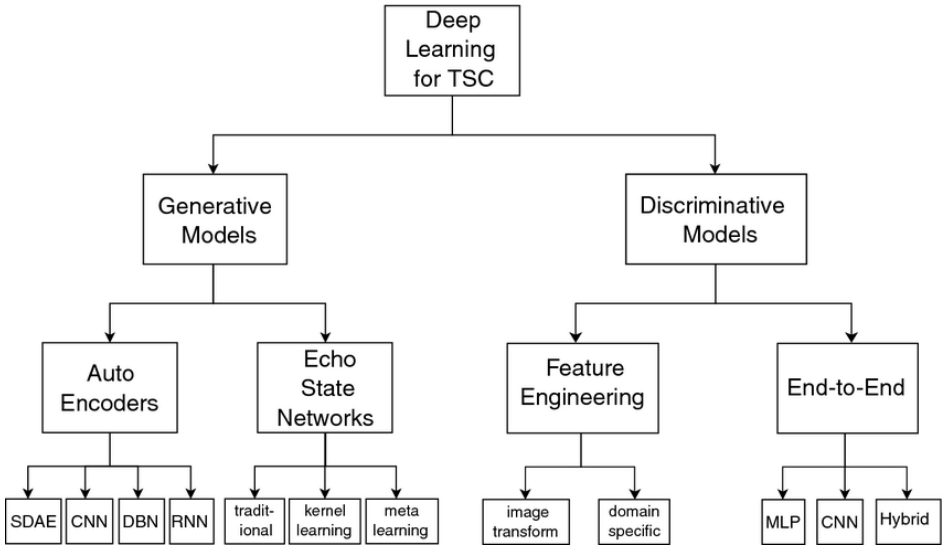


Fig. 3: The overview of the different deep learning approaches for TSC as shown in Ismail Fawaz, Forestier, et al. 2019.

MaxPooling operation followed by a bottleneck layer. Finally, the outputs of those two parallel operations are going to be concatenated together to form a new MTS, which is going to be the output of this module.
Finally, their proposed Inception module is to have three sets of filters with lengths = {10,20,40}, and the default bottleneck value was set to $m = 32$.

Also, this review has helped in identifying the different deep learning approaches, figure 3 from (Ismail Fawaz, Forestier, et al. 2019), used in TSC.

There was also the paper (Ismail-Fawaz et al. 2022). In this paper, the authors described the effect of adding handcrafted filters for the architectures of FCN and InceptionTime. They have created four models:

1. CO-FCN: Customs Only FCN
2. Hybrid FCN

3. Hybrid Inception
4. Hybrid InceptionTime

In the CO-FCN, they replaced the first convolution layer with custom filters. Afterwards, they wanted to test the effect of having a mix between the original architecture and the CO-FCN. Thus, they changed the first block into concatenating the resulting feature from both layers, the handcrafted and the convolution layers, creating the Hybrid FCN. This architecture helped the model to focus on other patterns. Finally, they decided to do the same hybrid architecture with InceptionTime, creating the Hybrid Inception and Hybrid InceptionTime. The results were that InceptionTime was better than Hybrid Inception due to InceptionTime being an ensemble. When they compared both ensembles, the Hybrid InceptionTime was better than InceptionTime.
For the handcrafted filters, they have created three filters:

1. Increasing Trend: To detect when the values increase in time.
2. Decreasing Trend: To detect when the values are decreasing in time.
3. Peak: When the series has an oscillation where there is an increasing trend followed by a decreasing trend with a huge variance between them.

Those filters were created because when backpropagation is used, the error is propagated back into the first layer, which makes it harder for the model to learn a general filter. And adding more filters could lead to overfitting, so an alternative and easier approach is to add a handcrafted filter.

There was another paper (Fawaz et al. 2018) where the authors discussed the effect of using transfer learning in time series. They aimed to see if the models could achieve better results if trained on another dataset and then fine-tuned for the target dataset. They tested on the 85 datasets of the old UCR archives, having a total of 7140 different models, and the results proved that transfer learning in TSC could lead to better or worse results. They have also tried to implement a method that depends on Dynamic Time Warping to measure inter-datasets similarities. Their results showed that we can use this method to predict the best dataset to be used as a source for a target dataset. For this method, they averaged the set of series per class to create a prototype, and then they calculated the distances between each pair of datasets. Finally, they chose the pairs with the minimum distance between their prototypes.

3.1.3 Traditional ways:

After studying the main architectures used for deep learning, we had to study the traditional ways used in TSC. It started by learning dynamic time wrapping -DTW- distance, first proposed in (Sakoe et al. 1978) as a dynamic programming algorithm for speech recognition. Due to the similarities between the two fields, it was adopted for TSC, and we learned it from (Tavenard 2021). It is a distance metric used to measure the similarity between two different time series, and it is specifically used for it. It has three conditions:

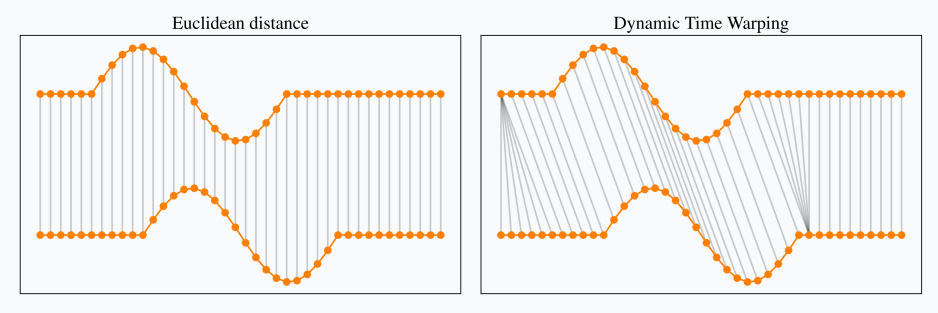


Fig. 4: The difference between Euclidean distance and DTW taken from Tavenard 2021.

1. The first index of the first series must be matched with the first index of the second series.
2. The last index of the first series must be matched with the last index of the second series.
3. The indexes are monotonically increasing, which means that both indexes for both series are either increasing or staying the same, but they are never decreasing.

It is essentially an alignment-based metric, and it finds the distances between matched features -distinctive patterns- which is also one of the reasons that make it slower, unlike other distance metrics, like Euclidean distance, which matches only timestamps. Figure 4 (Tavenard 2021) visualises the difference between the two distances.

This metric alone does not add much, but when we combined it with the K-nearest neighbour -KNN- classification algorithm, especially 1-NN, creating the (1-NN-DTW), it performed as one of the strongest traditional classifiers for time series, which made many researchers focus on researching traditional ways to develop other ensembling methods to outperform the (1-NN-DTW).
In fact, one of the first tasks was to develop this classification algorithm from scratch -Both the basic version and windowed version for DTW- using Python[3] which helped us better understand the algorithm and also improving our level in using modules in Python. We implemented the KNN and the Support vector machine -SVM- classifiers for the classifier part. As for the distance metrics, the DTW windowed is a modified version of DTW. It adds a constraint to keep the wrapping close to the main diagonal by restricting the set of possible matches.
Another purpose of this task was to be able to deal with the UCR archives in loading, manipulating, and extracting data from the TSV files and getting the classes -labels- from the TSV file, which were available in the last column. Also, the UCR datasets have the same hierarchy. First, each dataset is available in a directory called by its name, and there are three files inside this directory. The first two are for the dataset itself, and they are DATASET_TEST and DATASET_TRAIN. The last one is a file that explains this

[3]https://docs.python.org/3.9/

particular dataset in detail. All the tasks done during the internship can be found in the accompanying repository: https://github.com/MarioHabibFathy/Stage.

After doing this task, We had to compare the results obtained with the standard results[4] because NN-DTW is a deterministic algorithm, We found that there was a difference between both results. When we further investigated, we found that we did not use the Z-normalization on the dataset, which was something important that we learned, and upon using it, the results were correct.

Also, one of the things that we learned but needed more time to implement was proximity forest -PF- (Lucas et al. 2019). It is a tree-based ensemble. The difference is that in standard trees, the difference depends on value, which, in our case, will be on timestamps, and the problem is that different series belonging to the same class are not necessarily to be aligned by the same timestamp. So, PF uses 11 distance measures, the same one used in another algorithm called Elastic Ensemble (Lines et al. 2015), and from those 11 distances, PF chooses from them randomly for every node in the tree, and the splits are based on two criteria. First, the similarity of choosing an exemplar and second, choosing to form different forests where the examples are chosen randomly.

Some other methods and approaches were mentioned in detail at (Bagnall, Lines, et al. 2017). Then there was a recent study that can be considered as an updated version that has also included the recent methods like deep learning and convolutions (Middlehurst et al. 2023) and according to it, we have eight different categories for time series classification that are the following:

1. Distance Based
2. Feature Based
3. Interval Based
4. Shapelet Based
5. Dictionary Based
6. Convolution Based
7. Deep Learning
8. Hybrid

Our second task was implementing logistic regression from scratch to understand the softmax function. So, we had to learn some related functions, like the sigmoid function and the gradient descent. We also learnt about regularisation techniques used for it and implemented two of them, L1 and L2 regularisation.
After implementing the logistic regression from scratch, we had to implement it using

[4] https://www.cs.ucr.edu/~eamonn/time_series_data_2018/

TensorFlow[5] while also using GPU, so we had also to install CUDA[6]. It was an important task because, after that, most of the tasks that we received afterwards required the use of TensorFlow along with the GPU.

Those were the basic knowledge that any time series researcher needs to understand before diving into the world of time series.
After knowing the main basics, there were two main subjects that the internship took part in:

1. ROCKET
2. Jigsaw puzzle

We are going to discuss each one of them in detail.

3.2 ROCKET:

After understanding the basics, one of the major tasks that we had to work with was ROCKET (Dempster, Petitjean, et al. 2020) and its variant versions -MultiRocket and MiniRocket- (Tan et al. 2022; Dempster, Schmidt, et al. 2021), which they are all considered under the family of the convolution based approach.

3.2.1 Introduction:

ROCKET:

ROCKET, which stands for **R**and**O**m **C**onvolutional **KE**rnel **T**ransform, depends on the basic parameters of a kernel, which are:

1. Length
2. Weight
3. Bias
4. Dilation
5. Padding

The main idea of ROCKET is to have a lot of random kernels -according to the authors, they recommend having 10000 kernels- each one has a different combination of those parameters so that each kernel can capture a basic pattern in the series, with multiple kernels combined, can capture complex patterns.

After the kernels, it is followed by either a:

[5] https://www.tensorflow.org/
[6] https://developer.nvidia.com/cuda-toolkit

1. Logistic regression: Only when the number of training examples is significantly greater than the number of features.

2. Ridge regression: For the most cases. It is trained with L2 regularisation and using one versus the rest algorithm.

Also, those five parameters are not entirely random. ROCKET chooses from specific conditions:

1. Length: It chooses randomly from a specific length set {7,9,11} with equal probabilities

2. Weight: It is sampled from normal distribution $\mathcal{N}(0, 1)$

3. Bias: It is sampled from uniform distribution $\mathcal{U}(-1, 1)$

4. Dilation: It is sampled from an exponential scale $\lfloor 2^x \rfloor$, where x is sampled from a uniform distribution $x \sim \mathcal{U}(0, A)$ and $A = \log_2 \frac{l_{\text{input}}-1}{l_{\text{kernel}}-1}$ and l_{input} is the length of the input time series and l_{kernel} is the length of the kernel. It aims to ensure that the effective length of the kernel with the dilation is up to the length of the input series.

5. Padding: It is decided randomly with equal probability whether to apply padding or not. If it is decided to be applied, a number of zeros are appended to the star and the end of each series when the kernel is applied following this condition $\frac{l_{\text{kernel}}*d}{2}$ where d here is the value of the dilation, it is also applied in a way that the middle elements in the kernel are centred for every point in the series.

For the stride part, they used it always to be 1. As for the resulting features, they made it always double the number of kernels, i.e., for k kernels, there are $2k$ features. Also, they do not apply any nonlinear transformations like ReLU. Instead, they use max pooling and proportion of positive values -PPV-, which is calculated using the equation 3 where Z is the convolution output.

$$PPV(Z) = \frac{1}{n} \sum_{i=1}^{n} [z_i > 0] \tag{3}$$

Also, ROCKET is considered one of the fastest methods that are available where it achieves SOTA accuracy in 1 hour and 50 minutes on the 85 datasets of 'bake off', for comparison InceptionTime (trained and tested using GPUs) takes more than six days. We also would like to add that ROCKET is trained and tested using CPU, but according to the authors, it can also work on GPUs. Also, according to the same authors(Dempster, Petitjean, et al. 2020): "Timings for Rocket are averages over 10 runs, performed on a cluster using a mixture of Intel Xeon E5-2680 v3 and Intel Xeon Gold 6150 processors, restricted to a single CPU core per dataset per run"(P.2). This was the method they used to calculate the time for ROCKET.

MiniRocket:

After understanding ROCKET, we had to study its successor versions, MiniRocket and MultiRocket. Starting from MiniRocket, it stands for **MINI**mally **R**and**O**m **C**onvolutional **KE**rnel **T**ransform, the term minimally is used here because they have restricted the randomness from ROCKET, except for the bias that is the only random part of MiniRocket, but, the values that are selected are produced by an entirely deterministic procedure.

So, they made the following changes:

1. Length: It uses kernels of a fixed length of 9, and it uses a subset of 84 of two values kernels.

2. Weight: It is constrained between two values α and β. They have used the values of $\alpha = -1$ and $\beta = 2$, but it can be any values with the condition that the sum of values of the weight must be zero, and to do that, they used kernels with 3 values of β and 6 values of α which gives the 84 possible kernels.

3. Bias: It is extracted from the convolution output, and the randomness comes from selecting a random training example, where the bias values are extracted from the quantiles of this output. They select the [0.25, 0.5, 0.75] quantiles to be used in calculating the PPV. There is another deterministic version, where bias values are extracted for the entire training set from the convolution output.

4. Dilation: There is no big change from ROCKET, but because the number of features is fixed, they made the maximum number of dilation per kernel equal to 32. Otherwise, it is sampled from an exponential scale $\lfloor 2^x \rfloor$, where x is sampled from a uniform distribution $x \sim \mathcal{U}(0, A)$ and $A = \log_2 \frac{l_{\text{input}}-1}{l_{\text{kernel}}-1}$ and l_{input} is the length of the input time series and l_{kernel} is the length of the kernel. It aims to ensure that the effective length of the kernel with the dilation is up to the length of the input series.

5. Padding: It is applied for each kernel/dilation combination so that half of those combinations use it and the other half does not. If it was applied, they used the standard zero padding at the start and the end of each input series.

For the resulting features, they disregarded the max pooling operations and used only the PPV for features. Also, they generate a total of 9996 features because it is less than 10000 and a multiplication of the 84 kernels. Also, they have used the advantages of having two-valued kernels to optimise further MiniRocket. When they calculate the PPV, they are also able to have its opposite values, the proportion of negative values -PNV- because $PNV = 1 - PPV$, which is being calculated for free without any extra computation. For convolution operations, they have turned it from multiplication into addition because of these two values, which are explained using the equation 4, where X is used for time series $[x_0, x_1, ..., x_n]$ and W for kernel weight $[w_0, w_1, ..., w_m]$ with d for dilation. Also, they have reused the convolution output, especially at the smaller dilations, because the same combinations of kernel/dilation are used to compute multiple features, which makes multiple features calculated with the cost of a

single convolution operation. Finally, they compute all the kernels at once. They do that by taking advantage of having three values of β and six values of α by doing $\frac{2}{3}$ of the computation for a given dilation to all the 84 kernels at once.

$$X * W_d = \sum_{j=0}^{m-1} x_{i-(\lfloor \frac{m}{2} \rfloor .d)+(j.d)} . W_j, \forall i \in \{0, 1, ..., n-1\} \tag{4}$$

For classifiers, they use the same two classifiers as ROCKET, logistic regression trained with Adam for larger datasets and a ridge regression classifier.

Finally, MiniRocket is considered the fastest SOTA in TSC, where it took only 8 minutes to compute 109 datasets, which were restricted to only a single CPU time. In contrast, ROCKET, for the same datasets, took 2 hours and 2 minutes. That is why the authors have recommended that MiniRocket be the default variant of ROCKET. The computation time was averaged over 30 resamples.

MultiRocket:

Finally, the last variant we studied was MultiRocket. It stands for MiniRocket with multiple pooling operators and transformations. MultiRocket is built on MiniRocket with a few variations. First, it calculates the first-order difference for a given time series, which creates a second representation. Second, for the original series, along with its first-order difference, they are convolved with the 84 kernels from the MiniRocket. Lastly, MultiRocket uses four pooling operations, PPV, along with three others.

They have decided to use the first-order difference to have diversity for MiniRocket, which helps in describing the rate of change between timestamps which helps in identifying the slope and the presence of certain patterns and outliers.

For the five main characteristics of convolutions, MultiRocket uses the same values as MiniRocket, with a small change that uses a different set of dilations and biases for each representation. The reason is that the first-order representation is shorter than the original by a value of 1. Which results in a slightly different set of kernels. Otherwise, the other elements, length, weight and padding, are the same for both sets.

For the resulting features, they have added another three operations along with PPV, resulting in a total of 49728 features for MultiRocket. The operations that they added were: Mean of positive values -MPV-, Mean of indices of positive values -MIPV- and Longest stretch of positive values -LSPV-, Figure 5 (Tan et al. 2022) visualises each one of those pooling operations. The MPV, as the name suggests, calculates the mean of all the positive values resulting from the convolution using the equation 5 where Z^+ represents the vector of positive values with a length of m.

$$MPV(Z) = \frac{1}{m} \sum_{i=1}^{m} z_i^+ \tag{5}$$

The MIPV has the purpose of calculating the mean for all indices of the positive values. It captures the indices of the positive values in the array of the convolution output and then calculates the mean. it is calculated using equation 6. Where I^+ indicates the positive values indices and m is the number of positive values. In case $m = 0$, the MIPV returns -1.

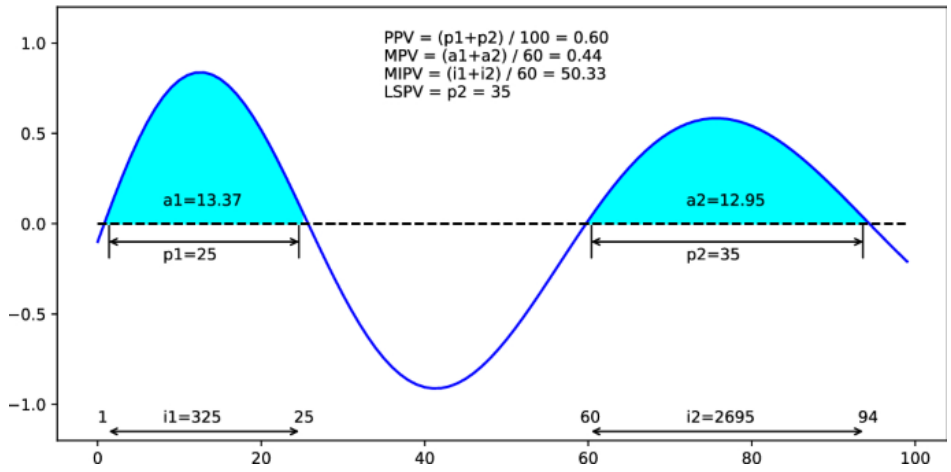


Fig. 5: This figure, as explained by Tan et al. 2022, shows how the different pooling operators are calculated in MultiRocket for the same resulting feature. The positive values are represented by $p1$ and $p2$, the sum of these values is $a1$ and $a2$, and sum of indices for those values is $i1$ and $i2$.

$$MIPV(Z) = \begin{cases} \frac{1}{m}\sum_{j=1}^{m} i_j^+, & \text{if } m > 0 \\ -1, & \text{Otherwise} \end{cases} \tag{6}$$

The LSPV, as the name suggests, aims to find the longest positive subsequence in the convolution output, and it can be represented using the following equation 7.

$$LSPV(z) = max[j - i \mid \forall_{i \leq k \leq j}\ z_k > 0] \tag{7}$$

For classifiers, they use the same two classifiers as ROCKET and MiniRocket, logistic regression trained with Adam for larger datasets and a ridge regression classifier.

Finally, MultiRocket is slower than MiniRocket but is considered faster than the other SOTA. It takes around 40 minutes on AMD EPYC 7702 CPU with a single thread, while if it uses 32 threads, the time is reduced to 5 minutes for all 109 datasets. For the same hardware, MiniRocket was able to complete the whole 109 datasets under 4 minutes for a single thread, while for the other case, it was less than 3 minutes. So, the authors have recommended using MultiRocket in multi-threaded mode.

3.2.2 Tasks

After we had studied ROCKET and its different variances, we had to do multiple tasks using ROCKET. First, we took its implementation from its official repository[7] and tested it to verify that we had done the setup correctly.

Our Main task was to test the dilation in ROCKET, which is considered one of its most important parts. The task was, instead of having the default random dilations as explained

[7] https://github.com/angus924/rocket

Tab. 1: Comparison between normal dilation and no dilation for ROCKET. The results are obtained after 5 runs.

Dataset	Normal accuracy	No dilation accuracy
SmoothSubspace	0.958667	0.473333333
SyntheticControl	0.999333	0.347333333
BeetleFly	0.88	0.5
BirdChicken	0.85	0.5
Coffee	1	0.535714286
UMD	0.977778	0.333333333
Meat	0.936667	0.333333333
ItalyPowerDemand	0.9724	0.501652089
Chinatown	0.978426	0.725947522
ECG200	0.852	0.622
DistalPhalanxTW	0.702158	0.097841727
MiddlePhalanxTW	0.537662	0.281818182
Fungi	0.997849	0.037634409
Wafer	0.99867	0.401427644

above, to do some control over it and to try to minimise the randomness.
The first part was to compare having normal dilation and having no dilation at all, shown in table 1. As the results show, the existence of the dilation element is a crucial part of ROCKET; without it, the accuracy declines. Those results are obtained as the mean after 5 runs. We want to add that after we tested on those 14 datasets, we ended this experiment because it was clear that the existence of the dilation is crucial.

Also, another test we did was the effect of increasing the number of kernels with no dilation compared to normal dilation. We did this test from 10000 kernels to 20000 kernels with a step of 100 kernels each time. Figure 6 shows the effect for the SmoothSubspace dataset.

After that, we wanted to do another test, this time to see the effect of having a constant dilation on the overall accuracy in ROCKET, but we faced multiple problems. First, the dilation itself depends on the length of the kernel, so we had two options: either we limit the kernel itself from the main function into a single value. However, it will cause multiple problems because it will also affect other elements, and we want only to study the dilation. The other option that we did was to fix the value of the kernel inside the dilation equation and do three different runs with each value of the kernel while keeping the kernel values random for other elements. The second problem was the fact that dilation is sampled from an exponential scale, so we overcame this problem by just creating a list that will contain all the possible values from 2^0 up to 2^A where A is the maximum value that can be reached by the equation used to generate the dilations in ROCKET. After we had the list, we passed the values to a modified version of ROCKET, which also takes dilation as part of its argument.

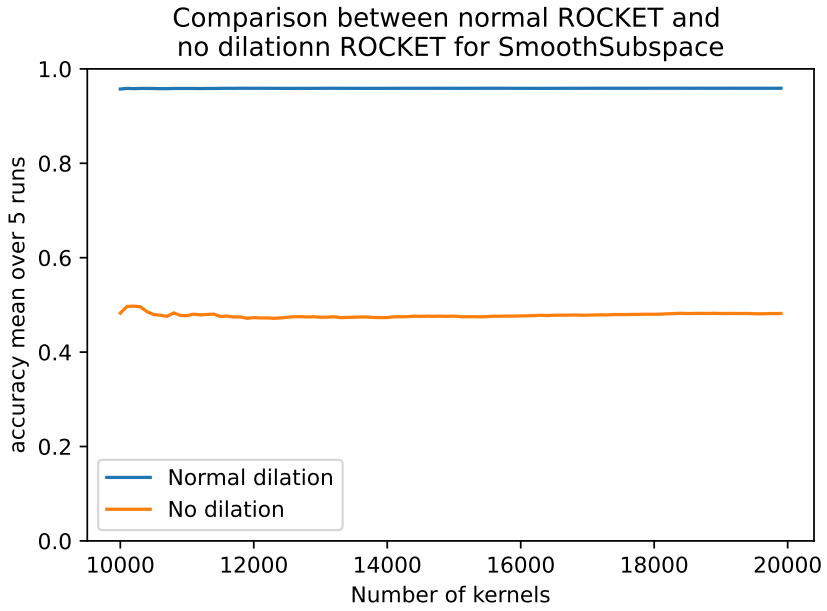


Fig. 6: This is the comparison between normal dilation and no dilation for ROCKET with the effect of increasing the kernels.

We did our tests initially on 37 datasets, but we had to exclude three of them because their length is short, like in SmoothSubspace, where it had created a single value for dilation. So, we ended up doing our tests with 34 datasets. Each test is averaged after 5 runs.

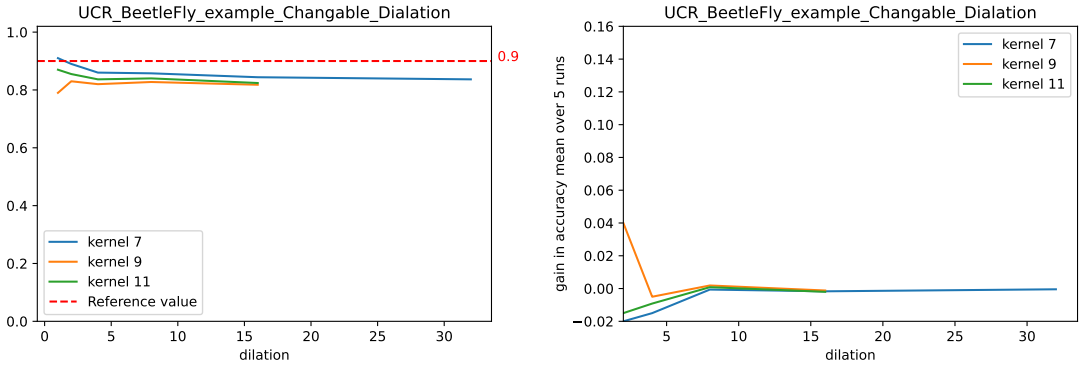
Also, the last thing that we had to do in ROCKET, after we those accuracies, was to draw the rate of change of those accuracies, and figure 7 is an example for BeetleFly dataset, that shows the value of accuracies with changeable dilation in figure 7a while in figure 7b it shows the rate of change of these values.

What we have noted was the following:

1. Most of the time, the accuracies are below the reported values.
2. Some experiments gave accuracies that were close to the reported values, sometimes nearly identical to them.
3. Some experiments gave accuracies better than the reported values.
4. The gain always converges at zero.
5. In some datasets, where the dilation is small, it has the biggest effect on the rate of change for the dilation.

3.3 Jigsaw:

The final period of the internship was focused on unsupervised learning using data augmentation techniques. It was for the remaining of the internship, and it was the longest part of the work.



(a) BeetleFly changeable dilation

(b) BeetleFly rate of change of dilation

Fig. 7: The changeable dilation for the BeetleFly dataset with the corresponding gain in accuracy.

3.3.1 Introduction

Data augmentation techniques are used to increase the generalisation of the data, and it is usually needed because the data in a dataset does not reflect the diversity and generalisation of real-world data. Also, the dataset itself could have problems like class imbalance, or it could be even so small as explained by (Iwana et al. 2021).
So, multiple researches were done for different ways used for data augmentation. However, unlike the data augmentation techniques used for images, for example, flipping, cropping and scaling. For time series, it is more complex, so that is the reason that there is a lot less research done on it. In fact, to our knowledge, there were a few surveys that were done for all the different techniques used for data augmentation in time series (Iwana et al. 2021; Wen et al. 2020; Talavera et al. 2022), which implies that there is now more focused research on this topic.

Most of the research done was on different ways like flipping, scaling, Fourier transform methods, and so on. However, other research focused on using autoencoders and Generative Adversarial Networks -GANs- as explained by(Iwana et al. 2021; Talavera et al. 2022). So for the internship, and because most work done on TSC in deep learning was inspired by its counterparts from the image domain, the main area of the research was to try to adapt from an image technique used for data augmentation called Jigsaw puzzle (Noroozi et al. 2016) which we are going to explain in brief in the following section.

3.3.2 Jigsaw in images:

In images it is a technique of unsupervised learning they first resize the training images to be 256 pixel either in height or width and then they extract a random area of size 225 x 225 which is then split into a grid of 3x3 each grid has a size of 75 x 75 pixels from each of those grids they take an area of size 64 x 64 pixels and the rest of the 11 pixels are going to

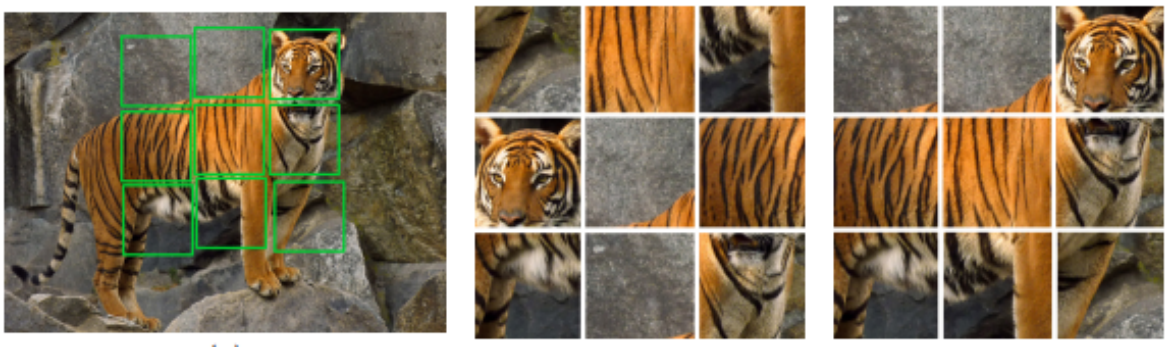


Fig. 8: Jigsaw in images as shown by Noroozi et al. 2016.

be gaps -shown in image 8 (Noroozi et al. 2016)- the purpose of these gaps is to make the known pattern like edge continuity difficult for the model to detect and they shuffle the order of the images in the order to be given to an encoder with 9 parallel layers and they repeat the same process for the same image to try and take different areas of it after finishing the preprocessing they take an average of 69 different permutations for the same image and feed it to the model in order to be able to generalise as possible and they shuffle the order of the images in order to prevent the encoder from misinterpreting a specific pattern.

This was a brief explanation of how the Jigsaw works with images, so we took the general ideas and tried to implement and adapt them to be able to be used in Time Series Classification.

3.3.3 Model Phases:

There were two main phases for our approach that we did: The first phase is training an FCN autoencoder to be able to reconstruct the original data after being permuted in order to be able to gain insight into the data and have more information.

The second phase is after training the autoencoder, we extract the encoder part and fine tune it to be able to classify the permuted data into their corresponding classes.

Those were the main phases of our approaches. However, we could not access the server to do our work due to some unforeseen events, so we had to work on the limited capabilities of our setup. So, we were unable to finish all sets of tests, but we did a lot of them. And there were different approaches that we did throughout the internship, which we are going to explain in detail later. For the final approach, we did exactly a total number of 1664 tests for the autoencoder part and around 3094 tests for the fine-tuning part.
For our final approach, we tested on four datasets: SmoothSubspace, UMD, ECG200, and Beef. They are available in the UCR archive (Dau et al. 2019). The reason we made our tests on only those 4 was to be able to do many tests with different parameters because, in our first approach, we did it on 34 datasets, which took much time to obtain results for a single test.

3.3.4 Background:

During our different approaches, we faced many problems, and it was mainly in how to adapt a technique that is used for images into being compatible to be used for TSC. So, here we are going to talk about our final approaches, along with the alternatives, if they existed. Those problems were more focused on the autoencoder phase than the fine-tuning phase.

3.3.5 Autoencoder:

When we worked on this part, we faced many problems when we tried to adapt it, and we were unable to adapt all aspects of the image approach into a time series approach. We even decided to add a new aspect, which is the bottleneck layer.

The aspect that we were unable to use in our final approach was adding gaps to our permuted data, and the following are the aspects that we used:

1. Number of segments: This is the number of segments that we divide our time series into.
2. Permutation list: This is the list that we use to generate different permutations.
3. Number of copies -permutations- to feed for our model: We also give our model a multiple number of different permutations for the same time series to be able to generalise it as possible.
4. Bottleneck: We also test the effect of adding a bottleneck in the autoencoder between the encoder and the decoder part.

3.3.6 Number of segments:

This was our first problem: How many sub-series should our series be divided into? The image approach was simple; they took 9 images of 64 pixels each, and it was a fixed number of 9, as we explained earlier, but for time series, we cannot simply resize them because each dataset has its own length. So, we decided to divide the series into multiple numbers of segments and this number to be our first variable.

The second problem was how to divide them. We had two options: either being divided into a fixed number of segments regardless of the length of the series or having the length of the series divided by a fixed number so we can have a variable number of segments that depends on the series length. We chose the first option because at the time of testing, which was during one of our earliest approaches, generating a permutation list was time-consuming and even impossible at that time. For example, if the length was 200 and we decided to divide by 4, we would have 50 segments, meaning we have to generate a list with different permutations for those 50 segments, while for the other case, we will have only 4 segments. For both cases, when fractions occur, we take only the integer value.

The third problem was how to divide them properly. There were two possibilities:

1. To make all segments of equal length with the exception of the last part, which will either be of the same length or bigger by 1 to $n-1$ where n is the number of segments we want it to be divided into.

2. To have two sets of segments length with the segments that are available in the first set have a bigger length by 1, and we mean by that if we have a series of length 18 and we want to divide it into 5 segments, the first 3 segments will be part of the first set of length 4, while the last 2 segments will be part of the set of length 3.

At that time, there was no big difference for us, so we chose the first option because it was simpler, and we did not think of extreme cases like the following: if the length was 18 and 5 segments, all 4 segments will be of length 3 while the last one will be of length 6.

The fourth problem is what happens after we divide them and permute their order. Should we merge them together or leave them? We have found it to be more of an architecture approach because in the image version, in the encoder part, they had 9 parallel layers each for an image. After the encoder, those parallel layers are merged together, and they have called this network context-free network. For our case, the parallel layers are going to depend on the number of segments. The other option was to merge them and use a simple FCN encoder. We decided to use the simple solution because it was faster and had been used before. We also had a plan to test the other architecture, but due to time and our resources, we were not able to test it.
In the last part of our final approach, we decided to test with the number of segments varying from 2 to 40 and in case the number of segments exceeded the length of the time series, we changed the segments to be of length 1.

3.3.7 Permutation list:

This was the second point that we adopted from the image jigsaw. In the image, they have a 9! possibilities, so they exclude those possibilities using different techniques like using the hamming distance between these images and if they found them in a particular permutation to be either so close or so far they exclude this permutation, and other methods to finish with an average of 69 out of a 1000 possible configuration. For time series, we have faced multiple problems until we were able to reach our final algorithm.

We thought to use DTW at first like they used the hamming distance, but we rejected it because of its time complexity, especially if long series were going to be used or if we were going to have a very large number of segments. So, we thought of another approach.

Our approach was generating a permutation list, which is a list of lists. Each one of those lists is a different permutation that depends on the number of segments we want the program to create. After we have this list, the program chooses a permutation from the list

Fig. 9: This is a screenshot that shows the difference in execution time between algorithm 1 and algorithm 2 for a number of segments =15. As seen for algorithm 1, it takes around 36 minutes, while for algorithm 2, it takes less than a second.

randomly for each series being permuted.

At first, the numbers were small so that we would generate all possible permutations for them, excluding the original case, but as the numbers got bigger, the time got longer until the system began to crash.

So, it became apparent we should search for another way, so we decided, instead of taking the whole set of possibilities, to take only a subset of them. So, what we did was starting from the number of segments = 4 instead of having all 23 possibilities, we will have $number of segments \times 5$, and we decided to commence from 4 to have only 20 possibilities while for 5 we will have only 25 instead of 119.

This approach was fine at the start, but after testing different approaches, we noticed that the reconstruction for the series in the autoencoder was heavily dependent on the permutation lists, and it becomes apparent when the number of segments is bigger than 10, there is a higher probability that the reconstructed series is completely lost.

So, we thought of a way to control the permutations, so we decided to implement a condition that no element can go further from its original position by more than 3 indexes so the n element can occur in any position from n-3 to n+3 in order not to make permutations totally random which could lead to data loss, so we did 2 implementations for it.

The first implementation was to generate random permutations. After each permutation is generated, the program checks the whole list, and the moment it finds that the condition was broken, it rejects this permutation. This implementation worked fine until the number of segments was more than 15, the program became slow, and it would even take up to one hour to generate all possible permutations for the list.

Because of this, we implemented our second algorithm, which is an optimised version that has literally reduced the time from hours to a few seconds. Figure 9 shows the difference in time. This algorithm has 2 lists. The first is an empty list called lis_genrated that will be filled with numbers to generate the permutation, and the second one will be filled from 0 to n-1, where n is the number of segments. This is going to be a reference list. Each time a number is inserted inside the lis_genrated list, this number is going to be removed from the reference list. The detailed algorithm will be found in Appendix A.

After doing some tests and calculating time, we had the structure of the final conditions

that we used to generate the permutation list:

1. If the number of segments was 2 or 3, we generate all possible permutations.
2. If the number of segments was 4 or 5, we use the first algorithm.
3. If the number of segments was 6 or higher, we use the optimised algorithm.

The reason that we used the old algorithm for 4 and 5 was that after testing, it showed to be faster than in the optimised version, so we kept it only for those two cases.
Also, each permutation in the list is unique, and no repetition is allowed. While we made the number to be 3 indexes in order to control the generation of the permutations, this number needs to be investigated further and to be another parameter that can be changed until the optimal condition is found.

3.3.8 Number of copies:

As our third point adopted from the image approach, how many permutations should we give our model? In images, they were taking an average of 69 permutations, and those permutations were chosen according to the hamming distance between the extracted 64 pixels images, in a condition that those permutations are not too far from each other for the information to be lost and also not to be to close so so the information becomes misinterpreted.

At first, we were just giving the model a single permutation for each series, and it was working, but the series reconstruction was not accurate. Sometimes, it was just a complete noise with no resemblances to the original series.

So we decided to see the effect of having multiple permutations for the same series being fed to the model -shown in figure 10 it is an example for the same series that has 3 different permutations-, in the first tests, we would give the model the same number of copies as the number of segments being used. It was working and even showed promising results, as shown in figure 11.

After that, we wanted to test the effect of having any number of copies, not just the same number as the number of segments, so we did our tests, and in our final approach, we used the number of copies to be from 1 up to the same number of segments.

We want to note that we wanted to see the effect of increasing the number of copies to be more the number of segments, what would that give us, but unfortunately, due to our capabilities, we were unable to see the effect.

Nonetheless, we have implemented it to be possible to use any number of copies. For each series, the program randomly draws different permutations from the permutations list to form a set that is usually between 2 and up to $number of segments \times 5$ with the exception of number of segments = 2 and 3. The program also augments data to be the same number of copies by copying each series in a dataset to be the same number as the requested number of

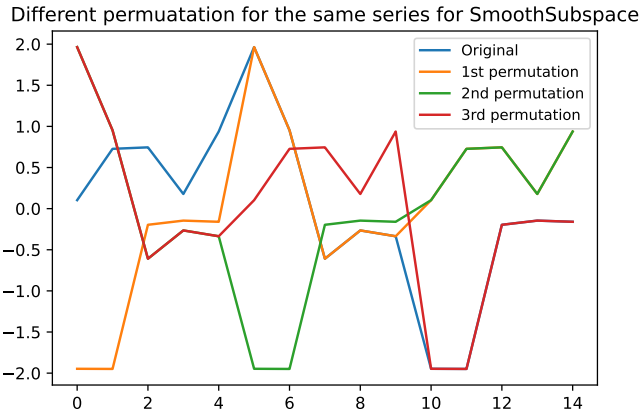


Fig. 10: In this example, the original series, which is in blue, is divided into 3 segments. Then, we have created 3 different permutations out of those 3 segments, which are fed into the model.

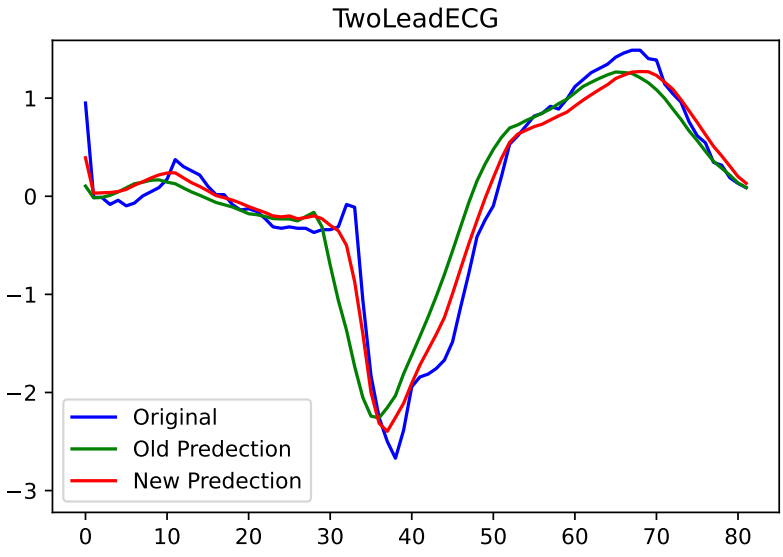


Fig. 11: This is the result that we have obtained from our first model, and it is from the dataset TwoLeadECG2. Here, the old prediction refers to a single permutation, while the new prediction refers to three permutations.

copies. We have implemented it in such a way that even if it were requested to have a number of copies higher than available in the set, the program would continue. After reaching the end of the set, wrap around and start the set again from the beginning.

Also, we would like to add that we also call it 'number of permutations' interchangeably because it shows how many different permutations for a single series were given for the model.

3.3.9 Bottleneck:

The last part of our contribution in the autoencoder part was the effect of having a bottleneck layer, as to our knowledge, this layer was not used in the image approach (Noroozi et al. 2016; Carlucci et al. n.d.; Chen et al. 2021), so we can say that this part is the addition of using time series classification techniques.
So, in the final approach, which is also our final architecture, we test the effect of adding or removing a bottleneck layer to the autoencoder between the encoder and decoder part to see its effect and which one is better. Moreover, from the initial results, not using a bottleneck was better because the reconstructed images showed it to be better with no bottleneck. That is why in our tests, we wanted to finish first, not having a bottleneck, and after we finished it, we tested the effect of having it, but we could not do extensive research on this point due to the reasons we mentioned earlier.

3.3.10 Gaps:

This was the last part we tried to adapt from the image version, but it failed. In images, the gaps had the purpose of stopping the model from learning obvious patterns, like edge continuity and colour patterns. So they have decided that out of the 75 pixels, they take 64 pixels for images while the remaining 11 pixels for the gap for a single grid, and in the general case, the gap between two images can vary between 0 up to 22 pixels. This was the general idea that we tried to adapt to our Jigsaw.
We had the following idea in mind:

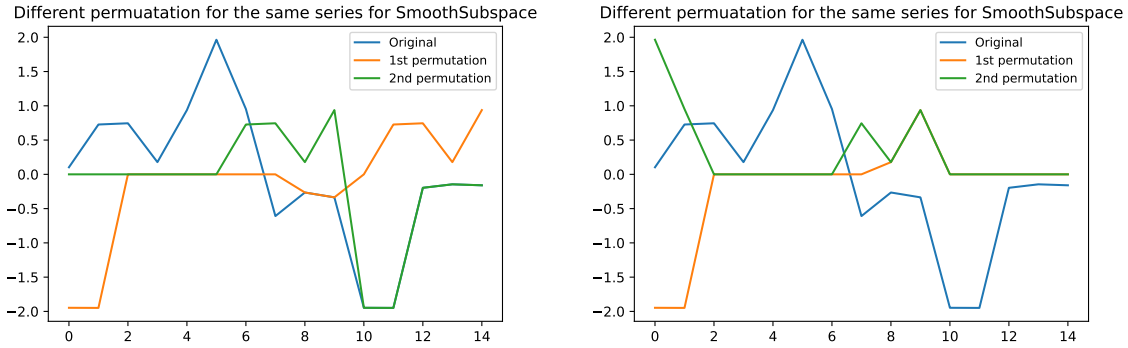
1. First, we find the length of a single segment, and we divide this value by two. Furthermore, in the case of fractions, we take only the integer value.

2. Second, we generate a list of all possible gap permutations. However, unlike the permutation list that has multiple permutations for a single number of segments, our gap lists function differently:

   (a) Each element in the list is a pair of values, the purpose of being a pair is that the first element signifies the beginning of the segment, while the second is the end.

   (b) Also, there is a condition that the sum of the pair must be equal to or less than the value we calculated in the beginning.

   (c) Using this condition, the program calculates half of the possible permutations and then mirrors them to save time.



(a) gap with a small similarity

(b) gap with no similarity

Fig. 12: The effect of having a gap in Jigsaw for time series.

(d) Finally, the list is shuffled to ensure randomness.

This was how our gap list was generated.

3. Third, when the gap is set to true to be applied, the program picks a gap permutation -a pair- at random, and when the program applies it, the following happens, depending on the value that exists for the first element, it results that the corresponding indexes for the segment to be swapped from their original values, into zeros, and the same happens at the end. The following figure shows the gap being applied to a series Figure 12.

Those were the steps that we did to be able to have a gap feature, but when we tested them in the autoencoder. The results were less than we expected, the reconstructed series had little similarity with the original 12a, and sometimes there is no similarity with the original series at all 12b. Even when we decided to test those results using the FCN classifier, the accuracy was varying. For example, in SmoothSubspace, sometimes the accuracy was around 50%, and other times it was near 1%. This implies that the Jigsaw for series could use gaps, but its parameters need further investigation, especially what percentage we should take from the segment to have gaps. In our initial approach, it was approximately 50%, so it requires further investigation.

That is the reason that we decided to discard the usage of this feature to be able to focus on the others that were more important to us.

3.3.11 Number of tests:

In total, there were many tests to be done for every single dataset. For each one of those tests, we trained a specific model. Unfortunately, we were unable to finish them all due to our available capabilities at that time, and the total test can be considered as follows:

1. There are 39 possibilities for the number of segments.

2. This number is increased to 818 possibilities when we include the number of copies.

3. It can also be increased to 1636 when we include the bottleneck conditions.

We were able to accomplish a few of those tests. We did this in the following order:

1. First, we created models from 2 segments to 40 segments, with a condition of only 1 permutation, and we used both cases for the bottleneck.

2. Second, we created all the models from 3 to 25 segments, with all other remaining permutations. However, we used only the condition of no bottleneck because it showed better potential.

3. Finally, due to time and capabilities, we created all remaining permutations for 40 segments and no bottleneck to be able to study the results.

So, in total, we were able to do exactly 416 models out of 1636 per dataset. For our four datasets, we have done exactly 1664 models.

3.3.12 Example:

This is a small example of how our autoencoder is going to function in the final approach:

If we select the number of segments = 5, the number of copies = 3, and no bottleneck for a dataset of length = 24, the following will happen:

1. First, the program will divide it into 5 segments with the following: from segments 1 to 4, it will have a length of 4 for each segment, while the last segment will have a length of 9 indexes. However, if the length for the series in the dataset was 26, the first 4 segments would have a length of 5 indexes, and the last segment would have a length of 6 indexes.

2. Second, in the creation of permutation list, the initial possibilities for 5 segments is 119 after removing the repetitive case, we will take only 25 of those cases and those 25 with the condition mentioned earlier, so for the case [3,2,4,1,0] it will be rejected because of the position of 0.

3. Third, in the number of copies, for each time series in the dataset, it will have 3 different permutations taken randomly from the list, which is going to be applied to the series when we merge those segments again. However, in a different order, and if we had 100 samples in our train set, we would now have 300 samples.

4. Finally, we have specified no bottleneck, so when the program builds a model, it will not add any bottleneck layer to the architecture.

This was how our Jigsaw works briefly for the autoencoder part.

3.3.13 Fine Tuning:

The second major part of our Jigsaw is the fine-tuning. This part was simpler, and we faced a few problems in it, but there were a few problems that were educational to us.

For the fine-tuning part, we first extracted the encoder part from the autoencoder. Then we added two layers to the encoder, first a GAP layer followed by a dense layer to transform it into a classifier, and then we fine-tuned the whole model. When we fine-tuned the model, it had the same parameters as the autoencoder, but we made a small change. For the number of segments, it will be the same, but for the number of copies, we only give the model a single permutation, and the reason is that it would take a lot of time, which, at this moment, we did not have a lot of it.

Also, we have decided to see the effect of having to freeze the encoder layers and to train the additional layers to fine-tune the model versus if we did not use any layer freezing and fine-tune the whole model.

One of the problems that we faced at first was the extraction of the layers. In the early stages of fine-tuning, we discovered that the code of extraction was not working properly, and it was just adding layers from the decoder layers to the extracted encoder, which caused the accuracy to decline.

After we fixed it, we discovered that this fix works well on the architecture that does not have a bottleneck, while for the architecture that has a bottleneck, it extracts the model with the bottleneck. So, we had to do another fix, and in the end, it worked fine.

3.3.14 Architectures:

We have done many architectures, and all have the basic idea of an encoder followed by a decoder to reconstruct the series. We have taken the idea to use an FCN encoder from (Ismail-Fawaz et al. 2023), which was a paper that focused on self-supervised learning using deep learning, but for their paper, they did not build a decoder, which was our task to do. We also had some ideas from (Fawaz et al. 2018), in the way they kept the same parameters for both approaches, with and without transfer learning, So we decided to do the same for the autoencoder and fine-tuning parts.

For the first architecture, we have used the following layers:

1. The first three layers are normal FCN encoders. They are three blocks, and each block has the same operation as it was mentioned in 1. The only difference is that the size of kernels is the following = $[5, 3, 3]$ and the number of filters = $[32, 64, 128]$. Other than that, they have the same activation function, ReLU, and for padding, we use the option 'same'.

2. After the encoder, the first main characteristics of the decoder were defined here. We have also used three blocks. Each block had the following: the first 1D up-sampling layer, followed by a 1D convolution layer, followed by a batch normalisation layer, and the last layer is an activation layer using the ReLU function.

This was our initial architecture, but due to some dimension error, we had to add a GAP layer after the decoder to fix this problem. Followed by a linear activation function. We used Adam for optimisation with a learning rate of 0.001, and for the loss, we used mean squared error. For batch size, it was the minimum between the number of training samples/10 and 16.

We have also tested the effect of having a bottleneck layer in this architecture. This layer consisted of convolution 1D with kernel = 1 and filter = 1, followed by batch normalisation, followed by ReLU activation function.

We tested this architecture on 36 datasets. Surprisingly, this architecture was able to reconstruct series similar to the original ones. However, we disregarded this architecture because when we created it, we used the methods of software development to just fix the error without understanding the meaning behind the solution. And this error was so beneficial to us because it gave us the opportunity to learn from the scientific method.
After this architecture, we have tried to do a number of different tests to be able to have the original accuracy. Moreover, to fix errors that we did in the previous one, like the number of kernels and features in the decoder, which must be the reverse of the ones available in the encoder.

Also, it was because of this mistake that we decided to do the test on only 4 datasets:

1. Beef
2. ECG200
3. SmoothSubspace
4. UMD

The reason is to be able to do multiple tests to have an accuracy near the first architecture.

In the second architecture, we fixed some of the issues:

1. The first three layers were the same, but we have changed the number of kernels to be $= [8, 5, 3]$ and the number of filters $= [128, 256, 128]$.
2. For the decoder, we replaced both layers of the 1D up-sampling layer, followed by the 1D convolution layer, to a single layer of Convolution 1D Transpose, which helped in resolving the error we had before, and the number of kernels $= [3, 5, 8]$ and the number of filters $= [128, 256, 128]$, so we disregarded the use of GAP layer to fix the dimension problem.

This was the second architecture. The only other change was batch size = 64. This architecture has produced less accurate results compared to the previous one.

For the final architecture, we had the same main architecture as the second. The only change was for the encoder in the number of kernels and the features. the number of kernels = $[40, 20, 10]$ and the number of filters = $[32, 64, 32]$. Furthermore, for the decoder, it was the opposite.

After that, we also decided to re-add the bottleneck layer to do the different tests for our method.

3.3.15 Comparison Criteria:

We used multiple ways to compare the reconstructed series and accuracies between the models that we did, and they are as follows:

1. RMSE
2. DTW
3. Accuracy

We used the first two to compare the reconstructed series in the autoencoder part, while the accuracy is used for the fine-tuning part.

RMSE:

The RMSE stands for Root Mean Square Error. We use it to compare the original data and its prediction for the autoencoder part, and we get the average for each autoencoder.

DTW:

We used it to make multiple comparisons between the original data and its prediction for the autoencoder part, and they are as follows:

1. Find the best and worst representation between the results of each autoencoder and the original data overall classes.
2. We also find the best and worst representation, but this time per class, to know whether certain patterns were generalised.
3. Lastly, we averaged all the distances to know which autoencoder parameters were able to have a near representation for the original datasets.

Accuracy:

After we extract the encoder part from the autoencoder, we fine-tune it, and then we find the average of 5 runs of accuracy to be our scoring criteria.

3.3.16 Results:

From what we have up to this moment, there were some notes that we were able to notice

Autoencoder:

First, for the autoencoder:

1. Almost all the results for no bottleneck layers were better than having a bottleneck layer for the condition of only one single permutation for each specific segment number. However, we need to mention that we were more focused on no bottleneck layer in our research because those primary results made us focus on doing our research on no bottleneck layers, which means that there is a possibility that after a specific number of permutations, the bottleneck layer could have better results.
2. What we have noticed is that when we increase the number of segments and/or number of permutations corresponding to a specific number of segments, the similarity of the reconstructed series becomes more accurate to the original one, with the exception of the number of segments equal to 2 because it has a high similarity with the original. We can see it already had the best score in two datasets, and we suspect the reason is that with only 2 segmentation and only one possible permutation, the autoencoder does not face many problems, so it can easily reconstruct the original series easily.
3. Also, the worst representations in DTW when we increase the number of segments and/or number of permutations become less distorted and have more resemblances to the original series.
4. From what we have always noticed, the tests with only a 1 permutation have the highest RMSE and average DTW. Then these values start to decrease with increasing the number of permutations, with the exception of segment 2 because segment 2 has the lowest values in two datasets, which are SmoothSubspace and ECG200, and we suspect the reasons to be the same as mentioned in 2.
5. Sometimes, a certain pattern is being generalised between classes, but as the number of segments and/or number of permutations increases, this generalisation becomes less frequent

For the results, both RMSE and DTW got the same best results, but there was a difference for the worst in some datasets where they got different results.

Fine Tuning:
For the second part, the results are obtained after taking the average of 5 times:

1. The accuracy of segmentation 2 is the highest in most cases, even with different parameters, and it is because of the reasons we mentioned earlier
2. The best scores we got so far are for 2 segments with no bottleneck and no freeze
3. When we use the freeze, some results have a standard deviation of 0, and we suspect the reason is due to the training of only the two layers that were added to the encoder
4. When we use freeze, it also has some of the worst values
5. We found that the parameters with the best scores were when we did not use any bottlenecks and there was no freeze. However, those results are inconclusive because we need more tests to confirm our findings.

Tab. 2: Autoencoder: Score for 2 segments, 1 permutation and no bottleneck

	RMSE	Average DTW	Best DTW	Worst DTW	Best Overall
Beef	0.35	39.87	12.23	192.56	No
ECG200	0.2	9.49	4.64	32.89	Yes
SmoothSubspace	0.26	3.02	1.75	6.23	Yes
UMD	0.51	24.32	3.78	68.86	No

Tab. 3: Fine tuning: Score for 2 segments only

	No Bottleneck No Freeze	No Bottleneck Freeze	Bottleneck No Freeze	Bottleneck Freeze
Beef	0.71	0.35	0.41	0.35
ECG200	0.83	0.76	0.94	0.78
SmoothSubspace	0.94	0.66	0.8	0.73
UMD	0.93	0.81	0.6	0.73

6. In Beef as shown in Figure 13a, it looks like the accuracy for all different parameters will end at a convergence around 0.4

7. In ECG200, as shown in Figure 13b, it concludes that the accuracy converges at a certain value according to the parameters used

8. In SmoothSubspace as shown in Figure 13c, there is a great decrease in accuracy after 2 segments, and it continues to decrease until around 5 segments, then it increases again until around 9 segments then it converges around a specific value with the exception of the case Bottleneck and no freeze which requires further studying

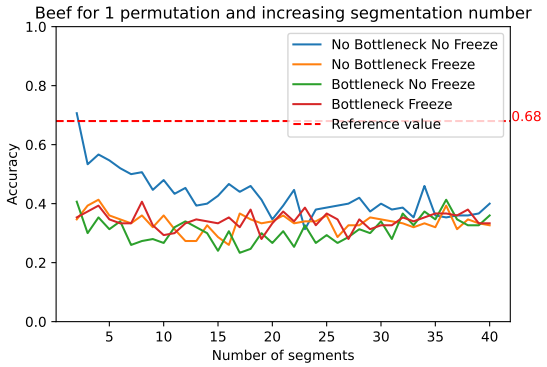
9. In UMD as shown in image Figure 13d, the accuracy values first decrease, then at the end, they increase, which implies that with the increase of the number of segments, it could become better

10. In Figure 14, when we had a constant number of segments = 40 and an increasing number of permutations, there was a small improvement, and we noticed that the accuracy converged towards a certain value.

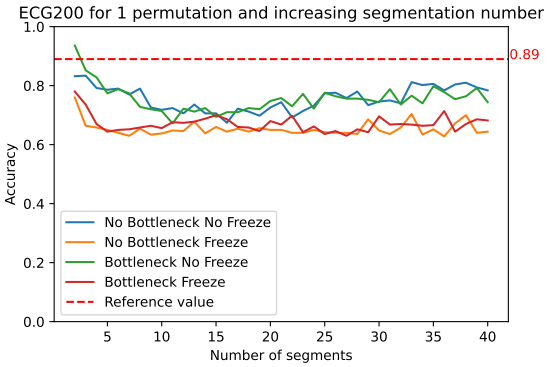
11. When we compared the results in Figure 15, it showed that with an increasing number of both permutations and segments, the accuracy is better than just having a single permutation and an increasing number of segments.

Now, these tables summarise the best and worst scores we have obtained so far for the autoencoders and the fine-tuning:

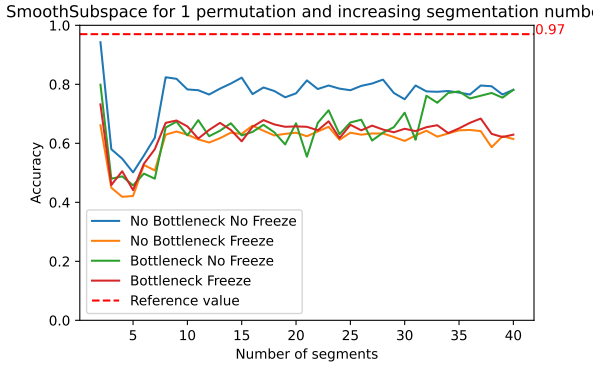
We also decided to include segmentation 2 in a single table 2 and 3 because of the reasons that we mentioned earlier.



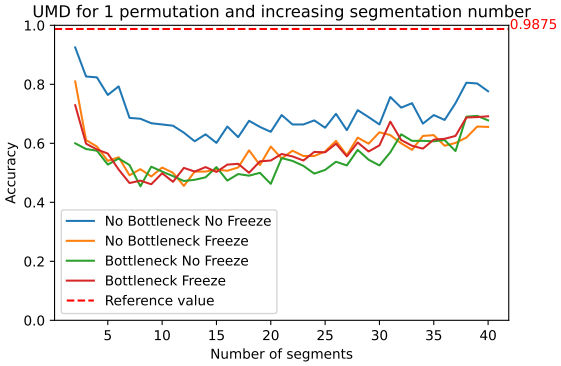
(a) Beef Dataset



(b) ECG200 Dataset



(c) SmoothSubspace Dataset

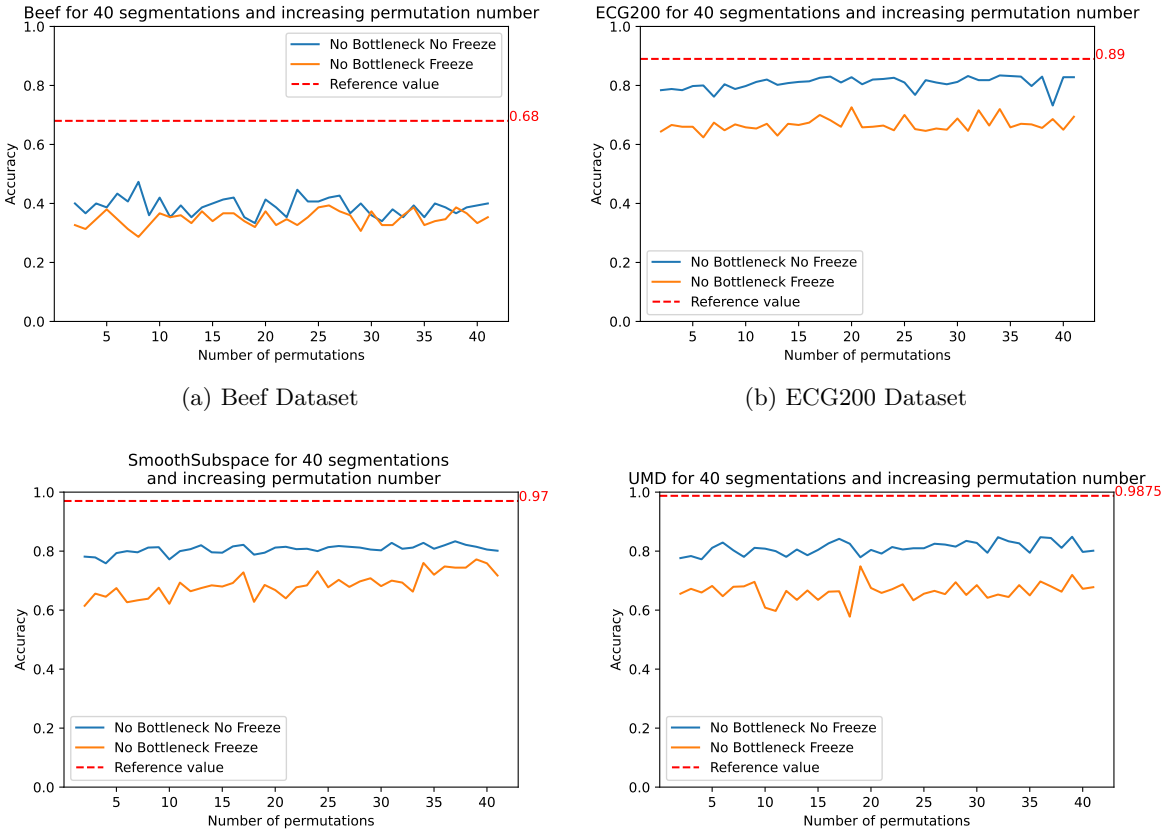


(d) UMD Dataset

Fig. 13: Accuracy of datasets with an increasing number of segments and 1 single permutation.

Tab. 4: Fine tuning: Score for the best values excluding 2 segments

	No Bottleneck No Freeze	No Bottleneck Freeze	Bottleneck No Freeze	Bottleneck Freeze
Beef	0.58	0.43	0.41	0.41
	4 seg 2 per	4 seg 2 per	36 seg 1 per	8 seg 1 per
ECG200	0.854	0.704	0.852	0.736
	3 seg 3 per	33 seg 1 per	3 seg 1 per	3 seg 1 per
SmoothSubspace	0.85	0.7	0.78	0.684
	9 seg 2 per	12 seg 12 per	40 seg 1 per	37 seg 1 per
UMD	0.85	0.66	0.69	0.69
	3 seg 2 per	39 seg 1 per	39 seg 1 per	40 seg 1 per



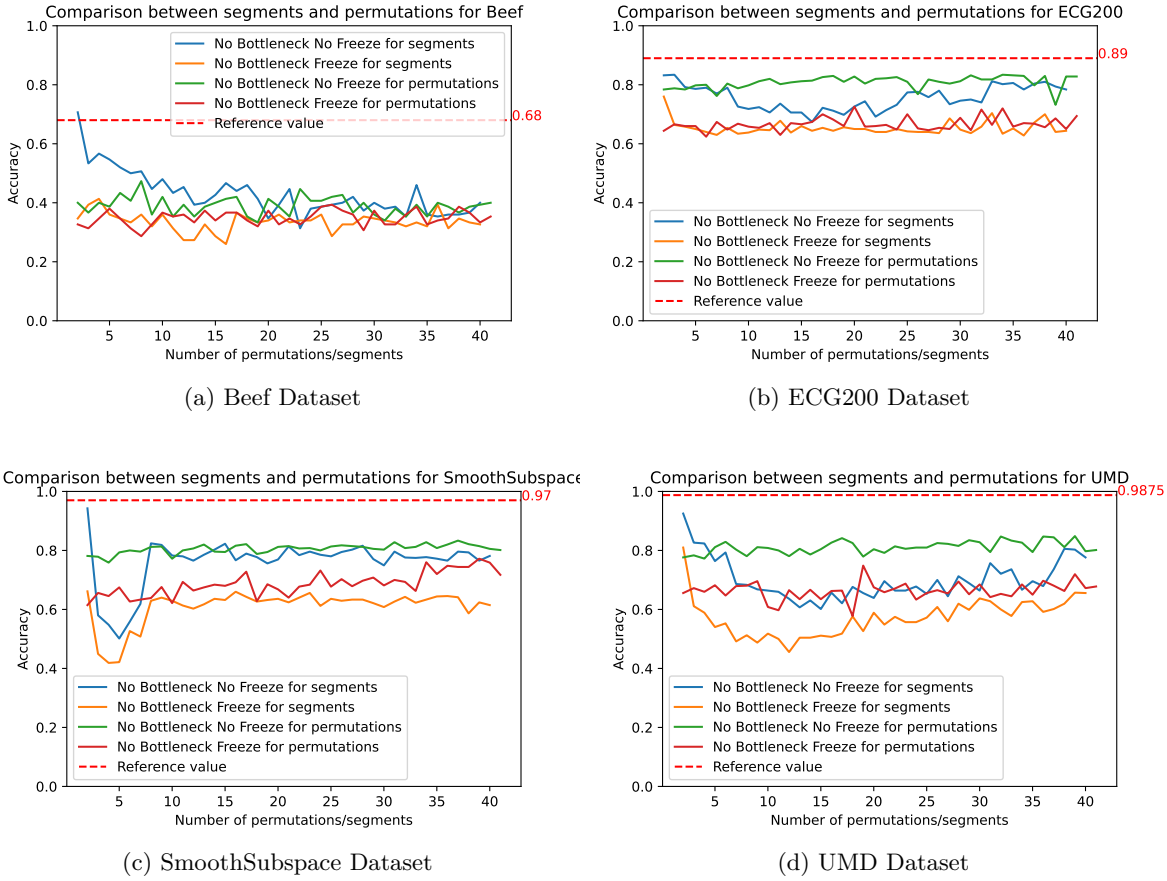
(a) Beef Dataset

(b) ECG200 Dataset

(c) SmoothSubspace Dataset

(d) UMD Dataset

Fig. 14: Accuracy of datasets with an increasing number of permutations and number of segments = 40.



(a) Beef Dataset

(b) ECG200 Dataset

(c) SmoothSubspace Dataset

(d) UMD Dataset

Fig. 15: Comparison between having a constant permutation number with an increasing number of segments -Blue and Yellow- and having a constant number of segments with an increasing permutation number -Green and Red-.

4 Conclusion:

This internship was so beneficial and insightful to me. I have gained a lot of information and experiences, and it has also helped me in deciding to continue my studies in doing a PhD in the future.
The following are some of the experiences that this internship has helped me with:

1. How to correctly read papers and to be able to extract useful information from them.
2. How to make my work more organised and understandable by others.
3. How to correctly present graphs and try to find the best representation for them, especially if they were multiple graphs with different conditions, but they were for the same subject.

Those were some of the experiences that I gained, and for the technical skills, this internship has helped me improve my level in various domains and has even helped me learn new subjects:

1. It has improved my Python level and to be able to write more organised code.
2. Improved my level in using TensorFlow and how to use it more properly.
3. Improved my level and knowledge in using Linux because before, I was just able to do basic stuff on it. Now, I am more capable of using it and can use some more advanced commands in the terminal.
4. It has also introduced me to new software called LaTeX[8]. It is a software that is used to write documents, but unlike Word[9] or LibreOffice[10], this software is similar to programming languages, and it helps to write more efficient documents, and it helps to manipulate the documents more easily.
5. It also improved my French level. At first, when I came to France, I was unable to understand what was assigned to me, and when my professors were explaining whatever subject, I could not follow them. At the end of the first term, my level had improved a little, but I could not fully understand what was required of me. That is when I started my internship. Because I had to use the language more often and had multiple engages in it, I am now quite capable of following the conversations easily and being able to understand at least 50% of the conversations without struggling.
6. It has also helped me better understand time series and how to deal with them.

Those were some of the experiences that I gained from the LAB. I was also able to contribute to the lab by helping them with the following tests:

[8] https://www.latex-project.org/
[9] https://www.microsoft.com/en-ww/microsoft-365/word?market=af
[10] https://www.libreoffice.org/

1. In ROCKET:
   (a) I was able to help them do a test to compare the accuracy between having the normal ROCKET and ROCKET with no dilation.
   (b) I was also able to help them in doing a dilation test and to see the effect of having a constant dilation on the accuracy of the kernel and not depending on the random element.
   (c) Then, I helped them by doing this test on an increasing constant dilation with 3 different kernels.
   (d) After that, I was able to help visualise the effect of increasing this constant dilation on the accuracy with different kernels and whether it helped or not, depending on the dataset.
2. In Jigsaw:
   (a) I was able to help them explore a new method of unsupervised learning and adapt this method from the image domain into the time series domain.
   (b) Provided them with the general ideas that I thought about for Jigsaw for time series and the approaches that I used, which they can further utilise in-depth to have better results.
   (c) Implemented the class used to generate Jigsaw puzzles for time series, which can be used directly by them or be further improved.
   (d) Gave them the results of a few datasets, which can help them decide whether to continue using this approach, modify it a bit, or disregard it.

Those were my main contributions to the LAB.

For the results, I had multiple aims, which I was able to achieve some of them, but a lot other I was not able to achieve:

1. I was able to achieve the following:
   (a) To better understand the researcher's life.
   (b) To better understand time series.
   (c) To decide whether to do a PhD or not.

My main aim, which also was the hardest, was to be able to publish a paper during my internship. I knew that it would be a difficult challenge, and when I started doing the Jigsaw, I hoped to publish a paper on it.

After I saw how much time it took, I just hoped to be able to finish at least half of the UCR archive, but unfortunately, because of some circumstances, I was unable to access any servers to launch the experiments, which forced me to launch them on the computer.

With no access to a server, I just hoped to be able to finish all the tests of the final 4 datasets before my internships ended, but what I was able to finish was about 30% for each of those datasets.

References

Bagnall, Anthony, Hoang Anh Dau, et al. (2018). "The UEA multivariate time series classification archive, 2018". In: *arXiv preprint arXiv:1811.00075*.

Bagnall, Anthony, Jason Lines, et al. (2017). "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances". In: *Data mining and knowledge discovery* 31, pp. 606–660.

Carlucci, Fabio M et al. (n.d.). "Domain Generalization by Solving Jigsaw Puzzles-Supplementary Material". In: ().

Chen, Pengguang, Shu Liu, and Jiaya Jia (2021). "Jigsaw clustering for unsupervised visual representation learning". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11526–11535.

Dau, Hoang Anh et al. (2019). "The UCR time series archive". In: *IEEE/CAA Journal of Automatica Sinica* 6.6, pp. 1293–1305.

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.

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

Fawaz, Hassan Ismail et al. (2018). "Transfer learning for time series classification". In: *2018 IEEE international conference on big data (Big Data)*. IEEE, pp. 1367–1376.

Ismail Fawaz, Hassan, Germain Forestier, et al. (2019). "Deep learning for time series classification: a review". In: *Data mining and knowledge discovery* 33.4, pp. 917–963.

Ismail Fawaz, Hassan, Benjamin Lucas, et al. (2020). "Inceptiontime: Finding alexnet for time series classification". In: *Data Mining and Knowledge Discovery* 34.6, pp. 1936–1962.

Ismail-Fawaz, Ali et al. (2022). "Deep learning for time series classification using new hand-crafted convolution filters". In: *2022 IEEE International Conference on Big Data (Big Data)*. IEEE, pp. 972–981.

— (2023). "Enhancing Time Series Classification with Self-Supervised Learning". In.

Iwana, Brian Kenji and Seiichi Uchida (2021). "An empirical survey of data augmentation for time series classification with neural networks". In: *Plos one* 16.7, e0254841.

Lines, Jason and Anthony Bagnall (2015). "Time series classification with ensembles of elastic distance measures". In: *Data Mining and Knowledge Discovery* 29, pp. 565–592.

Lucas, Benjamin et al. (2019). "Proximity forest: an effective and scalable distance-based classifier for time series". In: *Data Mining and Knowledge Discovery* 33.3, pp. 607–635.

Middlehurst, Matthew, Patrick Schäfer, and Anthony Bagnall (2023). "Bake off redux: a review and experimental evaluation of recent time series classification algorithms". In: *arXiv preprint arXiv:2304.13029*.

Noroozi, Mehdi and Paolo Favaro (2016). "Unsupervised learning of visual representations by solving jigsaw puzzles". In: *Computer Vision–ECCV 2016: 14th European Confer-*

ence, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VI. Springer, pp. 69–84.

Ruiz, Nataniel et al. (2023). "Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22500–22510.

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.

Talavera, Edgar et al. (2022). "Data augmentation techniques in time series domain: A survey and taxonomy". In: *arXiv preprint arXiv:2206.13508*.

Tan, Chang Wei et al. (2022). "MultiRocket: multiple pooling operators and transformations for fast and effective time series classification". In: *Data Mining and Knowledge Discovery* 36.5, pp. 1623–1646.

Tavenard, Romain (2021). *An introduction to Dynamic Time Warping*. https://rtavenar.github.io/blog/dtw.html.

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

Wen, Qingsong et al. (2020). "Time series data augmentation for deep learning: A survey". In: *arXiv preprint arXiv:2002.12478*.

Zhang, Zongxi et al. (2023). "A Similarity-based Time Series Source Dataset Selection Method for Transfer Learning". In: *Journal of Physics: Conference Series*. Vol. 2428. 1. IOP Publishing, p. 012038.

A Algorithm used for optimizing list generation

Algorithm 1 Optimized algorithm for generating the list faster

```
Function Generate_List_With_Condition ()
Output: Returns a list of permutations
initialize lis_generated ← [RandomInt(0, 3)]
initialize all_elements ← list(range(0, self.segments_num))
all_elements.remove(lis_generated[0])
initialize k ← 1
while k < self.segments_num do
    if k ≤ self.segments_num - 2 then
        initialize candidate_num ← RandomInt(max(0, lis_generated[k-1]-4),
        min(self.segments_num-1, lis_generated[k-1]+4))
        if candidate_num ∉ lis_generated then
            lis_generated.append(candidate_num)
            if min(all_elements) ∉ lis_generated and (abs(min(all_elements) - k) ≥ 3) then
                lis_generated[-1] ← min(all_elements)
                candidate_num ← min(all_elements)
            end if
            all_elements.remove(candidate_num)
            k ← k + 1
        end if
        check ← CheckElement(lis_generated)
        if check ≠ True then
            lis_generated.remove(check)
            all_elements.append(check)
            k ← k - 1
        end if
    else
        lis_generated.append(all_elements[0])
        k ← k + 1
    end if
end while
return lis_generated
```

Algorithm 2 Check Element Function

```
Function CheckElement (lis)
Input: lis - List of elements
Output: Returns an element from the list or True
initialize n ← len(lis)
initialize indices ← list(range(n))
for i in indices do
    if abs(i - lis[i]) ≥ 4 then
        return lis[i]
    end if
end for
return True
```

Summary

This report covered the internship at the lab MSD at the Université de Haute-Alsace. It focused on time series classification and the aspects related to it. It started by explaining the structure of the lab at the university, then the basic knowledge for the field of time series classification, and also what the lab's contributions were to the field. After explaining the basic knowledge, we explained our contribution to the lab. In ROCKET, we explained the different tests we did with the dilations and how they affected the performance. In Jigsaw, we explained the technique in images and how we adapted it for time series. Then, we explained the different parameters used in our adaptation, followed by a discussion about the architecture used, and we showed the obtained results and tried to analyse them. We finished the report by discussing our internship, its benefits and its impact on us. We also specified our contribution to the lab and how we benefitted the lab.

Key words: Time series Classification, Data augmentation, Deep learning, convolution, Jigsaw puzzle, Rocket.