IA321
Synthetic Data Augmentation For Time Series

Groupe 2 :
Paul POIRMEUR
Andre COSTA WERNECK
Lorenzo PERRIER DE LA BÂTHIE
Florine LEFER
Asma KHALIL

March 21, 2024

Contents

1 Introduction

Parfois la collecte de données est coûteuse, difficile, ou restreinte pour des raisons de confidentialité. L'augmentation de données synthétiques pour les séries temporelles permet de générer artificiellement de nouvelles données qui simulent fidèlement les caractéristiques statistiques des données réelles. Ainsi sont améliorées la robustesse et la performance des modèles de machine learning.

Considérons le secteur de l'énergie, où la précision des prévisions de la demande est vitale pour la gestion de l'approvisionnement en énergie et la planification des infrastructures. Les entreprises utilisent des séries temporelles historiques de la consommation d'énergie pour prédire les tendances futures. Cependant, ces séries temporelles peuvent être limitées en termes de variabilité et de volume, en particulier pour les nouvelles zones de développement ou pendant des périodes de changement rapide des habitudes de consommation, comme cela a été observé pendant la pandémie de COVID-19.

L'augmentation de données synthétiques peut jouer un rôle crucial dans ce contexte. Par exemple, en utilisant des techniques comme le bootstrapping, le SMOTE, la transformation par ondelettes, ou des modèles génératifs adversariaux (GAN), on peut générer de nouvelles séries temporelles qui capturent la variabilité quotidienne et saisonnière de la demande en énergie. Ces données synthétiques peuvent ensuite être utilisées pour enrichir l'ensemble de données d'entraînement, permettant aux modèles de mieux généraliser et de prédire avec précision la demande en énergie dans des scénarios non observés dans les données historiques.

Lien du repository GitHub du projet : https://github.com/LorenzoPerrier/TimeSeriesAugmentation

2 Approche Méthodologique

Afin d'évaluer la praticité des méthodes de génération de données pour la complétion de datasets déséquilibrés, nous avons adopté la méthodologie suivante. Initialement, nous avons évalué la précision d'une tâche de classification effectuée sur un dataset complet et équilibré. Ensuite, nous avons manuellement déséquilibré une classe du dataset afin d'observer l'impact sur la précision. Finalement, nous avons employé un modèle de génération de données pour rééquilibrer la classe affectée puis examiné l'effet sur la précision de la classification.

Dans ce contexte, deux équipes ont été constituées. La première a été chargée d'implémenter divers modèles de classification sur des séries temporelles, afin de comparer les approches existantes et créer un benchmark permettant d'évaluer les méthodes de génération de données. La deuxième s'est concentrée sur l'implémentation de différentes tâches de génération de données pour en permettre la comparaison.

Au cours de notre projet, nous avons été amenés à transformer les séries temporelles en images, car la génération de séries temporelles s'est révélée peu concluante. Ainsi, notre benchmark se concentre sur l'évaluation de différentes variations d'un dataset :

- Le dataset initial, idéalement équilibré.
- Le dataset déséquilibré : une classe a été réduite par rapport aux autres.
- Le dataset sous forme d'images : le dataset initial converti en images en utilisant la méthode du Gramian Angular Field pour chaque série temporelle.
- Le dataset sous forme d'images déséquilibré.
- Le dataset sous forme d'images rééquilibré : des données générées par un modèle viennent compléter la classe déséquilibrée afin qu'elle contienne le même nombre d'éléments que les classes équilibrées.

En comparant les performances des différents modèles de classification sur ces variations de dataset, nous pouvons évaluer la capacité des modèles de génération à créer des images qui améliorent la précision de classification sur des datasets déséquilibrés.

3 Time series data

Les données de séries temporelles se caractérisent par des observations collectées sur des intervalles de temps successifs et régulièrement espacés. Les exemples incluent les cours de bourse, les mesures de température et les chiffres de vente, où chaque donnée est associée à un horodatage spécifique. L'objectif principal de l'analyse

des séries temporelles est de prédire les événements futurs en identifiant les motifs et les tendances au sein des données passées.

4 Datasets utilisés

Table 1: Comparaison des datasets GUNPOINT et FaultDetectionA

Caractéristique	**GUNPOINT**	**FaultDetectionA**
Échantillons d'entraînement	160	10912
Échantillons de test	40	2728
Longueur d'une série	150	5120
Nombre de classes	2	3
Dimensionnalité	Uniaxiale	Uniaxiale
Type de données	Mouvements	Capteurs (*SENSOR*)
Répartition des classes	Binaire	9,09% / 45,55% / 45,55%

[1]Pour le dataset FaultDetectionA, les pourcentages de répartition des classes correspondent respectivement à des conditions non endommagées, endommagées à l'intérieur, et endommagées à l'extérieur.

GUNPOINT est destiné à la reconnaissance de l'activité humaine (HAR) et se concentre sur deux classes d'activités liées aux mouvements de la main, enregistrées comme séries temporelles univariées. Il est principalement utilisé pour distinguer entre tirer une arme fictive d'un étui et pointer du doigt.

FaultDetectionA fait partie d'un ensemble de données plus large, conçu pour la détection d'anomalies ou de défaillances à partir de séries temporelles univariées obtenues par des capteurs. Ce dataset, issu de mesures prises sur un système électromécanique sous différentes conditions, vise la classification multi-classes pour identifier les états non endommagés et endommagés (à l'intérieur ou à l'extérieur) des équipements.

5 Déséquilibrage du dataset

```
[ ]  def make_dataset_imbalanced(X, y, class_to_reduce=0, reduction_factor=0.1):
         """
         Reduces the number of samples in the specified class by the reduction factor.

         Parameters:
         X (np.array): Feature data.
         y (np.array): Labels.
         class_to_reduce (int): The class whose samples are to be reduced.
         reduction_factor (float): The fraction of the class_to_reduce samples to keep (0 < reduction_factor <= 1).

         Returns:
         (np.array, np.array): The imbalanced features and labels.
         """
         # Indices of the class to reduce and the other class
         reduce_indices = np.where(y == class_to_reduce)[0]
         other_indices = np.where(y != class_to_reduce)[0]

         # Randomly select a subset of the class to reduce
         reduced_indices = np.random.choice(reduce_indices, int(len(reduce_indices) * reduction_factor), replace=False)

         # Combine the reduced class indices with the other class indices
         new_indices = np.concatenate([reduced_indices, other_indices])
         np.random.shuffle(new_indices)  # Shuffle the indices to mix the classes

         # Create the new imbalanced dataset
         X_imbalanced = X[new_indices]
         y_imbalanced = y[new_indices]

         return X_imbalanced, y_imbalanced
```

Figure 1: Fonction du deséquilibre

Ici, nous avons choisi de réduire les données de la classe 0 par défaut (pour le "GUNPOINT"). Initialement, chaque classe comprenait 80 données. En appliquant un facteur de réduction de 0.1, le nombre d'échantillons dans la classe "1" est resté à 80, tandis que celui de la classe "0" a été réduit à 8.

6 Génération

Pour rééquilibrer nos datasets, nous avons exploré la technique du SMOTE, ainsi que l'utilisation de VAE et VQ-VAE, en complément des CGANs. Nous avons d'abord tenté de générer des séries temporelles, sans succès. Nous avons donc finalement transformé les données temporelles en données images, se basant sur la technique du Gramian Angular Field.

6.1 SMOTE

SMOTE, ou Synthetic Minority Over-sampling Technique, est une technique de suréchantillonnage visant à équilibrer les classes dans les ensembles de données pour l'apprentissage automatique. Contrairement au suréchantillonnage simple, qui se contente de reproduire les échantillons de la classe minoritaire, SMOTE génère de nouveaux échantillons synthétiques qui sont des variations des échantillons existants. Voici comment SMOTE fonctionne en détail :

SMOTE commence par identifier la classe minoritaire pour laquelle des échantillons supplémentaires sont nécessaires. Pour chaque échantillon de la classe minoritaire, SMOTE trouve ses k plus proches voisins dans l'espace des caractéristiques. Après, pour chaque échantillon de la classe minoritaire, un échantillon synthétique est généré en choisissant l'un des k plus proches voisins et en générant un échantillon aléatoire le long de la ligne qui relie l'échantillon en question à ce voisin. Finalement, une fois les échantillons synthétiques générés, ils sont ajoutés à l'ensemble de données original, ce qui augmente la taille de la classe minoritaire et rend l'ensemble de données plus équilibré. Pour plus de détails, voir [2].

6.2 VAE et VQ-VAE

Nous avons d'une part utilisé un Auto-Encodeur Variationnel (VAE) pour la génération de données temporelles. Bien que l'entraînement se soit déroulé sans encombre, la génération de données conditionnée par classe n'a pas donné les résultats escomptés. Nous avons alors procédé à une analyse de l'espace latent en recourant à une Analyse en Composantes Principales (ACP) réduite à deux dimensions, comme illustré dans la Figure 2.

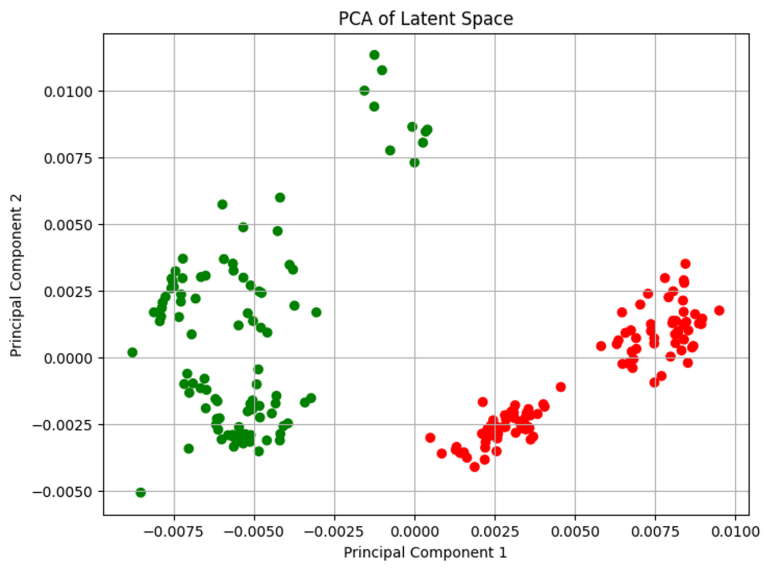


Figure 2: ACP de l'espace latent du VAE

Cette analyse a révélé une séparation linéaire claire entre les classes selon les deux axes principaux, indiquant que l'encodeur devait avoir appris certaines caractéristiques distinctives pour chaque classe. Néanmoins, les moyennes des caractéristiques latentes par classe étaient presque identiques, ce qui a conduit à la production de 2 séries temporelles non distinguables. Face à ces difficultés, nous avons envisagé l'utilisation d'un clustering sur l'espace latent complet. Cependant, guidés par les conseils de nos encadrants, nous avons opté pour une approche basée sur un Auto-Encodeur Vectoriel Quantifié (VQ-VAE), réputé pour sa capacité à mieux structurer l'espace

latent. Notons qu'à ce moment nous sommes passées de données temporelles à données images, comme conseillé par nos encadrants, celles-ci s'avérant plus prometteuses. Après avoir identifié une implémentation de référence d'un VQ-VAE fonctionnel [1], nous l'avons adaptée à notre jeu de données, fichier VQ_VAE_bis.ipynb. Cette adaptation a inclus la modification des couches de convolution transposée du décodeur par des convolutions standards, à l'exception de deux couches dédiées à l'upsampling. De plus, une attention particulière a été accordée au paramètre β, régulant l'importance de la commitment loss, ce qui a permis d'affiner la qualité des images reconstruites et d'éliminer les artefacts de grille observés précédemment avec $\beta = 0,25$, la valeur usuelle de ce paramètre, Figures 3, 9.

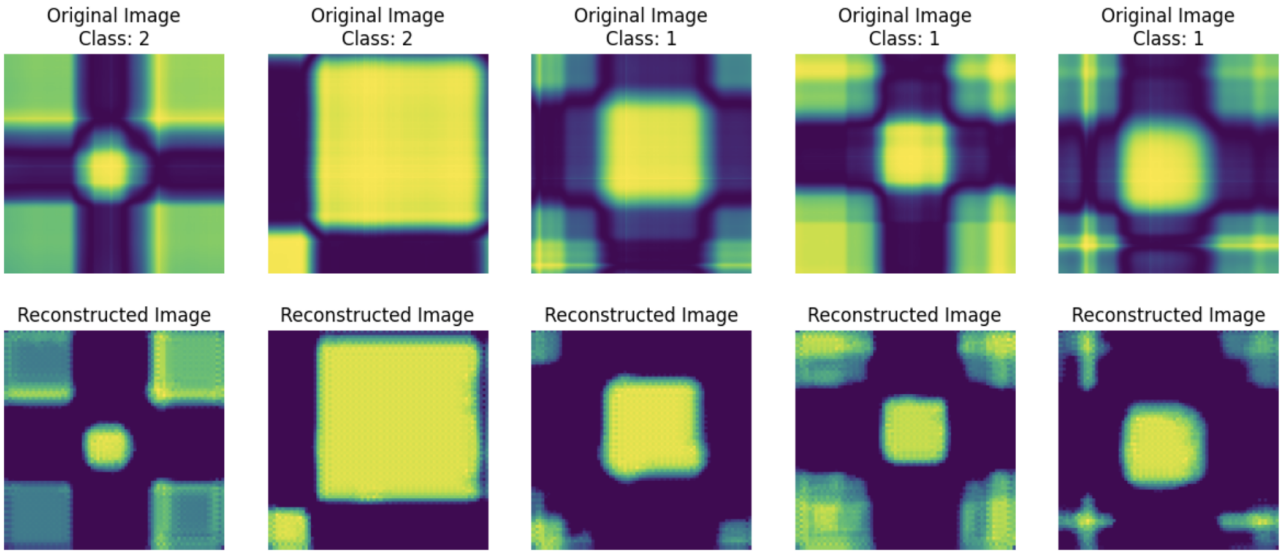


Figure 3: VQ-VAE : Images originales vs reconstruites avec $\beta = 0.25$

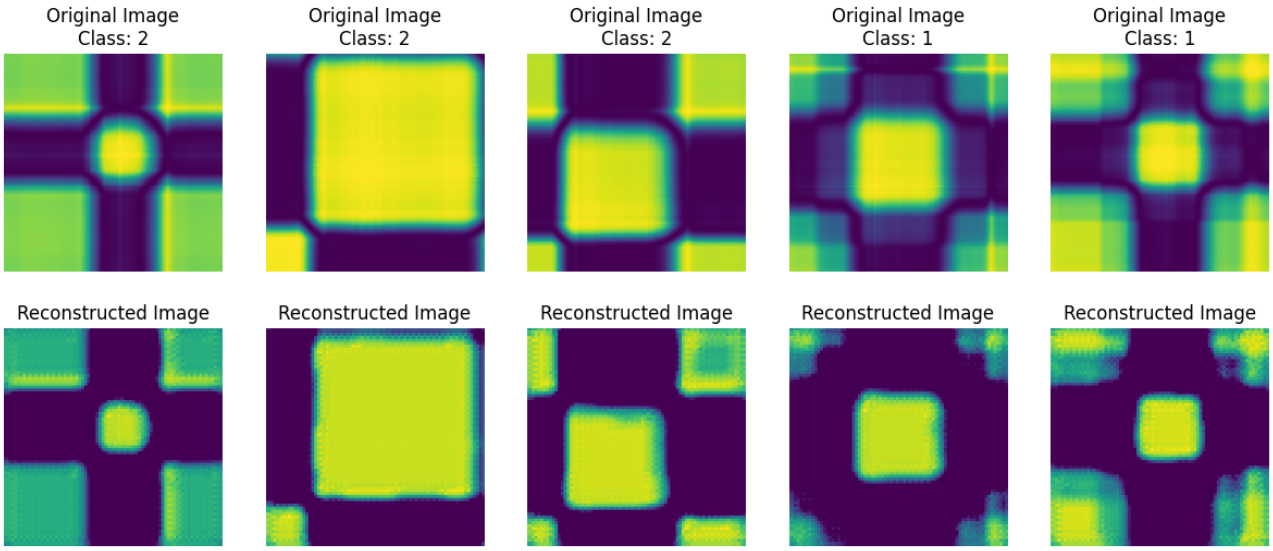


Figure 4: VQ-VAE : Images originales vs reconstruites avec $\beta = 0.5$

Ensuite, nous nous sommes penchés sur la phase de génération. Il s'agit de déterminer les distributions des vecteurs du dictionnaire sur l'espace latent, conditionnellement à une classe. Une méthode couramment adoptée consiste à utiliser le réseau PixelCNN [4] pour apprendre les a priori sur les vecteurs latents. Toutefois, il nous a été recommandé d'opter pour un CGAN afin de mener à bien cette tâche. Par conséquent, nous avons associé

un VQ-VAE à un CGAN, permettant ainsi la génération d'images via le décodeur du VQ-VAE.

6.3 VQ VAE couplé à un Conditional GAN pour la génération

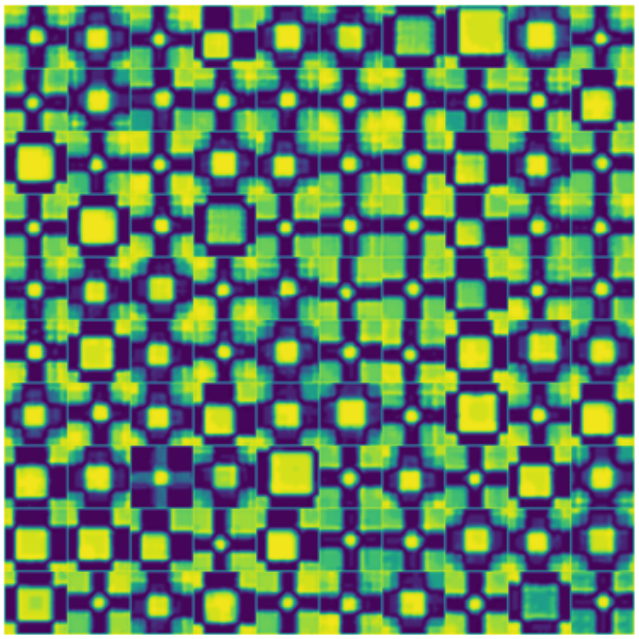


Figure 5: Images reconstruites avec un VQ VAE

Le code du VQ-VAE que nous avions récupéré d'un dépôt GitHub était complexe. En ajoutant à cela un manque de temps, nous ne sommes pas parvenus à entraîner un GAN conjointement à celui-ci pour la génération. Nous avons donc décidé d'entraîner un VQ VAE plus simple (avec des couches de upsample / convolution) pour reconstruire correctement une image. Nous obtenons un codebook d'embeddings qui est utilisé par le décodeur pour générer des images. Cependant, bien que nous sachions que le codebook contient les mots pour générer une image, nous ne savons pas vraiment comment les assembler pour que le décodeur les comprenne et génère une image réaliste. C'est pourquoi nous entraînons un GAN conditionnel pour assembler les mots du codebook afin de les transmettre au décodeur et de générer des images qui semblent les plus plausibles.

Par conséquent, nous avons entraîné un GAN avec une architecture MLP qui devait générer des embeddings. Le discriminateur devait alors dire si l'image générée par le décodeur est réelle ou fausse.

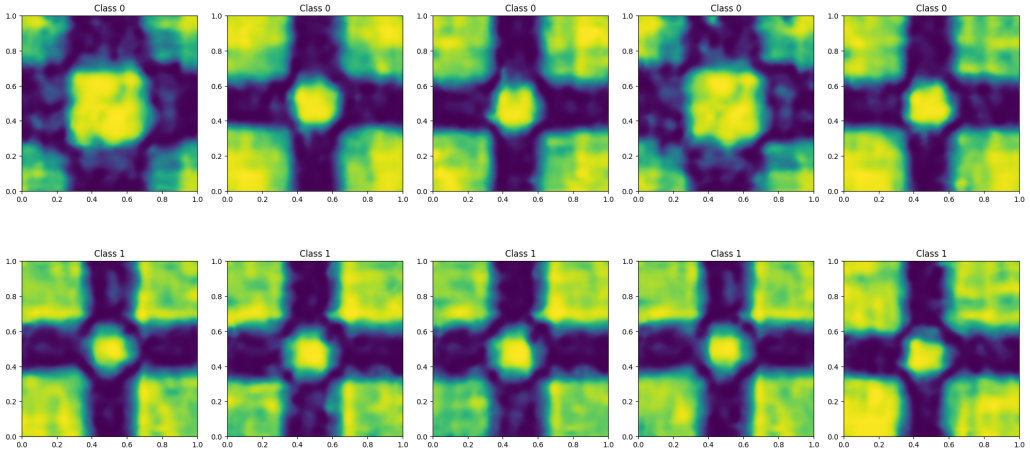


Figure 6: Images générées avec un CGAN + decoder du VQ VAE

On remarque que la qualité des images n'est pas satisfaisante. En effet, la reconstruction du VQ VAE n'étant pas initialement parfaite et le GAN a également du mal à trouver les bons embeddings du codebook à utiliser.

Avec un peu plus de temps, nous aurions pu régler ces problèmes. D'une part, il aurait fallu modifier l'architecture du VQ VAE pour obtenir une reconstruction quasi-parfaite des images (ou obtenant des résultats de classification similaire au dataset de base avec les réseaux de classification utilisés). Ensuite, nous aurions également pu jouer sur le nombre d'embeddings et la dimension des embeddings du codebook, sachant qu'un codebook avec un grand nombre d'embeddings et/ou avec une grande dimension permettrait de mieux capturer les détails des images mais rendrait plus difficile la tâche de choisir les bons embeddings pour le GAN. Enfin, nous aurions pu tester différentes architectures du générateur du GAN (ici MLP).

6.4 Conditional GAN

Une méthode très efficace pour générer de nouveaux éléments à partir d'exemples données sont les réseaux adversariaux génératifs. Dans notre cas, nous voulons pouvoir choisir la classe que nous voulons générer, c'est pour cela que nous nous sommes plutôt tournés vers un GAN conditionnel, c'est à dire qu'en plus du bruit en input du générateur, nous rajoutons un embedding de la classe qui nous intéresse. Cependant l'entraînement d'un GAN est très compliqué : on ne sait pas exactement quand l'arrêter et les hyperparamètres ainsi que l'architecture du générateur et du discriminant sont difficiles à choisir. De plus, il faut veiller à ce que le discriminant ou le générateur ne s'améliorent pas trop vite l'un par rapport à l'autre, sinon l'on se retrouve avec une loss à 0 et l'autre à 1 donc l'entraînement stagne.

En ce qui concerne l'architecture du générateur, nous avons testé différentes méthodes de génération d'image : MLP, convolutions transposées, upsample et convolutions.

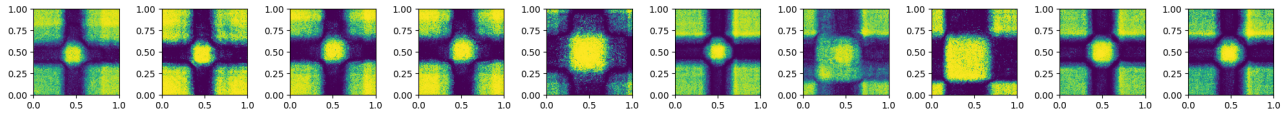


Figure 7: CGAN : Images générées avec une architecture MLP

On remarque un certain éparpillement des pixels générés. Bien que l'on puisse reconnaître les motifs de l'image, la qualité de la génération n'est pas optimale car un MLP a du mal à comprendre la cohérence locale d'une image. C'est pour cela que nous nous sommes plutôt penchés vers les convolutions transposées.

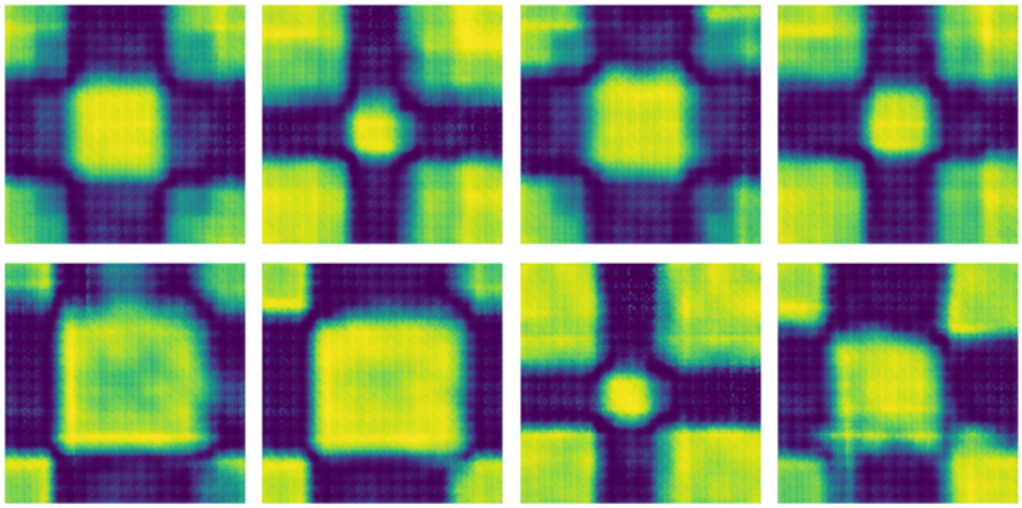


Figure 8: CGAN : Images générées avec une architecture convTranspose

En utilisant les convolutions transposées, on s'aperçoit que l'on a des images plus lisses mais elles ont un défaut ! En effet, on remarque un motif en dammier qui se répète sur toute l'image, dû aux convolutions

transposées dont les noyaux se recoupent sur certains endroits de l'image. Utiliser des couches d'Upsample suivies de couches de convolutions est donc plus approprié pour la génération d'images.

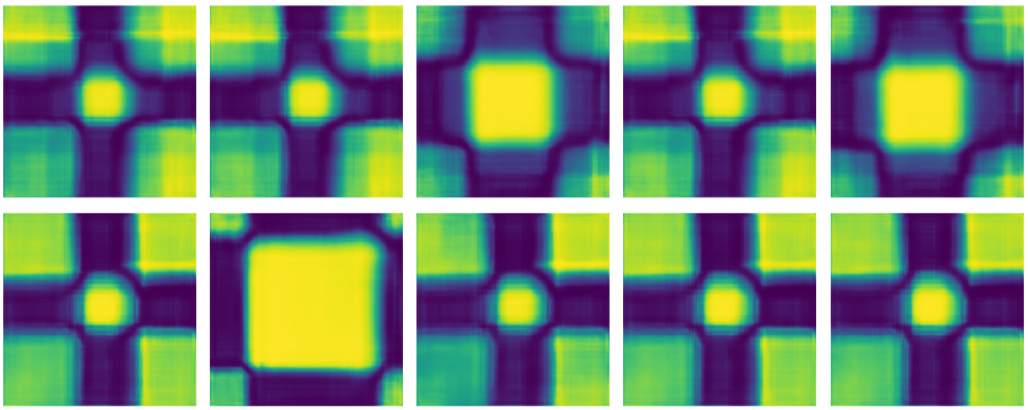


Figure 9: CGAN : Images générées avec une architecture Upsample/convolution

On obtient ainsi la meilleure qualité d'image. Nous avons ensuite généré des images pour la classe sous-représentée afin d'évaluer le gain de performance sur la classification.

Conlusion : Notre CGAN permet en ce moment de générer des images de meilleure qualité (ce qui se voit par ailleurs sur les performances lors de la classification). Cependant, si l'on arrivait à avoir un VQ-VAE avec un bon pouvoir de reconstruction, alors la génération avec le GAN serait plus simple en utilisant le decoder du VQ VAE. De plus, nous avons rencontré des difficultés sur la génération d'images avec peu de données et avec plus de classes comme le dataset MiddlePhalanx ou FaultDetectionA. Les courbes sont beaucoup moins faciles à distinguer, les images se ressemblent donc beaucoup. C'est là un des désavantages d'utiliser la transformation en image pour la génération.

7 Classification

Dans cette section, nous détaillerons la construction des modèles et l'approche adoptée pour la tâche de classification en trois étapes, conformément à ce qui a été présenté dans la section sur l'Approche Méthodologique.

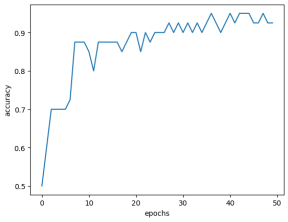
Cette procédure a été réalisée pour des données sous forme de séries temporelles et d'images, grâce à la transformation GASF (Gramian Angular Summation Field, voir [5]), qui permet de convertir des séries temporelles en images, en conservant leurs informations temporelles et spatiales.

Il convient également de noter que pour l'ensemble de données FautDetectionA, il n'a pas été nécessaire de déséquilibrer l'ensemble de données, car il est naturellement déséquilibré.

7.1 MLP

Une première approche envisagée pour la classification des séries temporelles a été d'utiliser un réseau de neurones à perceptrons multicouches (MLP). L'implémentation réalisée se composait uniquement d'une à deux couches entièrement connectées, suivies d'une fonction softmax pour effectuer la prédiction. Bien que ce modèle soit simple à mettre en œuvre, il n'est pas le plus performant pour les séries temporelles, car il ne prend pas en compte l'information temporelle intrinsèque à ces données. Cependant, la tâche de classification sur le dataset GunPoint n'étant pas particulièrement complexe, nous avons réussi à obtenir des résultats satisfaisants.

On peut observer dans les Figures 10 et 11 que la classification avec un MLP permet d'atteindre des précisions très élevées. Il est notamment observable que le déséquilibre du dataset rend la tâche de classification plus complexe ; le modèle nécessite davantage d'époques pour atteindre des précisions comparables, et la précision finale n'est pas aussi élevée. Il est important de noter que les échelles d'époques diffèrent entre les graphiques sur le tableau 1.



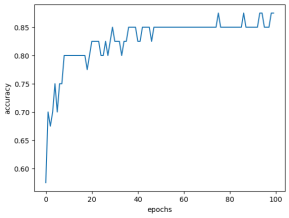
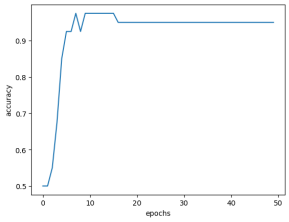


Figure 10: Précision sur l'ensemble test à (dataset complet à gauche et dataset déséquilibré à droite)



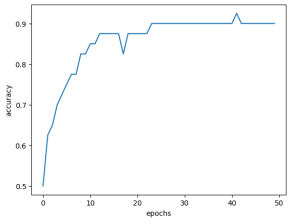
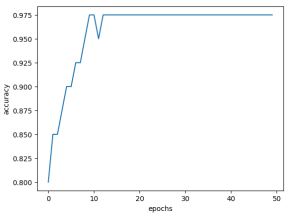


Figure 11: Précision sur l'ensemble test en image (dataset complet à gauche et dataset déséquilibré à droite)



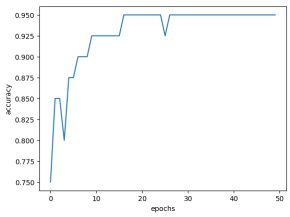


Figure 12: Précision sur l'ensemble test en image rééquilibré (avec génération CGAN à gauche et CGAN/VQVAE à droite)

La Figure 12 révèle que le rééquilibrage du dataset à l'aide de données générées conduit à des performances équivalentes ou supérieures à celles observées avec des datasets complets et équilibrés. L'entraînement sur un dataset rééquilibré conduit à de meilleurs résultats et ce, plus rapidement que sur les datasets déséquilibrés. Les images générées par le CGAN permettent un entraînement de meilleure qualité que celui utilisant les images issues de la combinaison des CGAN et VQ-VAE.

7.2 LSTM

Les LSTM, ou réseaux de neurones à mémoire longue-courte durée, sont un type de réseau de neurones récurrents (RNN) conçus pour surmonter les limitations des RNN traditionnels dans l'apprentissage des dépendances à long terme. Cette capacité à se souvenir et à accéder à des informations du passé rend les LSTM particulièrement bien adaptés pour la classification de séries temporelles, où comprendre le contexte et la séquence des événements dans le temps est crucial pour faire des prédictions précises.

Ainsi, dans la partie LSTM, nous avons commencé par utiliser un modèle très simple, contenant seulement une couche LSTM et une couche linéaire à la fin pour calculer les scores pour chaque classe. En outre, la technique de *dropout* a été utilisée, car les LSTM ont généralement tendance à s'adapter de manière excessive (*overfitting*) et il serait donc important d'avoir la possibilité de régulariser le réseau.

Après quelques tests, une certaine instabilité a été observée dans la réponse du modèle, bien qu'il ait déjà bien fonctionné et que la précision des tests ait été supérieure à 92 %. Afin d'améliorer les résultats et de préparer une architecture pour des ensembles de données plus complexes, un deuxième classifieur a été mis en œuvre avec deux couches LSTM, qui utilisaient également la technique du dropout pour la régularisation.

Ces architectures ont été utilisées et comparées dans tous les ensembles de données étudiés. Pour choisir le meilleur paramétrage du modèle, une validation croisée a été effectuée et les paramètres qui ont rapporté la meilleure précision de validation ont été choisis.

Il convient également de mentionner que, dans le but de limiter encore plus le *overfitting* et de garantir le meilleur modèle possible, les techniques de *checkpoint* et *early stop* ont été utilisées.L'emploi de *checkpoints* sauvegarde le modèle à chaque amélioration de la *loss* de validation, assurant ainsi la disponibilité du meilleur modèle pour les tests futurs. La technique de *early stop* arrête l'entraînement lorsque la métrique surveillée ne progresse pas sur un nombre défini d'époques, aidant à prévenir l'*overfitting*.

Le meilleur modèle sur les deux ensembles de données testés est le LSTM à deux couches implémenté avec Tensorflow/Keras. Ses résultats sur l'ensemble de test pour le dataset Gunpoint se trouvent dans la section 8

du présent rapport. Voici également quelques courbes concernant la performance lors de l'entraînement pour le même dataset.

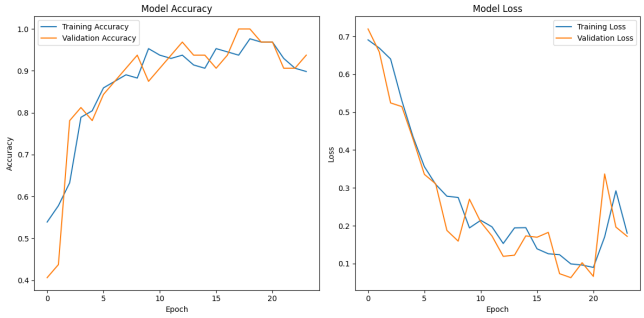


Figure 13: LSTM appliqué aux séries temporelles: GUNPOINT

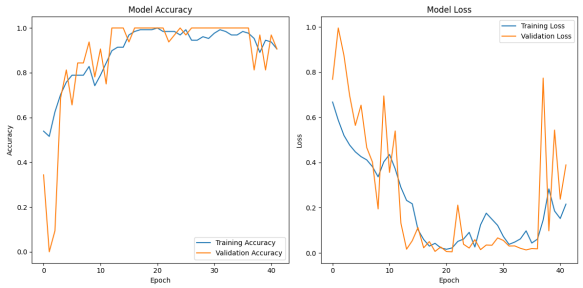


Figure 14: Performance sur le dataset reéquilibré avec SMOTE

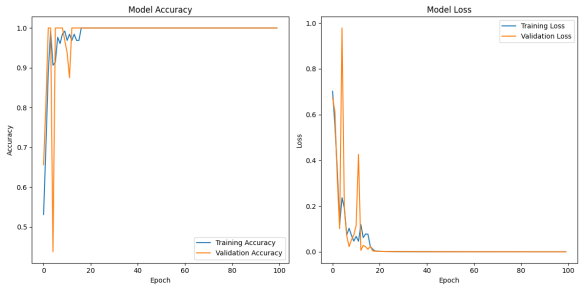


Figure 15: Performance après rééquilibrage avec les images générées par le DCGAN

Les graphiques montrent que les GANs ont efficacement généré des données mimant les réelles, avec une précision de test de 97,5%, un résultat significatif malgré la simplicité du dataset Gunpoint. Par ailleurs, SMOTE et VQ-VAE ont notablement amélioré la performance du modèle à environ 85% en test, démontrant leur utilité pour l'augmentation de données malgré une fiabilité inférieure à celle des GANs. Il est important de souligner que ces résultats étaient attendus pour le SMOTE et pour le GAN, mais pas pour le VQ-VAE, qui peut encore être amélioré.

Pour le dataset FaultDetectionA, les performances ont considérablement diminué, comme prévu, étant donné que cet ensemble de données est bien plus grand, plus complexe et présente une séparation très subtile entre les trois classes présentes. Les modèles utilisés ont été adaptés pour gérer ici un problème multi-classe, mais en conservant la même architecture et les mêmes techniques de régularisation, d'arrêt anticipé et de points de contrôle.

Pour l'ensemble original, qui est déjà déséquilibré, une performance de 79% a été obtenue, ce qui prouve l'efficacité du modèle même pour un ensemble de données difficile. Il est également important de souligner les précisions par classe, vues ci-dessous.

À partir de cela, il est possible de constater de manière intéressante que le modèle ne se trompe pas dans la classe qui possède le moins d'exemples, mais plutôt dans les classes où la différence entre les membres est minime. Sans aucun doute, les classes 1 et 2 sont très similaires entre elles et moins semblables par rapport à la classe 0. De cette manière, notre modèle présente d'excellentes précisions pour les classes 0 et 2, mais échoue à classer correctement la classe 1, qu'il confond considérablement avec la classe 2.

En avançant vers l'étape de rééquilibrage du dataset, ni les images générées par les GANs ni la méthode SMOTE n'ont été efficaces. De plus, il s'est produit quelque chose qui mérite d'être mentionné : les données générées par le SMOTE ont encore plus confondu le modèle, qui a commencé à classer presque tous les échantillons comme membres de la classe 2, posant un problème pour la performance de notre LSTM et montrant que cette technique n'est pas toujours la plus appropriée. Il est important de souligner que cette confusion

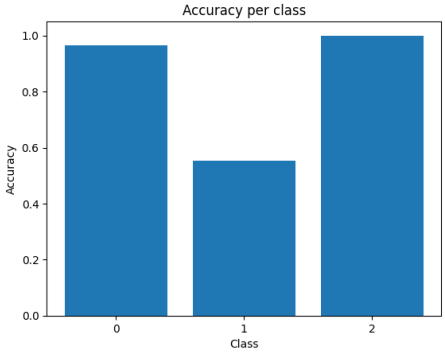


Figure 16: Performance par classe sur le dataset original FaultDetectionA avec LSTM

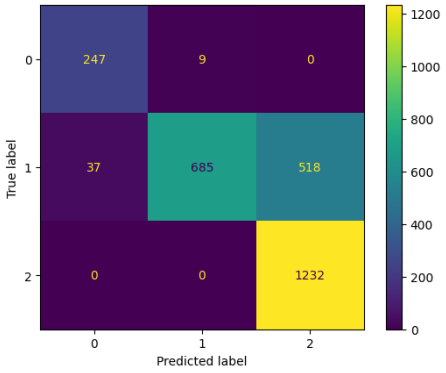


Figure 17: Confusion Matrix par classe pour le dataset FaultDetectionA avec LSTM

du réseau est compréhensible étant donné la manière dont le SMOTE génère ses échantillons artificiels et, s'appuyant sur le fait que les membres des classes se ressemblent beaucoup, on conclut que la génération de données par le SMOTE n'est pas assez précise pour différencier des données très proches avec des différences subtiles.

Enfin, plus de détails sur l'implémentation et sur les performances des modèles mis en œuvre, ainsi que sur les tests réalisés, se trouvent dans les différents notebooks contenus dans le dossier LSTM du github envoyé.

7.3 CNN

Le modèle que nous avons appliqué comporte deux couches de convolution qui appliquent des filtres (kernels) à l'entrée pour extraire des caractéristiques importantes. Chaque filtre apprend à reconnaître des motifs spécifiques.

Après chaque couche convolutive, une couche de mise en commun réduit la dimensionnalité de la sortie, résumant les caractéristiques en régions plus petites afin de rendre le modèle plus efficace et moins sensible aux variations de la position des caractéristiques.

Après les couches de convolution et de pooling, le modèle comprend des couches entièrement connectées qui permettent de combiner les caractéristiques apprises et de faire des prédictions. La première couche entièrement connectée (fc1) réduit la dimensionnalité à 128, et la seconde (fc2) produit la sortie finale correspondant aux classes de prédictions.

Nous avons appliqué la validation croisée afin de valider les performances de notre modèle et de choisir les meilleurs hyperparamètres . Sur les données brutes, nous avons obtenu les résultats suivants :

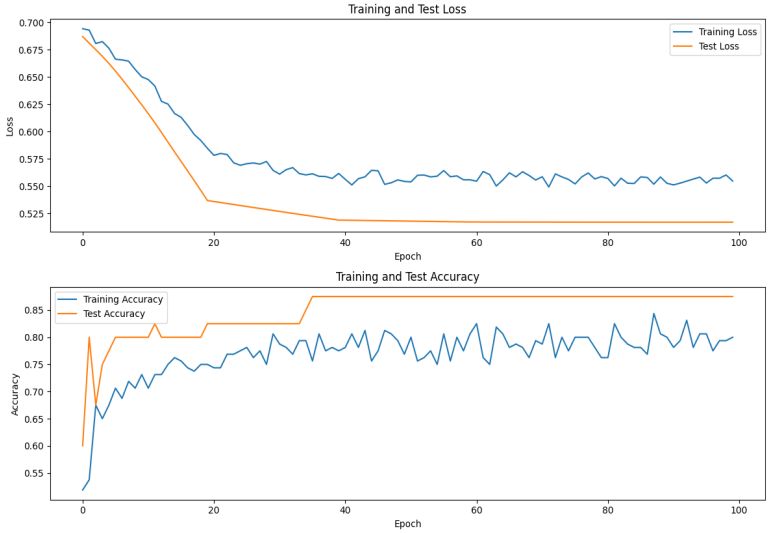


Figure 18: CNN appliqué aux séries temporelles : GUNPOINT

7.4 Performances sur le dataset deséquilibré et rééquilibré avec le SMOTE

Le rééquilibrage des données avec les images générées a amélioré la précision des tests. Malgré la variablité observée, le modèle est plus stable et plus performant. Le rééquilibrage des données peut contribuer à réduire le sur-ajustement en fournissant des exemples supplémentaires pour la classe minoritaire, ce qui aide le modèle à mieux apprendre les caractéristiques de toutes les classes.

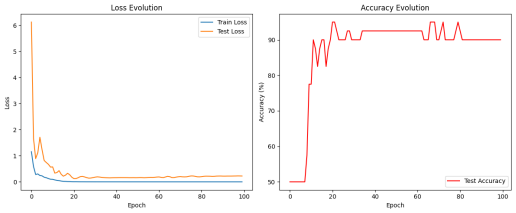


Figure 19: Performance sur le dataset déséquilibré

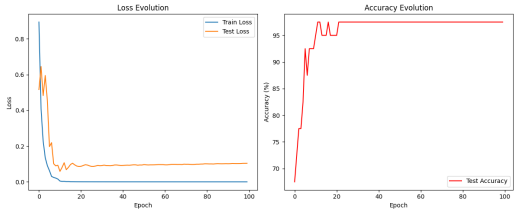


Figure 20: Performance après rééquilibrage avec les images générées

7.5 Transformer

Nous avons tout d'abbord défini l'encodeur qui consistait en une normalisation de couche, des multi-têtes attention et un réseau feed forward constitué de couches de convolution 1D. L'attention multi-têtes permet au modèle de se concentrer sur différentes parties de la séquence d'entrée pour chaque prédiction, tandis que le FFN traite chaque position dans la séquence indépendamment des autres. Des connexions de saut sont utilisées autour de ces deux parties pour faciliter le flux de gradient et aider à combattre le problème de la disparition du gradient dans les réseaux profonds.

Encodeurs Transformer : Le modèle utilise plusieurs blocs encodeurs Transformer (layer_normalization, multi_head_attention, dropout, conv1d), chacun suivant un schéma similaire :

Layer_normalization : Normalise les activations des couches précédentes pour chaque échantillon afin d'accélérer la convergence et d'améliorer les performances du modèle.

Attention multi-têtes : Permet au modèle de se concentrer simultanément sur différentes parties de la séquence d'entrée, afin de mieux capturer les dépendances à long terme.

Abandon : Utilisé pour la régularisation en abandonnant aléatoirement des unités pendant la formation afin d'éviter le surapprentissage.

Connexions résiduelles : Après chaque sous-couche d'attention et de réseau neuronal feed-forward, une connexion résiduelle est ajoutée suivie d'une normalisation de la couche. Cela permet d'éviter que le gradient ne disparaisse dans les réseaux profonds.

Réseau Feed-Forward (FFN) : À la suite de chaque attention multi-têtes, le modèle met en œuvre un simple RFP à l'aide de couches conv1d. Ces couches agissent comme une transformation linéaire de la dimension des caractéristiques, suivie d'une activation ReLU pour la non-linéarité.

7.6 ViT

Le Vision Transformer (ViT) est une nouvelle architecture qui adapte le modèle Transformer, couramment utilisé dans le traitement du langage naturel, aux tâches de classification d'images. Contrairement aux réseaux neuronaux convolutionnels traditionnels, le ViT applique le mécanisme du transformateur directement aux parcelles d'images. Dans notre cas, nous avons tout d'abord transformé nos données en images, puis divisé ces images en patchs que nous avons finalement passées dans le PatchEncoder. Chaque encodage de patch passe ensuite par une couche PositionEmbedding pour conserver le contexte positionnel.

Encodeur transformateur : Les patchs incorporés avec des informations de position passent par une série de couches TransformerBlock, où le modèle apprend à s'intéresser à différents patchs de manière sélective.

Mise en commun et classification : La sortie de l'encodeur transformateur est ensuite regroupée et passe par une couche softmax pour déterminer les probabilités de classe.

Procédure de formation : Le modèle est compilé avec l'optimiseur Adam et la fonction de perte d'entropie croisée catégorielle clairsemée, qui convient aux tâches de classification avec des classes mutuellement exclusives. Le processus de formation est contrôlé à l'aide d'une fonction de rappel ReduceLROnPlateau, qui ajuste le taux d'apprentissage lorsque la perte de validation atteint un plateau, et d'une fonction de rappel EarlyStopping pour éviter le surajustement. Le SMOTE a pu atténuer le problème de surajustement comme le montre

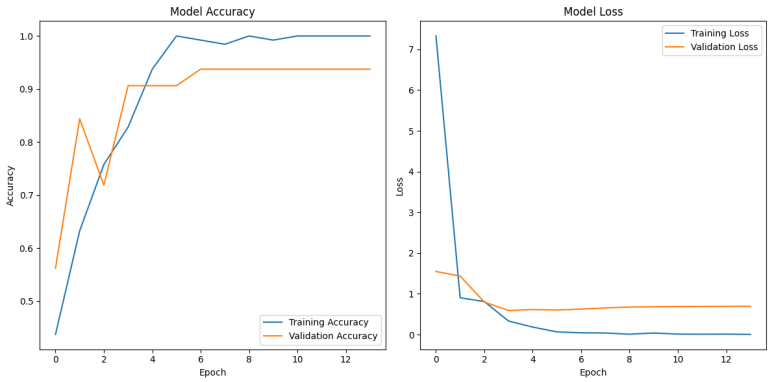


Figure 21: Performance après SMOTE

la réduction de l'écart entre la précision d'entraînement et de test. En effet cette amélioration au niveau de la précision suggère que l'ajout des données synthétiques a donné au modèle plus d'informations pour apprendre davantage sur les caractéristiques de la classe minoritaire.

7.7 minGPT

Dans cette étude, une démarche déployée pour adapter le modèle minGPT [3], une réimplémentation concise du Generative Pretrained Transformer (GPT) en PyTorch, à l'analyse de séries temporelles. Initialement conçu

pour traiter des séquences textuelles, minGPT se distingue par son architecture épurée et son module de self-attention causale. L'adaptation aux séries temporelles repose sur la modification de la première couche, traditionnellement dédiée aux embeddings de mots, par une couche fully connected. Cette adaptation permet au modèle de traiter directement des séries temporelles, exploitant ainsi leur nature séquentielle de manière analogue aux données textuelles. La tête de sortie du modèle a été adaptée pour convenir aux tâches de classification. Les résultats obtenus grâce à cette implémentation sont présentés ci-dessous.

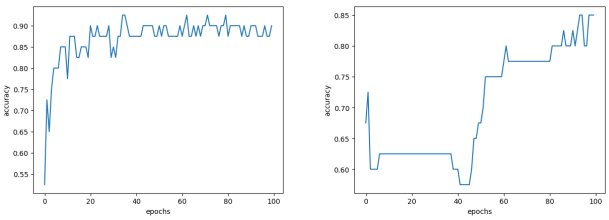


Figure 22: Précision sur l'ensemble test initial (sur dataset complet à gauche et dataset déséquilibré à droite)

Il apparaît, au vu des valeurs de précision obtenues sur l'ensemble de test pendant l'entraînement, que ce modèle n'est pas particulièrement adapté à ce type de données. Le dataset est probablement trop restreint pour une architecture aussi complexe. Néanmoins, ce modèle reste prometteur pour des tâches de deep learning appliquées aux séries temporelles, dans un contexte où les données sont plus abondantes.

En fin de compte, une tentative a également été faite pour implémenter un minGPT adapté aux images résultant de la transformation de GASF. Cependant, cette mise en œuvre s'est avérée complexe et il n'a pas été possible de la finaliser à temps pour cette étude. Ainsi, elle reste un travail à réaliser dans le futur.

8 Résultats

Nous résumons les performances de nos modèles de classification appliqués aux ensembles de données équilibrés, déséquilibrés, et rééquilibrés grâce à la technique de SMOTE, ainsi qu'à ceux basés sur les données générées.

	CNN	**LSTM**	**MLP**	**Transformers**
Dataset de base	0.925	0.95	0.95	0.99
Dataset avec une classe réduite à 10%	0.95	0.5	0.875	0.9
Dataset rééquilibré utilisant SMOTE	0.7	0.85	0.95	0.9
Dataset en image	0.975	0.975	0.975	0.9
Dataset réduit en image	0.9	0.725	0.90	0.85
Dataset rééquilibré (avec images générées par CGAN)	0.95	0.975	0.975	0.7

Table 2: Les résultats de précision pour les différents modèles appliqués au dataset "GUNPOINT"

9 Conclusion

En clôture de notre rapport sur "Synthetic Data Augmentation For Time Series", nous pouvons affirmer que les approches de génération de données synthétiques, notamment SMOTE, VAE, VQ-VAE, et les GAN conditionnels, ont démontré une capacité notable à améliorer la robustesse et la performance des modèles de machine learning dans le contexte de la classification de séries temporelles. Nos expériences sur des ensembles de données variés, tels que "GUNPOINT" et "FaultDetectionA", ont souligné l'importance de l'augmentation des données, non seulement pour contrecarrer les effets du déséquilibre des classes, mais aussi pour renforcer la capacité prédictive des modèles face à des situations non observées dans les données historiques.

En particulier, le rééquilibrage des ensembles de données à l'aide de SMOTE et la génération d'images synthétiques via les CGANs ont produit des améliorations significatives dans la précision de classification des

modèles de deep learning tels que CNN, LSTM, MLP, et Transformers. Cela valide l'efficacité des données synthétiques dans l'enrichissement des ensembles d'entraînement et la réduction de sur-ajustement. La transformation des séries temporelles en images à l'aide de la méthode GASF a ouvert de nouvelles voies pour la classification et la génération de données, en conservant les informations temporelles et spatiales essentielles.

Le projet a également abordé des défis techniques complexes, comme l'entraînement des GANs et l'optimisation des auto-encodeurs variés, démontrant ainsi notre capacité à naviguer et à appliquer des techniques avancées de machine learning. L'intégration du VQ-VAE avec le CGAN, bien que n'ayant pas produit la qualité de génération escomptée, a offert des perspectives prometteuses pour de futures améliorations.

En conclusion, notre recherche confirme le potentiel des techniques d'augmentation de données synthétiques dans l'amélioration de la prédiction des séries temporelles et ouvre des pistes pour l'exploration future. Elle souligne également l'importance d'une méthodologie rigoureuse et d'une évaluation critique dans l'application des modèles de deep learning. L'ensemble du travail accompli, documenté soigneusement dans notre dépôt GitHub, fournit une base solide pour les chercheurs et les praticiens désireux d'approfondir le domaine de l'augmentation des données synthétiques.

Ce projet, mené avec rigueur et dévouement par notre équipe, représente une avancée significative dans l'utilisation de l'intelligence artificielle pour l'augmentation des données dans les séries temporelles. Il illustre notre engagement à pousser les frontières de la connaissance et à développer des solutions innovantes aux problèmes complexes de l'industrie et de la recherche.

References

[1] Airalcorn2. Pytorch implementation of vector quantized variational autoencoder (vq-vae). https://github.com/airalcorn2/vqvae-pytorch/tree/master, 202X.

[2] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[3] karpathy. mingpt, 2022. https://github.com/karpathy/minGPT?tab=readme-ov-filelicense.

[4] A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu. Conditional image generation with pixelcnn decoders. *CoRR*, abs/1606.05328, 2016.

[5] Z. Wang and T. Oates. "encoding time series as images for visual inspection and classification using tiled convolutional neural networks.". *AAAI Workshop*, 2015.