Classification automatique des signaux acoustiques de l'observatoire de château Landon

Rapport de Stage

➢ Plan du rapport :

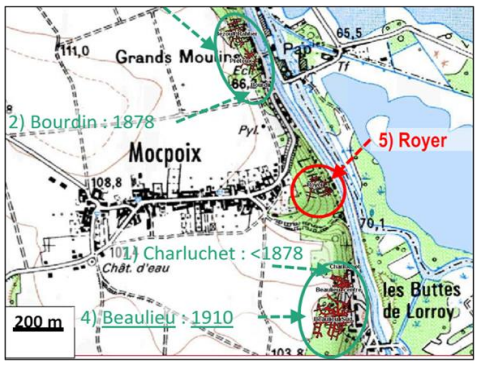
1. Introduction:

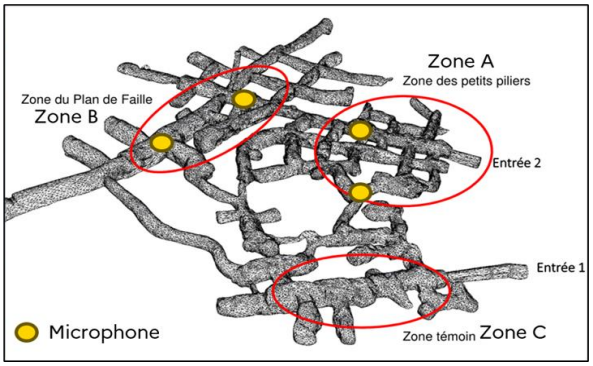
Dans le cadre de ce stage, l'objectif était de travailler sur une base de données comprenant plusieurs enregistrements de signaux acoustiques provenant d'une cavité souterraine située à Château Landon, nommée Royer. Ces enregistrements ont été effectués dans le cadre d'une surveillance acoustique visant à comprendre l'évolution du comportement de la carrière au fil du temps.

Cette surveillance n'était pas exclusive aux enregistrements acoustiques. D'autres types de suivis ont été ajoutés dans la carrière, pouvant être classés en deux catégories : un suivi "eau", concernant les grandeurs physiques telles que la pression, l'humidité, la température et la teneur en eau, pour lesquels de nombreux capteurs ont été installés, ainsi qu'un suivi "solide", portant sur les déplacements, fissures et déformations des roches, pour lequel plusieurs capteurs ont également été mis en place.

Les enregistrements des signaux acoustiques sont déclenchés par un algorithme lorsque l'intensité du signal capté dépasse un seuil prédéfini. Nous avons envisagé de développer un modèle simple de machine Learning qui se base sur la classification des signaux selon des caractéristiques bien choisis dans le but de distinguer entre différents types d'enregistrements principalement en deux catégories : chute de bloc et autres.

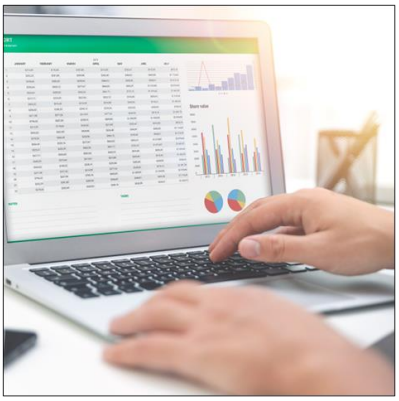
➢ Présentation du site de château Landon :





2. Problématique et Objectifs du stage et plan d'action:

➢ 2.1 Problématique du stage :



- Beaucoup de signaux enregistrés quotidiennement
- Classification manuelle laborieuse de ces signaux
- Manque de précision dans les classifications

➢ 2.2 Objectifs du stage :

- Réaliser une classification automatique des signaux
- Améliorer la qualité des signaux acoustiques
- Analyser la corrélation des signaux avec d'autres données statistiques.

➢ 2.3 Plan d'action :

● Exploration et validation du catalogue existant (2019/2021) : Stratégie de révision de l'ancien catalogue et reclassification des signaux acoustiques utiles pour l'apprentissage automatique.

● Amélioration de la qualité des signaux acoustiques : Application de filtres passe-haut pour éliminer les bruits indésirables dans les signaux utiles et augmenter la précision.

● Recherche des caractéristiques utiles pour la classification : Identification des caractéristiques temporelles et fréquentielles pouvant distinguer les catégories de signaux acoustiques, telles que la durée, les rapports énergétiques, les amplitudes, les centroïdes de fréquence, etc.

● Construction d'une base de données pertinente : Création d'une base de données Excel labellisée contenant les caractéristiques sélectionnées pour l'apprentissage des modèles de machine learning.

● Validation de la base de données : Analyse des colinéarités entre les variables, analyse en composantes principales, matrices de corrélation pour assurer une indépendance relative entre les caractéristiques.

● Développement des modèles de machine Learning : Exploration de divers modèles de machine Learning pour la classification automatique et comparaison de leurs performances.

● Application du modèle sur les signaux non classifiés (21/22) : Test des performances des modèles de machine learning sur une nouvelle base de données sans entraînement préalable pour évaluer leur comportement avec de nouvelles données.

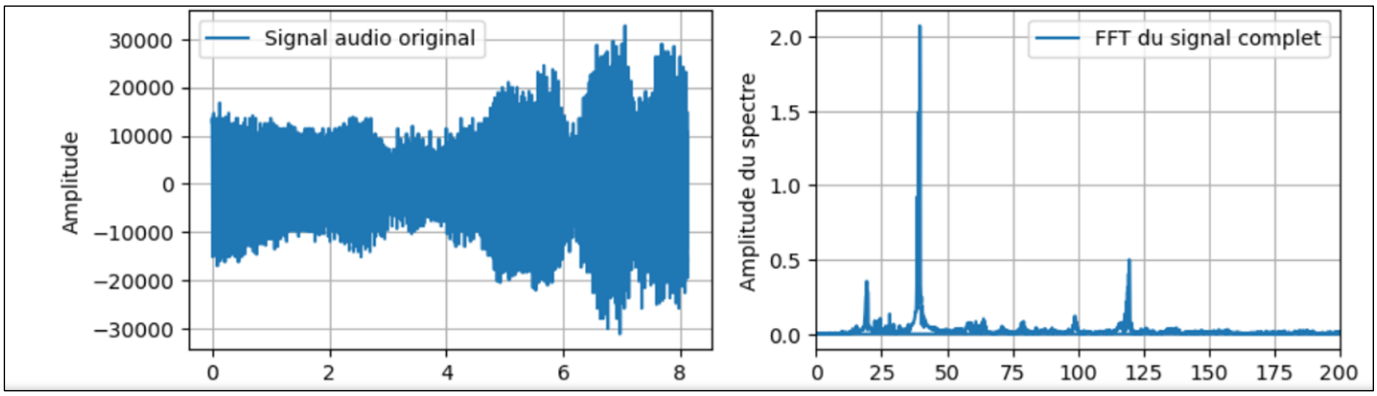
● Développement des modèles de Deep Learning : Développement de modèles tels que le CNN spectrogramme, le CNN temporel, le RNN temporel, et comparaison de leurs performances avec les modèles de machine learning.

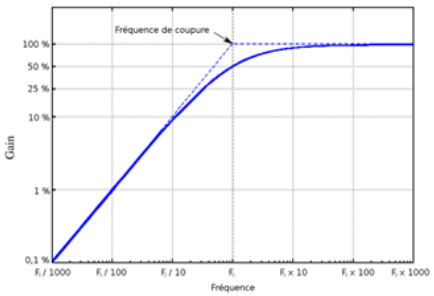
● Analyse de corrélation avec d'autres données statistiques : Étude de la corrélation avec d'autres données statistiques, comme les données pluviométriques.

3. Exploration & validation du catalogue existant :

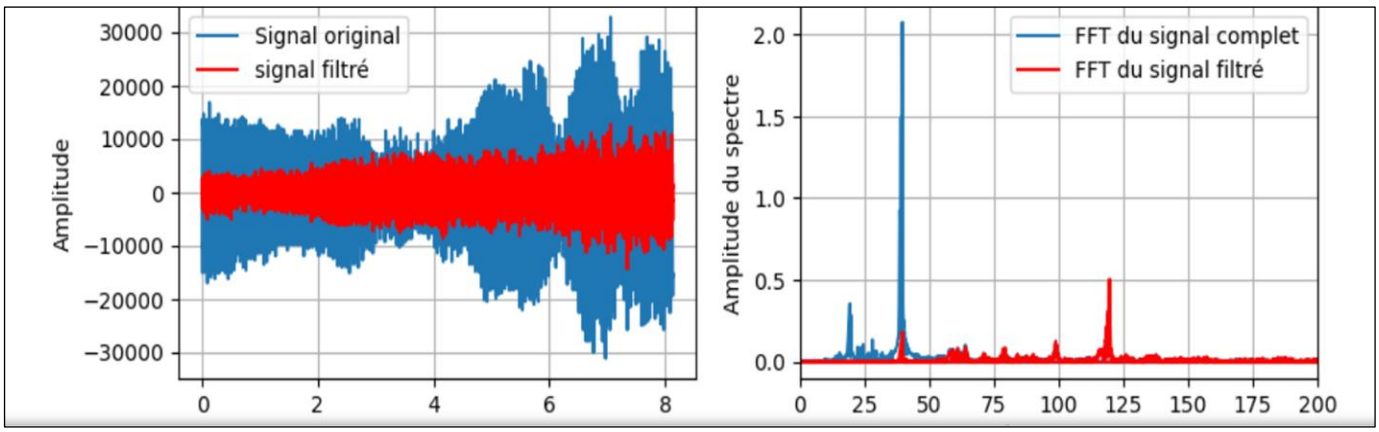
Après avoir écouté quelques échantillons de signaux acoustiques de chute de bloc, nous avons constaté la présence d'un fort bruit de fond perturbant le signal. Cela pourrait être lié à la sensibilité des microphones utilisés pour la surveillance, ce qui risque de perturber notre modèle étant donné que nous ne disposons pas d'une grande base de données pour l'entraîner. C'est pourquoi nous avons commencé à appliquer des filtres passe-bas simples Avec une fréquence de coupure basée sur l'analyse spectrale du signal. Nous écoutons ensuite le signal filtré et le comparons avec l'original.

➢ 3.1) Application d'un filtre passe-haut sur un signal 'autre ' :

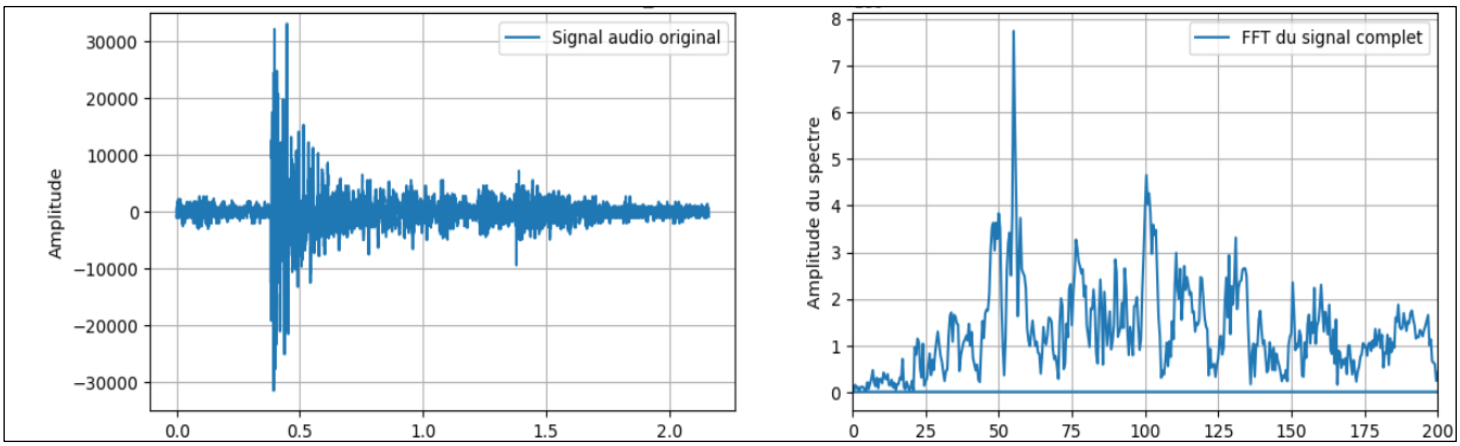


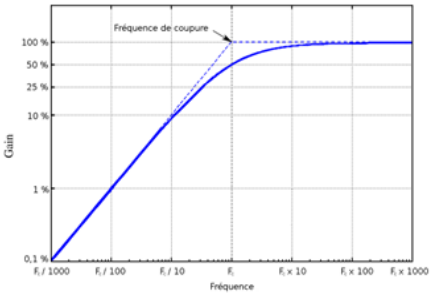


➢ une atténuation significative des amplitudes des basse fréquence du signal.

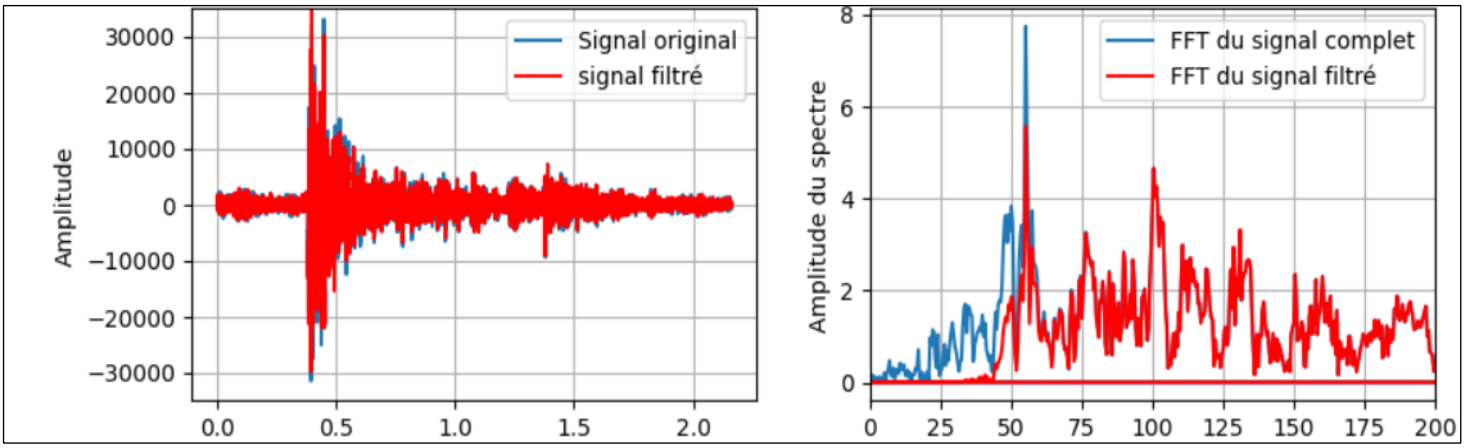


➢ 3.2) Application d'un filtre passe-haut sur un signal 'Chute de blocs ' :





➢ Le signal reste relativement intact puisque ses amplitudes sont de hautes fréquences.



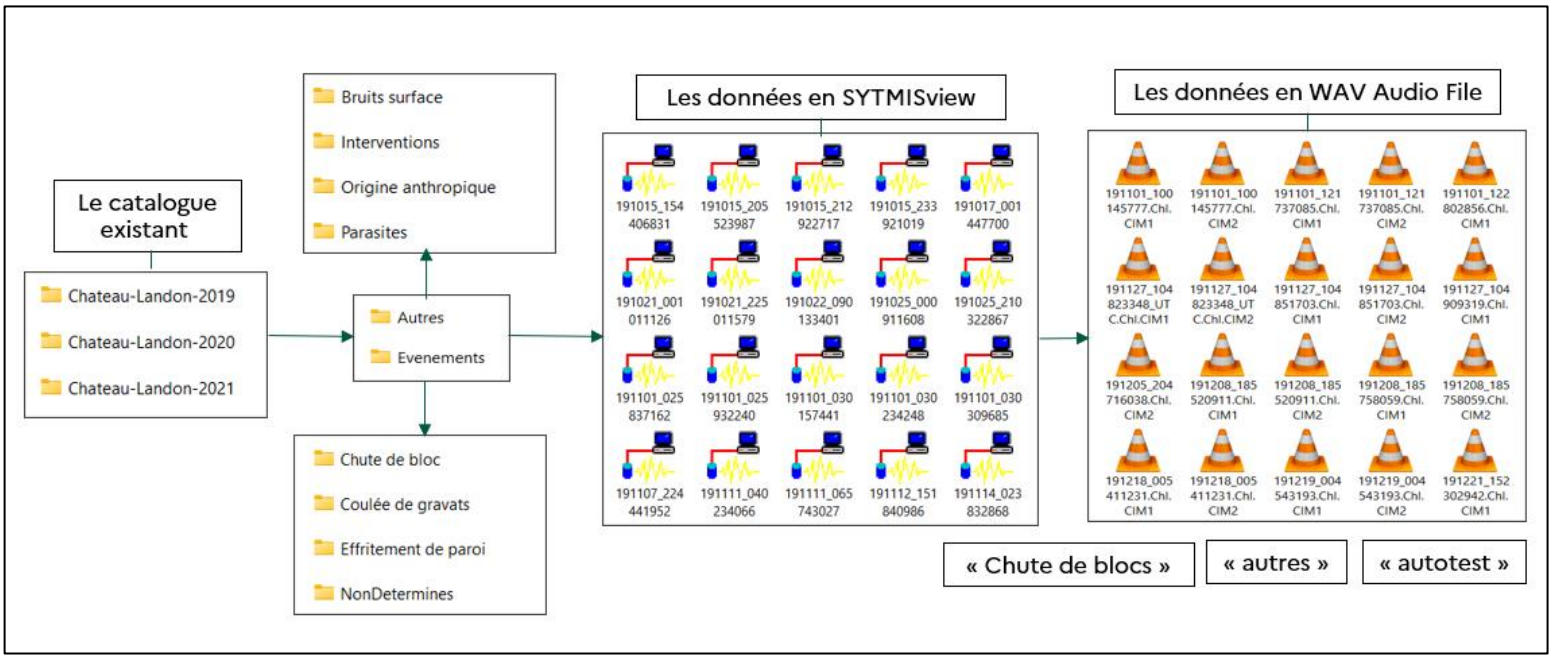
Ce filtre passe-haut avec une fréquence de coupure de 50 Hz permet d'atténuer les basses fréquences des signaux « autres » tout en ayant peu d'impact sur les chutes de bloc. Il sera très utile pour certaines caractéristiques temporelles que nous définirons dans la prochaine partie.

➢ 3.3) Validation du catalogue existant :

Pour valider le catalogue existant réalisé par l'ancien stagiaire, qui a classifié manuellement les signaux acoustiques de la carrière de Royer sur le site de Château-Landon pour les années 2019, 2020 et 2021, nous avons réexaminé ses regroupements. Il avait classé les signaux en deux groupes principaux : « événements » et « autres ». Les événements étaient subdivisés en quatre catégories : chutes de bloc, effritement de parois, coulées de gravats, et signaux non déterminés. Les autres signaux étaient divisés en quatre sous-catégories : bruit de surface, interventions, origines anthropiques et parasites.

Cependant, pour notre stage, cette répartition n'est pas idéale pour l'apprentissage d'un modèle de machine Learning, car nous ne disposons pas d'une grande base de données pour toutes ces classes. Par conséquent, nous avons revu ces catégories en écoutant les signaux et les avons reclassifiés en trois catégories essentielles pour notre étude de classification automatique : chute de bloc, autres, et autotest, sans subdiviser davantage ces catégories.

Le schéma ci-dessous illustre la stratégie de validation du catalogue existant ainsi que la nouvelle classification qui sera utilisée pour la construction d'une base de données pertinente pour nos modèles.



4. Recherche des caractéristiques utiles pour la classification des signaux :

Dans cette section, nous nous concentrons sur l'identification des caractéristiques pertinentes susceptibles de servir de données d'entraînement à notre modèle. Ces caractéristiques nous permettent également de définir des critères de distinction et de classification entre les signaux de chute de bloc et les autres.

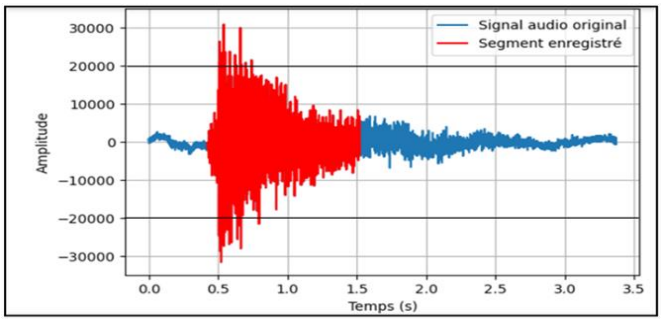
Tout d'abord, nous avons envisagé d'étudier l'aspect temporel du signal en examinant les variations de ses amplitudes et en recherchant des relations et des rapports calculables à partir de cet aspect temporel. Après avoir visualisé les signaux, nous avons noté que les signaux correspondant aux chutes de bloc qui nous intéressent présentent des variations plus importantes au niveau des amplitudes au moment de la chute. Les amplitudes maximales atteignent parfois 30000 digits, ce qui n'est pas le cas pour les autres événements. De plus, la différence entre l'amplitude maximale et minimale est plus significative, et par conséquent, le rapport max/min+1 peut être un indicateur significatif dans notre étude.

Ensuite, nous avons observé que l'énergie dans le cas des chutes de bloc présente un aspect particulier. Nous avons remarqué une concentration de l'énergie dans une bande bien précise, correspondant au moment de la chute. C'est pourquoi nous avons pensé que le calcul de la concentration de l'énergie dans cette bande pourrait être une caractéristique efficace dans notre démarche de classification.

➢ 4.1) Des caractéristiques temporelles et fréquentielles :

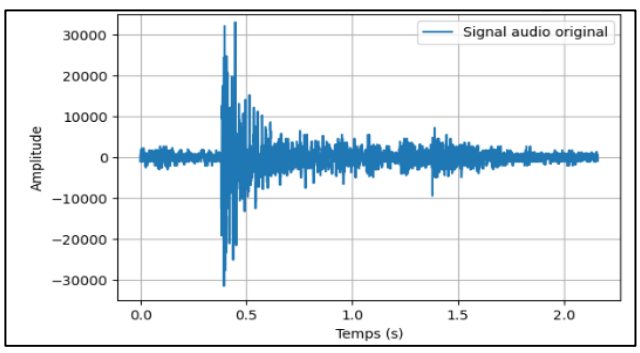
1. La fonction **`percentage_exceed_threshold`** donne des résultats précis en calculant le pourcentage d'amplitudes dépassant un seuil donné (nous avons choisi 20000 comme seuil). Cela s'avère très utile pour différencier les signaux de chute de bloc puissants.

```python
def percentage_exceed_threshold(signal, threshold):
    # Calculer le nombre d'amplitudes dépassant le seuil
    count_exceed_threshold = np.sum(np.abs(signal) > threshold)
    # Calculer le pourcentage correspondant
    percentage = (count_exceed_threshold / len(signal)+1) * 100
    return np.log(percentage)
```



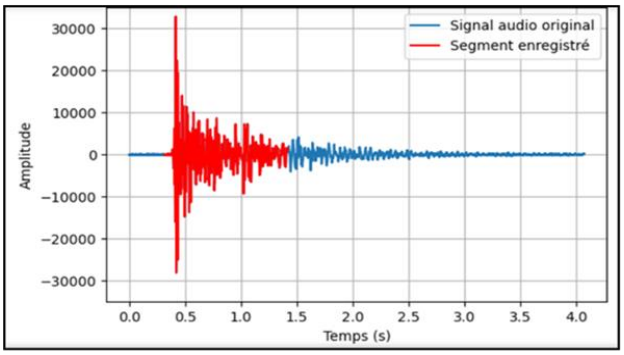
2. La fonction **`rapportminmax`** calcule le rapport entre l'amplitude maximale et minimale du signal. Cette fonction est également très pertinente dans notre étude car elle fournit une mesure distinctive entre les deux catégories de signaux.

```python
def rapportminmax(signal):
    if len(signal)==0:
        return 0
    max_val=np.amax(np.abs(signal))
    min_val = np.amin(np.abs(signal))
    rapport = max_val / (min_val+1)
    return rapport
```



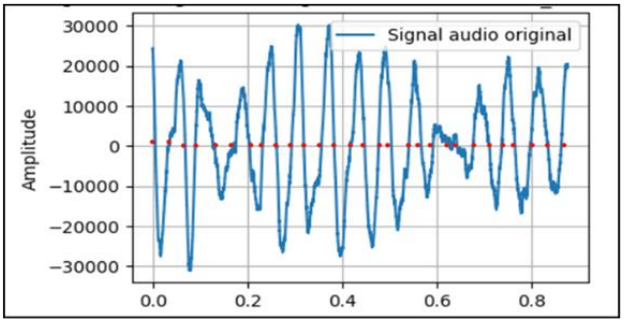
3. La fonction **`rapport_energie`** est extrêmement utile pour notre étude car elle permet de calculer le rapport d'énergie entre une bande utile du signal et l'énergie totale du signal. Cette fonction distingue vraiment entre les deux catégories de signaux et fournit des informations cruciales pour la classification.

```python
def rapport_energie(signal):
    segment ,a,b= intervalle_utile(signal, Fs, segment_duration)
    energie_utile = np.sum(np.abs(segment) ** 2)
    energie_totale = np.sum(np.abs(signal) ** 2)
    rapport = energie_utile / (energie_totale+1)
    return rapport
```



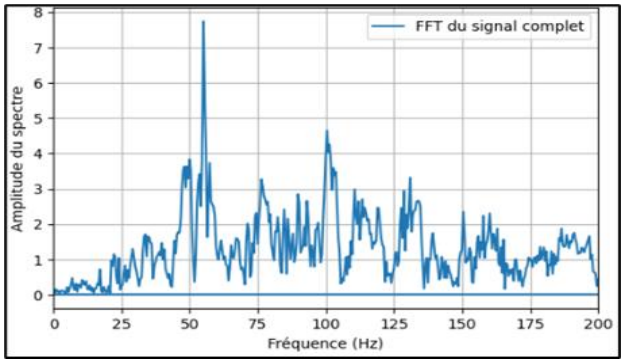
4. La fonction **TPPZ** est définie comme le nombre de fois où un signal passe de manière positive à négative ou vice versa au cours d'une période temporelle spécifique. Cette mesure est souvent utilisée pour des signaux comme les signaux audio ou les signaux de vibration, où elle peut fournir des indications sur des caractéristiques telles que la fréquence, la périodicité, ou même la présence de bruit ou d'artefacts.

```python
def nombre_passages_par_zero(signal):
    count = 0
    signal= lowpass_filter(signal, 500, fs, order=5)
    segment = intervalle_utile1(signal, Fs, segment_duration)
    for i in range(1, len(segment)):
        if segment[i-1] * segment[i] < 0:
            count += 1
    return count
```



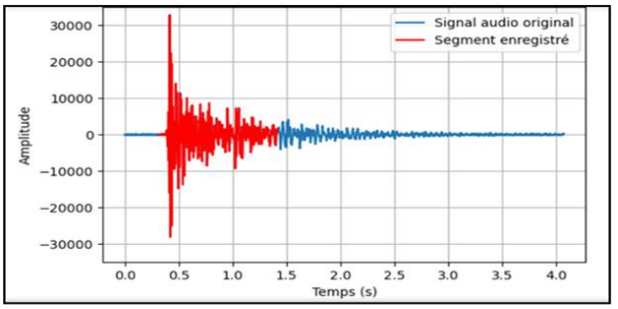
5. La fonction **`centre_de_gravite_freq`** calcule le centre de gravité des fréquences dans le spectre de fréquence d'un signal. Elle permet de visualiser la distribution des centres de gravité des spectres, et elle peut être une caractéristique utile pour distinguer entre les signaux. Nous avons observé que les signaux de type 'autres' sont des signaux de basses fréquences, tandis que les chutes de blocs sont relativement de hautes fréquences.

```python
def centre_de_gravite_freq(x, Fs):
    X = np.fft.fft(x)
    magnitude = np.abs(X)
    freqs = np.fft.fftfreq(len(x), 1/Fs)
    freqs_filtered = freqs[(freqs >= 0) & (freqs <= 1000)]
    magnitude_filtered = magnitude[(freqs >= 0) & (freqs <= 1000)]
    centre_gravite = np.sum(np.abs(freqs_filtered) * magnitude_filtered)
    return centre_gravite
```



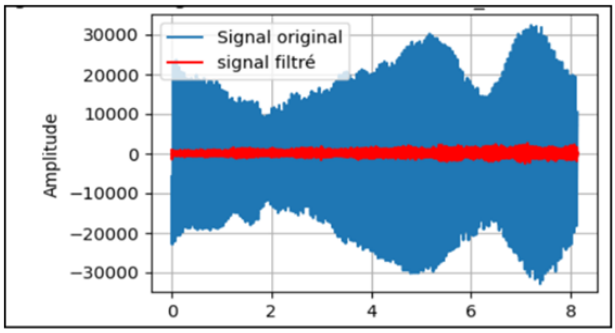
6. La fonction **SNR** calcul le rapport signal sur bruit, ce qui permet de déterminer la bande utile de signal. Le bruit est simplement la bande avant. Ensuite, nous faisons le rapport de la somme des amplitudes absolues. Cette mesure est cruciale pour évaluer la qualité du signal et pour la distinction entre les signaux.

```python
def SNR(signal):
    segment, start_index, end_index = intervalle_utile(signal, Fs, segment_duration)
    # Construire le bruit en excluant l'intervalle utile
    noise = np.concatenate((signal[:start_index], signal[end_index:]))
    # Calculer les moyennes des valeurs absolues du signal utile et du bruit
    mean_utile = np.sum(np.abs(segment)) / (len(segment) + 1) +0.001
    mean_noise = np.sum(np.abs(noise)) / (len(noise) + 1) +0.001
    # Calculer le SNR
    snr = mean_utile / mean_noise
    return   np.log(1+snr)
```

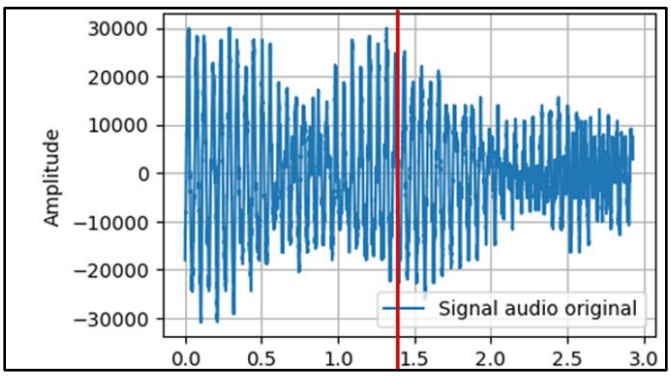


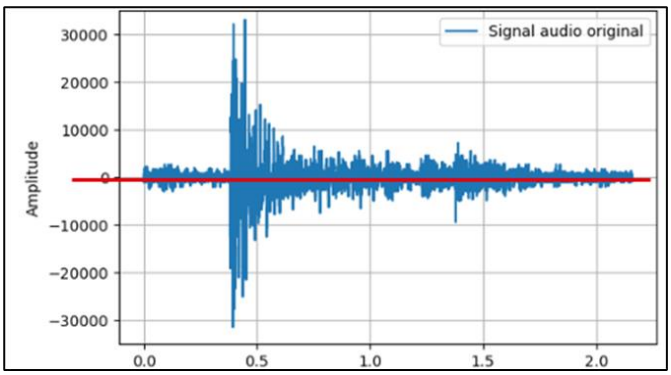
7. La fonction **energie_moyenne** calcule l'énergie moyenne du signal, c'est-à-dire la somme des amplitudes au carré divisée par le nombre total d'échantillons dans le signal. Cette caractéristique peut également être utile, notamment pour les signaux "autres" de basses fréquences qui s'atténuent après un filtre passe-haut, ce qui entraîne une diminution de leur énergie.

```python
def energie_moyenne(signal):
    # Calcul du carré de chaque échantillon
    carre_signal = np.square(signal)
    # Calcul de l'énergie moyenne
    energie_moyenne = np.mean(carre_signal)
    return np.log(energie_moyenne)
```



8. La fonction **SYM_horizontale** calcule la symétrie horizontale d'un signal. Pour les chutes de bloc, on remarque que le signal est très symétrique.
9. La fonction **SYM_verticale** calcule la symétrie verticale d'un signal. Pour les autres signaux, on remarque qu'ils sont symétriques verticalement.





10. La fonction **durée** permet de calculer la durée du signal, et peut être utile dans la classification puisque les signaux chute de blocs ont une courte durée.

➢ 4.2) Autres caractéristiques liées aux événements :

➢ Facteur de cohérence :

Le facteur de cohérence mesure la similitude entre deux signaux en fonction de leur fréquence. Il indique à quel point deux signaux sont cohérents l'un avec l'autre. Une valeur de cohérence élevée suggère que les signaux sont très similaires à travers les fréquences, ce qui peut être utile pour identifier les événements corrélés entre différents capteurs.

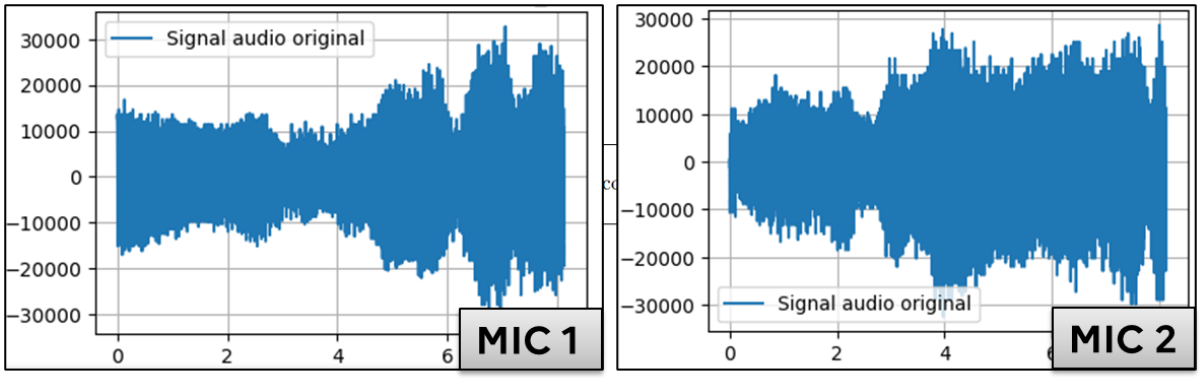
$$\text{coh}_1 = 1 - \frac{|\text{TPPZ}_1 - \text{TPPZ}_2|}{\sum_{i=1}^{2} \text{TPPZ}_i + 1}$$

$$\text{coh}_2 = 1 - \frac{|\text{rappe}_1 - \text{rappe}_2|}{\sum_{i=1}^{2} \text{rappe}_i + 1}$$

$$\text{coh}_3 = 1 - \frac{|\text{rappm}_1 - \text{rappm}_2|}{\sum_{i=1}^{2} \text{rappm}_i + 1}$$

$$\text{coh}_4 = 1 - \frac{|\text{rappa}_1 - \text{rappa}_2|}{\sum_{i=1}^{2} \text{rappa}_i + 1}$$

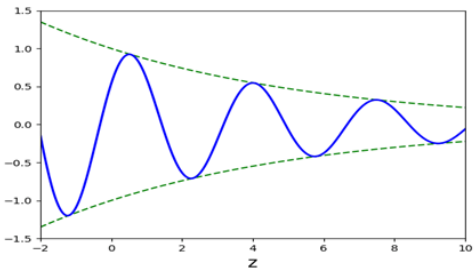
$$\text{facteur\_coherence} = \sum_{j=1}^{4} \text{coh}_j$$



➢ Facteur d'atténuation :

Le facteur d'atténuation mesure la diminution de l'amplitude du signal lorsqu'il se propage à travers un milieu ou une distance. Il quantifie combien le signal perd de sa force au fil du temps ou de la distance. Une valeur d'atténuation élevée indique une perte d'énergie importante.

$$\text{différence\_absolue} = (\text{moyenne\_signal1} - \text{moyenne\_signal2})^2$$



5. Construction d'une base de données : Data augmentation & Validation du data base

On a construit une base de données initiale qui était utile pour l'entraînement de mon premier modèle de machine learning. Cette base de données comprend 181 signaux considérés comme des chutes de blocs, 200 signaux classés dans la catégorie "autres", et 620 signaux d'autotests. En tout, cela représente 1001 lignes et 15 colonnes, soit environ 15 000 données.

➢ 5.1) Augmenter la base des données des signaux chute de bloc :

La data augmentation est une technique essentielle pour améliorer les performances des modèles d'apprentissage machine, surtout lorsque les données disponibles sont limitées. Dans le contexte de la classification des signaux acoustiques de chute de blocs, cette méthode devient particulièrement importante. En effet, nous n'avons pas suffisamment de données sur les signaux de chute de bloc, ce qui peut rendre difficile l'entraînement de modèles précis et robustes.
Pour pallier ce manque de données, nous avons mis en œuvre une fonction de data augmentation

Le code effectue plusieurs transformations sur les signaux audio existants pour créer des variations nouvelles et uniques. Voici une explication détaillée :

1. **Segmentation et ajustement d'amplitude** : La fonction commence par extraire un segment utile du signal original et ajuste les amplitudes de ce segment ainsi que des parties restantes du signal par des facteurs aléatoires.
2. **Ajout de bruit** : Pour rendre les signaux plus réalistes, un bruit gaussien est ajouté au signal augmenté.

3**. Troncature du signal** : Les dernières 0.4 secondes du signal sont supprimées pour simuler des variations naturelles dans la durée des signaux. Grâce à ces augmentations, nous avons pu enrichir notre base de données de signaux de chute de blocs, ce qui permet d'entraîner des modèles plus performants et d'améliorer leur capacité à généraliser à des nouveaux signaux

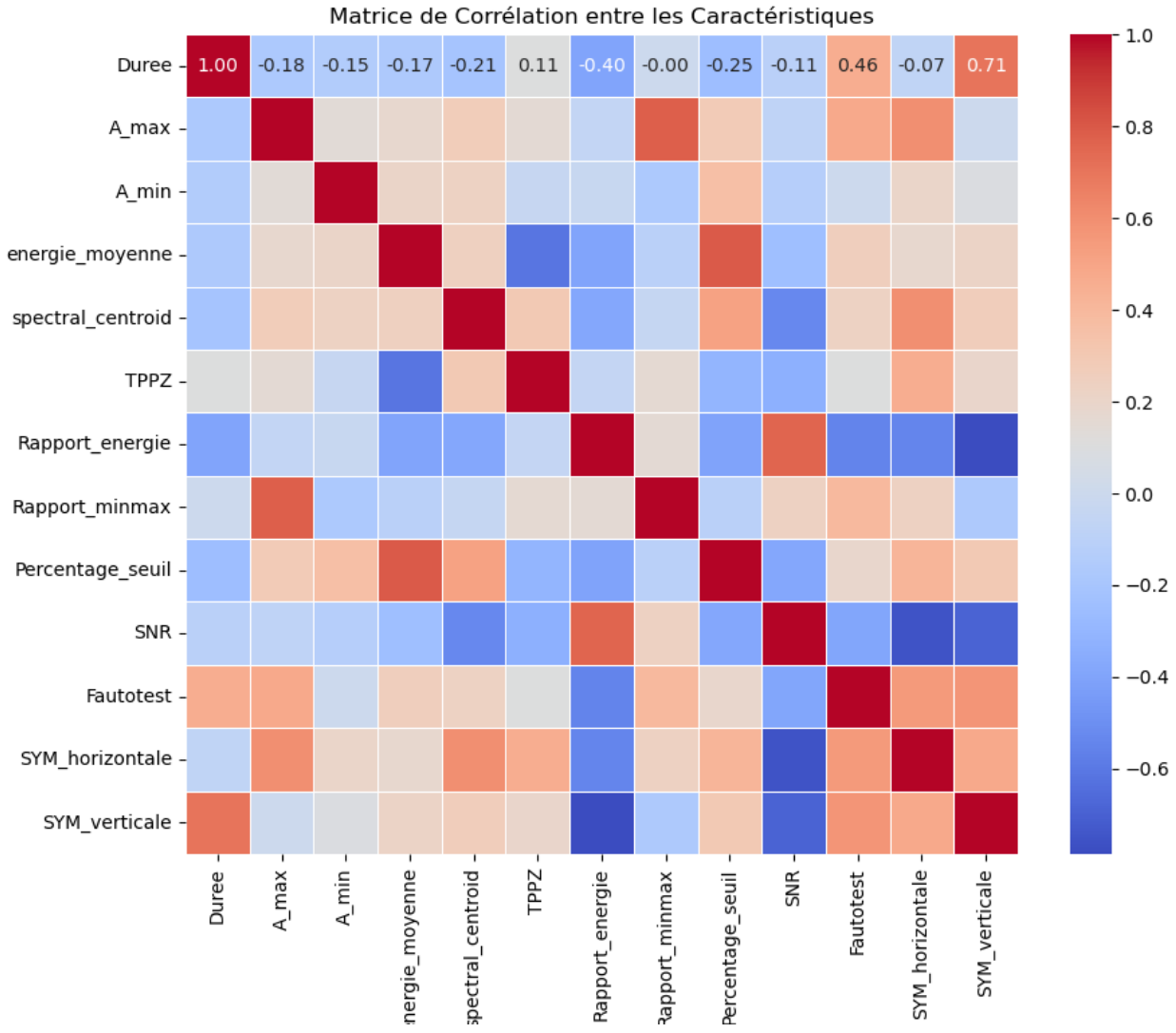
	Evenement	Etiquette	MIC	Duree	A_max	A_min	energie_moyenne	spectral_centroid	TPPZ	Rapport_energie	Rapport_minmax	Percentage_se
0	191101_100145777	0	1	2.160230	37571.419716	0.015267	16.098465	139.417640	278	0.948251	37006.454456	4.6153
1	191101_100145777	0	2	1.960255	31292.190595	0.100981	15.853596	139.337505	288	0.947278	28422.096265	4.6124
2	191101_100145777	0	3	2.160230	28094.432504	0.065506	16.348314	129.851180	246	0.904775	26367.210739	4.6100
3	191101_100145777	0	4	1.960255	38491.665149	0.270383	16.826775	134.337612	252	0.913939	30299.260531	4.6175
4	191101_121737085	0	1	2.142357	26375.373385	0.051212	16.366360	127.069151	116	0.984815	25090.431980	4.6123
...	...	...	...	...	...	...	...	...	...	...	...	
1552	191231_060702261	2	2	2.314086	30569.587369	0.000101	17.587855	133.289222	31	0.999997	30566.487341	4.6523
1553	191231_120702807	2	1	2.314461	30594.660253	0.000006	17.577348	133.382294	15	0.999997	30594.476309	4.6518
1554	191231_120702807	2	2	2.314461	30616.131721	0.000160	17.590306	133.299265	21	0.999997	30611.230962	4.6523
1555	191231_180702395	2	1	2.314711	30715.822902	0.001027	17.578897	133.372946	17	0.999998	30684.307160	4.6517
1556	191231_180702395	2	2	2.314711	30554.771376	0.000242	17.583848	133.293625	27	0.999997	30547.390910	4.6522

➢ 5.2) Validation de la base de données : détection des colinéarités :

❖ Matrice de corrélations des caractéristiques :

Une matrice de corrélation est un outil statistique qui permet d'évaluer les relations entre plusieurs variables. Chaque élément de la matrice représente le coefficient de corrélation entre deux variables, indiquant ainsi la force et la direction de leur relation linéaire. Les coefficients de corrélation varient de -1 à 1, où -1 indique une corrélation négative parfaite, 1 une corrélation positive parfaite, et 0 aucune corrélation.

Dans le contexte de l'analyse des signaux acoustiques de chutes de blocs, la matrice de corrélation est particulièrement utile pour détecter les colinéarités entre les différentes caractéristiques extraites des signaux. Une forte colinéarité entre deux variables signifie qu'elles véhiculent une information redondante, ce qui peut influencer négativement les performances des modèles de machine learning en termes de surajustement et d'interprétabilité.



La matrice de corrélation fournie ci-dessus présente les corrélations entre différentes caractéristiques des signaux acoustiques :

Une forte corrélation entre les caractéristiques peut s'expliquer par plusieurs raisons :

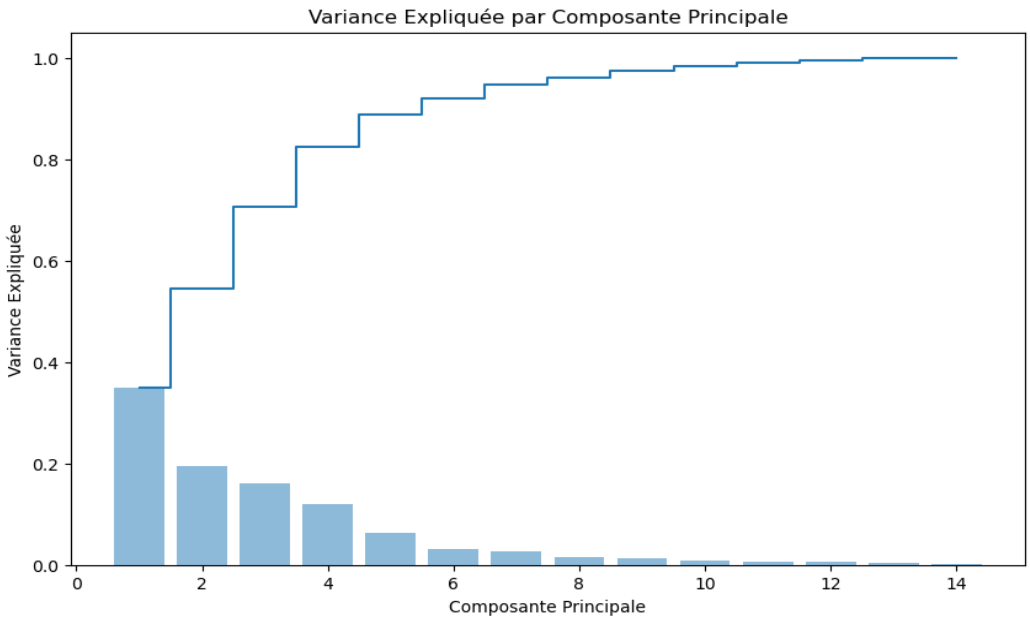
1. **Similarité entre les formules mathématiques des caractéristiques** : Par exemple, **A_max** et le **rapport_minmax** sont liés par la formule A_max / (A_min + 1). De même, le **rapport d'énergie** et le **SNR** sont tous deux basés sur l'intervalle utile pour calculer leurs ratios.

2. **Explication physique** : La variation d'une caractéristique peut avoir une conséquence immédiate sur une autre. Par exemple, **l'énergie moyenne** et le **pourcentage_seuil** calculent le nombre d'amplitudes dépassant un seuil défini. Plus le pourcentage seuil augmente, plus le nombre d'amplitudes importantes augmente, rendant le signal plus énergétique. De même, **TPPZ** et **spectral_centroid** montrent que plus la fréquence dans le barycentre est élevée, plus le nombre de passages par zéro sera grand.

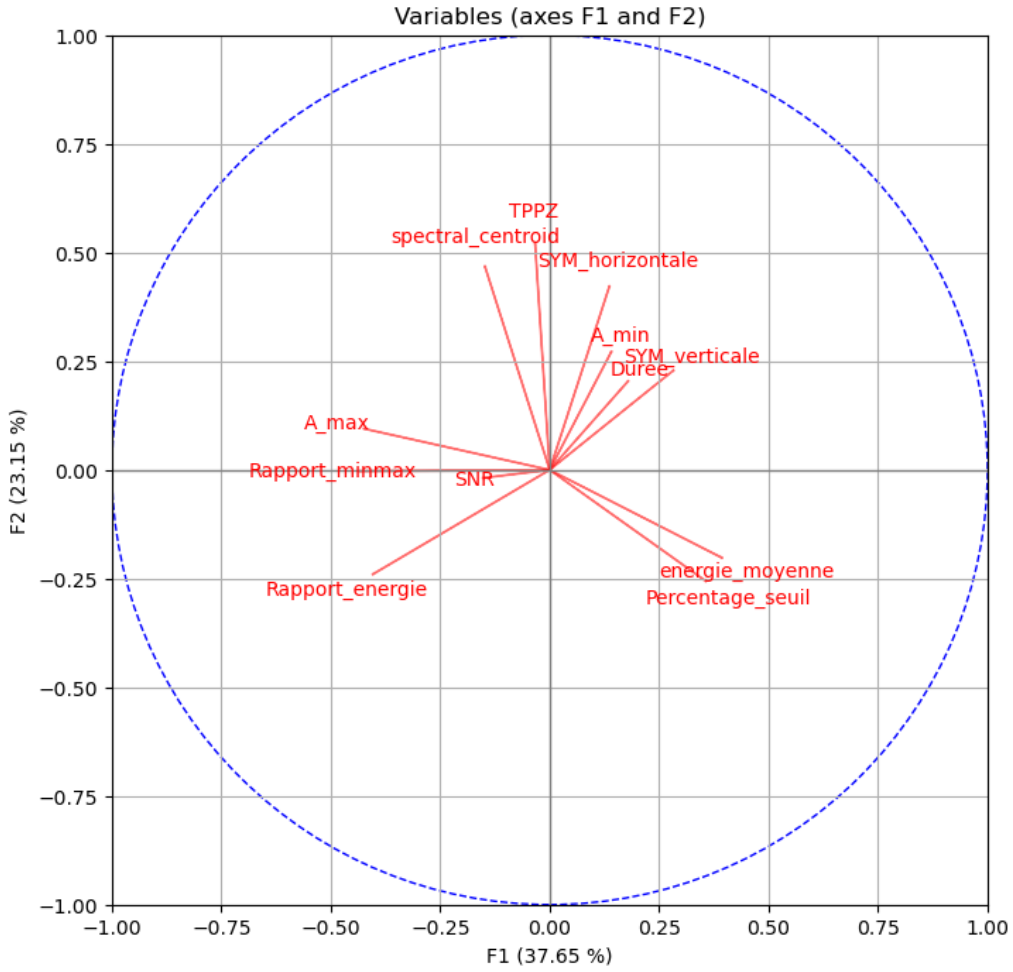
Apparemment, il n'y a pas une forte corrélation entre les caractéristiques étudiées, ce qui est bénéfique pour l'application d'un modèle de machine learning robuste et capable de capturer des relations non linéaires ou subtiles entre les variables pour une meilleur classification.

❖ Analyse en composante principale : PCA

L'Analyse en Composantes Principales (PCA) est une méthode statistique essentielle pour réduire la dimensionnalité des données tout en préservant leur essence. Elle transforme un ensemble de variables potentiellement corrélées en un ensemble de composantes principales non corrélées, facilitant ainsi l'analyse et la modélisation des données. Dans le cadre de mon étude visant à classifier automatiquement les signaux acoustiques, j'ai utilisé PCA pour explorer les corrélations entre les caractéristiques extraites des signaux. Cependant, les résultats dans le graphique ci-dessous ont montré que les deux premières composantes principales expliquent ensemble 60% de la variance totale des caractéristiques, suggérant une faible réduction de la dimensionnalité.



En conséquence, dans mon cas spécifique, l'utilisation de PCA n'est pas nécessaire car mes données ne présentent pas une dimensionnalité suffisamment élevée pour justifier une réduction significative. L'analyse PCA a plutôt servi à analyser les colinéarités entre mes caractéristiques, fournissant ainsi des insights précieux pour l'étape suivante de classification automatique des signaux acoustiques.



Ce schéma illustre l'influence des caractéristiques sur les deux premières composantes principales. La composante principale F1 est représentée sur l'axe des abscisses et F2 sur l'axe des ordonnées. Une caractéristique orthogonale par rapport à un axe indique qu'elle n'a pas d'influence significative sur cette composante particulière. En revanche, une caractéristique alignée avec l'axe d'une composante suggère une corrélation (colinéarité) avec cette composante. Ce schéma permet également de regrouper les caractéristiques en différents groupes selon leur direction et leur magnitude respective, offrant ainsi une visualisation claire des relations entre les variables étudiées.

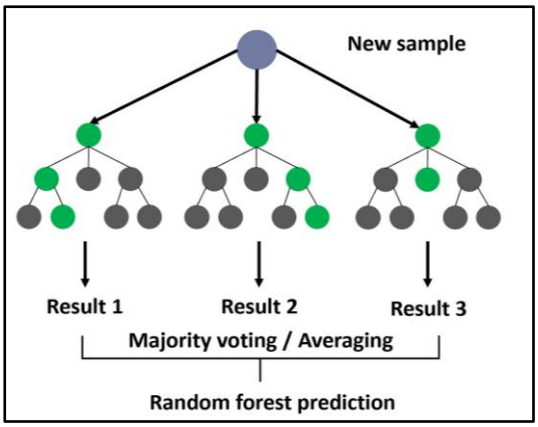
6. Utilisation des modèles de machine learning pour une classification automatique :

➢ Introduction :

Dans cette section nous avons introduit les modèles ML conçus pour faire la classification.

➢ 6.1) Définitions des modèles ML choisis pour la classification :

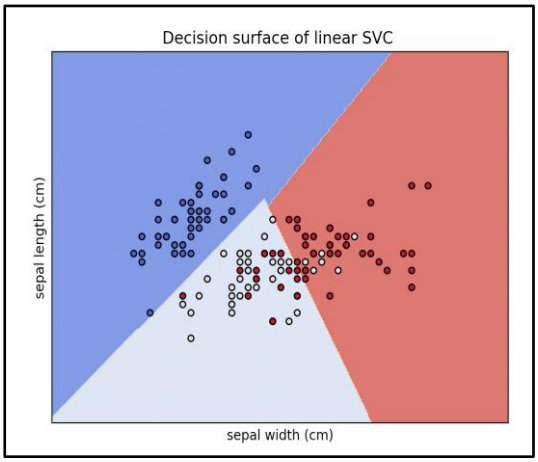
RandomForestClassifier :



Définition : RandomForest Classifier est un modèle d'ensemble basé sur des arbres de décision. Il construit plusieurs arbres de décision lors de l'entraînement et fusionne leurs prédictions pour améliorer la généralisation et le contrôle du surapprentissage. Paramètres principaux :

- **n_estimators** : Le nombre d'arbres dans la forêt.
- **Random_state** : Contrôle la reproductibilité des résultats en initialisant le générateur de nombres aléatoires.

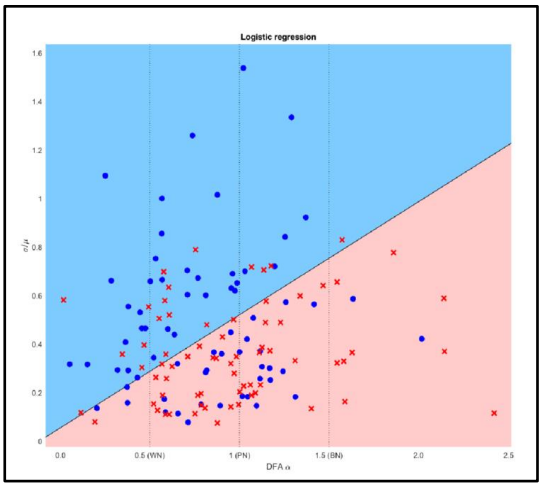
SVC (Support Vector Classifier) :



Définition : SVC est un classificateur basé sur les machines à vecteurs de support. Il utilise des vecteurs de support pour définir des hyperplans séparateurs optimaux entre différentes classes dans un espace de caractéristiques de haute dimension. Paramètres principaux :

- **kernel='rbf'** : Utilise une fonction noyau de type RBF (Radial Basis Function) pour transformer l'espace des caractéristiques.
- **random_state** : Contrôle la reproductibilité des résultats en initialisant le générateur de nombres aléatoires.

LogisticRegression :



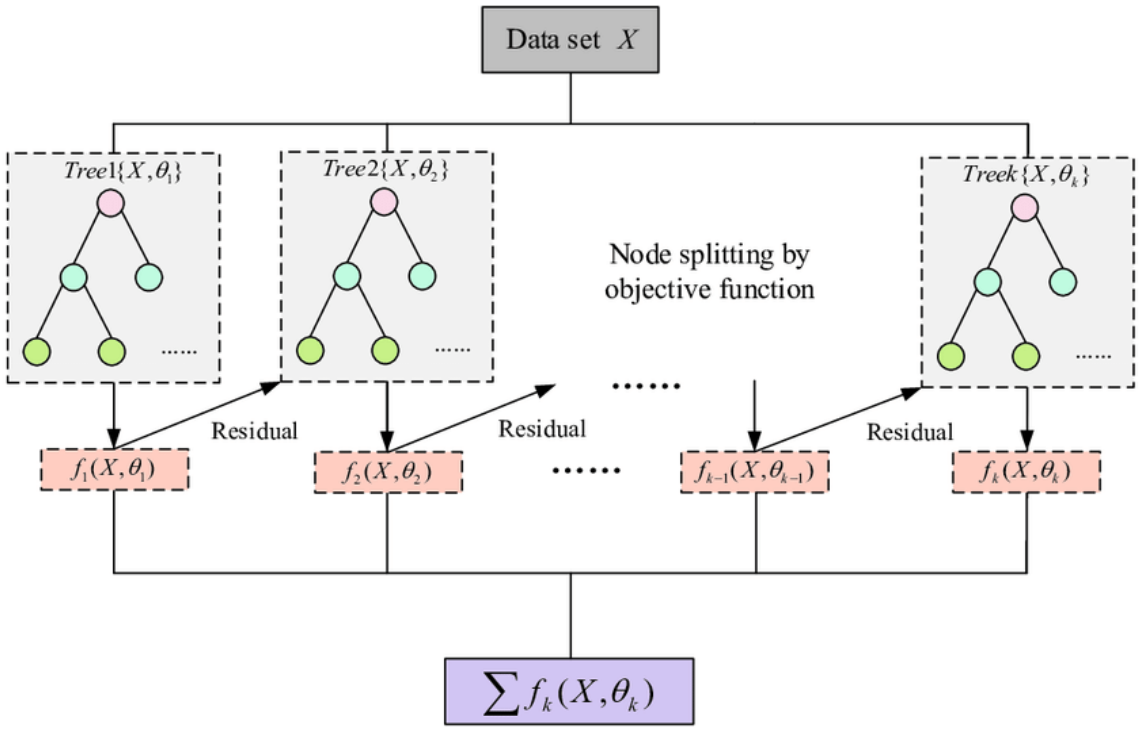
Définition : Logistic Regression est un modèle linéaire utilisé pour la classification binaire et multiclasse. Il modélise la probabilité d'appartenance à chaque classe à l'aide de la fonction logistique. Paramètres principaux :

- **max_iter=1000** : Nombre maximum d'itérations pour la convergence de l'optimisation.
- **random_state** : Contrôle la reproductibilité des résultats en initialisant le générateur de nombres aléatoires.

XGBClassifier (XGBoost Classifier) :

Définition : XGB Classifier est une implémentation de Gradient Boosting qui utilise des arbres de décision pour la classification. Il est optimisé pour la performance et l'évolutivité. Paramètres principaux :

- **objective='multi:softmax'** : Définit l'objectif comme étant la classification multiclasse utilisant la fonction de perte softmax.
- **num_class=3** : Nombre de classes dans le problème de classification.
- **random_state** : Contrôle la reproductibilité des résultats en initialisant le générateur de nombres aléatoires.



XGBoost et Random Forest Classifier sont deux approches populaires en apprentissage automatique pour la classification et la régression. Random Forest construit un ensemble d'arbres de décision indépendants, agrégeant leurs prédictions pour une décision finale, ce qui le rend robuste aux données bruitées et facile à interpréter. En revanche, XGBoost utilise le boosting pour construire les arbres séquentiellement, corrigeant les erreurs des prédictions précédentes pour améliorer la performance sur des ensembles de données de taille moyenne à grande et des caractéristiques complexes ou fortement corrélées. XGBoost nécessite souvent un accord plus minutieux des hyperparamètres, tandis que Random Forest offre une interprétabilité directe avec moins de réglages requis, ce qui en fait un choix adapté pour des ensembles de données diversifiés et plus faciles à analyser individuellement.

➢ 6.2) Les types d'entraînement choisis pour la classification :

❖ Entraînement des modèles sur les événements :

Un événement est défini comme l'ensemble des signaux captés par les microphones correspondant à un même événement. Lors de la construction de la base de données, nous multiplions le nombre de caractéristiques déjà définies par le nombre de microphones disponibles. Ainsi, chaque ligne représente un événement indépendant avec toutes les caractéristiques de ses signaux. De plus, nous ajoutons le facteur de cohérence et d'atténuation à la base de données.

MIC1_A_max	MIC1_A_min	MIC1_energie_moyenne	MIC1_spectral_centroid	MIC1_TPPZ	MIC1_Rapport_energie	MIC1_Rapport_minmax	...	MIC2_Rapport_energie	MIC2_I
37571.419716	0.015267	9.805605e+06	139.417640	392	0.948251	37006.454456	...	0.904775	
26375.373385	0.051212	1.281798e+07	127.069151	118	0.984815	25090.431980	...	0.871260	
32727.197367	0.000681	5.220823e+06	135.194882	161	0.982626	32704.931248	...	0.887775	
30060.375213	0.060430	6.088868e+06	144.278374	554	0.954483	28347.343047	...	0.830296	
32653.461245	0.016583	4.653828e+06	151.422896	807	0.927620	32120.789575	...	0.887361	
...	...	...	...	...	...	...	...	...	
30967.395841	0.000043	4.344659e+07	133.382810	15	0.999997	30966.077900	...	0.999997	
30796.319940	0.000161	4.308551e+07	133.369798	19	0.999998	30791.356010	...	0.999997	
30861.870086	0.000622	4.323526e+07	133.384755	21	0.999997	30842.688936	...	0.999997	
30594.660253	0.000006	4.302741e+07	133.382294	23	0.999997	30594.476309	...	0.999997	
30715.822902	0.001027	4.309411e+07	133.372946	19	0.999998	30684.307160	...	0.999997	

➢ Entraînement des modèles sur les signaux :

Au lieu d'avoir de nombreuses colonnes avec les mêmes caractéristiques, chaque signal sera placé dans une ligne indépendante avec ses propres caractéristiques. Cela permet d'augmenter la quantité de données et d'obtenir une meilleure précision pour les étiquettes. Cependant on peut plus utiliser les caractéristiques liées aux événements, à savoir le facteur de cohérence et le facteur d'atténuation.

MIC	Duree	A_max	A_min	energie_moyenne	spectral_centroid	TPPZ	Rapport_energie	Rapport_minmax	Percentage_seuil	SNR	Fautotest
1	2.160230	37571.419716	0.015267	16.098465	139.417640	278	0.948251	37006.454456	4.615359	1.485737	96
2	1.960255	31292.190595	0.100981	15.853596	139.337505	288	0.947278	28422.096265	4.612476	1.413082	82
3	2.160230	28094.432504	0.065506	16.348314	129.851180	246	0.904775	26367.210739	4.610018	1.283890	156
4	1.960255	38491.665149	0.270383	16.826775	134.337612	252	0.913939	30299.260531	4.617527	1.341113	382
1	2.142357	26375.373385	0.051212	16.366360	127.069151	116	0.984815	25090.431980	4.612320	1.515727	142
...	...	...	...	...	...	...	...	...	...	...	...
2	2.314086	30569.587369	0.000101	17.587855	133.289222	31	0.999997	30566.487341	4.652326	2.370808	-1
1	2.314461	30594.660253	0.000006	17.577348	133.382294	15	0.999997	30594.476309	4.651803	2.465714	-1
2	2.314461	30616.131721	0.000160	17.590306	133.299265	21	0.999997	30611.230962	4.652318	2.416800	-1
1	2.314711	30715.822902	0.001027	17.578897	133.372946	17	0.999998	30684.307160	4.651747	2.465621	-1
2	2.314711	30554.771376	0.000242	17.583848	133.293625	27	0.999997	30547.390910	4.652210	2.395139	-1

➢ 6.3) Résumé pour les performances des modèles pour chaque type d'entraînement :

❖ Etude sur les performances d'un modèle ML :

Dans cette section, nous présentons les performances du modèle à l'aide des métriques suivantes : précision, rappel, F1-score et support. Ces métriques sont utilisées pour évaluer la capacité du modèle à distinguer entre les chutes de blocs et les autres événements acoustiques.

- **Précision** : La précision mesure la proportion des prédictions positives correctes parmi toutes les prédictions positives. Elle est définie comme suit :

$$\text{Précision} = \frac{TP}{TP + FP} \times 100\%$$

où TP représente les vrais positifs et FP les faux positifs.

- **Rappel (recall)** : Le rappel, également connu sous le nom de sensibilité ou taux de vrais positifs, mesure la capacité du modèle à identifier correctement les échantillons positifs. Il est calculé comme suit :

$$\text{Rappel} = \frac{TP}{TP + FN} \times 100\%$$

où le FN représente les faux négatifs.

- **F1-score** : Le F1-score est une mesure combinée de la précision et du rappel, offrant un équilibre entre les deux. Il est particulièrement utile lorsque les classes sont déséquilibrées. Il est défini comme suit :

$$\text{F1-score} = 2 \times \frac{\text{Précision} \times \text{Rappel}}{\text{Précision} + \text{Rappel}}$$

Par exemple, pour la classe 0, un F1-score de 0.88 signifie que le modèle a une bonne performance équilibrée en termes de précision et de rappel pour cette classe.

- **Support** : Le support représente le nombre d'occurrences réelles de chaque classe dans les données de test. Par exemple, pour la classe 0, le support est de 70, ce qui signifie qu'il y a 70 échantillons de chute de bloc dans les données de test.

- **Exactitude (Accuracy)** : L'exactitude mesure la proportion totale de prédictions correctes parmi toutes les prédictions. Elle est définie comme suit :

$$\text{Exactitude} = \frac{TP + TN}{TP + TN + FP + FN} \times 100\%$$

où TN représente les vrais négatifs. Dans notre cas, une exactitude de 0.93 signifie que 93% des prédictions du modèle sont correctes.

- **Macro average** : La moyenne macro (macro avg) calcule la moyenne des métriques pour chaque classe et fournit une vue d'ensemble des performances du modèle sur toutes les classes, sans tenir compte de la proportion des classes.
- **Weighted average** : La moyenne pondérée (weighted avg) calcule la moyenne des métriques pour chaque classe en tenant compte de la proportion de chaque classe dans les données. Cela donne une vision plus précise des performances globales du modèle lorsqu'il y a un déséquilibre des classes.

Ces métriques sont cruciales pour évaluer et comprendre les performances des modèles de détection de chute de bloc, en nous assurant qu'ils sont capables de distinguer efficacement entre les événements de chute de bloc et les autres signaux acoustiques.

❖ Résumé pour les performances des modèles pour l'entraînement sur les événements pour la base 21/24 :

➢ RandomForestClassifier :

```
Accuracy du RandomForestClassifier sur la nouvelle base de données: 0.8695652173913043
Classification Report sur la nouvelle base de données:
              precision    recall  f1-score   support

           0       1.00      0.50      0.67        18
           1       0.62      1.00      0.77        15
           2       1.00      1.00      1.00        36

    accuracy                           0.87        69
   macro avg       0.88      0.83      0.81        69
weighted avg       0.92      0.87      0.86        69

Confusion Matrix sur la nouvelle base de données:
 [[ 9  9  0]
 [ 0 15  0]
 [ 0  0 36]]
```

➢ SVC (Support Vector Classifier) :

```
Accuracy du SVC sur la nouvelle base de données: 0.8985507246376812
Classification Report sur la nouvelle base de données:
              precision    recall  f1-score   support

           0       1.00      0.61      0.76        18
           1       0.68      1.00      0.81        15
           2       1.00      1.00      1.00        36

    accuracy                           0.90        69
   macro avg       0.89      0.87      0.86        69
weighted avg       0.93      0.90      0.90        69

Confusion Matrix sur la nouvelle base de données:
 [[11  7  0]
 [ 0 15  0]
 [ 0  0 36]]
```

➢ LogisticRegression :

```
Accuracy du LogisticRegression sur la nouvelle base de données: 0.8985507246376812
Classification Report sur la nouvelle base de données:
              precision    recall  f1-score   support

           0       1.00      0.61      0.76        18
           1       0.68      1.00      0.81        15
           2       1.00      1.00      1.00        36

    accuracy                           0.90        69
   macro avg       0.89      0.87      0.86        69
weighted avg       0.93      0.90      0.90        69

Confusion Matrix sur la nouvelle base de données:
 [[11  7  0]
 [ 0 15  0]
 [ 0  0 36]]
```

➢ XGBClassifier :

```
Accuracy du XGBClassifier sur la nouvelle base de données: 0.8695652173913043
Classification Report sur la nouvelle base de données:
              precision    recall  f1-score   support

           0       1.00      0.50      0.67        18
           1       0.62      1.00      0.77        15
           2       1.00      1.00      1.00        36

    accuracy                           0.87        69
   macro avg       0.88      0.83      0.81        69
weighted avg       0.92      0.87      0.86        69

Confusion Matrix sur la nouvelle base de données:
 [[ 9  9  0]
 [ 0 15  0]
 [ 0  0 36]]
```

➢ *conclusion* :

Les quatre modèles ont correctement classifié tous les signaux de test de type "autres" et les autotests. Cependant, ils ont rencontré des difficultés à classifier les signaux de chute de blocs. SVC et la régression logistique ont montré une légère amélioration, réussissant à reconnaître 61% des signaux de chute de blocs. Cette dégradation des performances des modèles peut s'expliquer par un mauvais ordre des signaux des microphones dans la base de données. L'importance des signaux en termes de puissance de la chute varie selon le lieu de l'événement. Par exemple, dans un événement composé de quatre signaux, les microphones 1 et 2 peuvent capter des signaux de forte amplitude, tandis que les microphones 3 et 4 captent peu en raison de leur éloignement. Si les caractéristiques des signaux des microphones 1, 2, 3 et 4 sont ordonnées respectivement dans la base de données, mais que dans un autre événement ce sont les microphones 3 et 4 qui captent la chute, l'ordre des microphones restant le même dans la base, cela perturbe les modèles.

❖ **Résumé pour les performances des modèles pour l'entraînement sur les signaux :**

➢ **RandomForestClassifier :**

```
Accuracy sur la nouvelle base de données: 0.9264705882352942
Classification Report sur la nouvelle base de données:
              precision    recall  f1-score   support

           0       1.00      0.79      0.88        70
           1       0.80      1.00      0.89        61
           2       1.00      1.00      1.00        73

    accuracy                           0.93       204
   macro avg       0.93      0.93      0.92       204
weighted avg       0.94      0.93      0.93       204

Confusion Matrix sur la nouvelle base de données:
 [[55 15  0]
 [ 0 61  0]
 [ 0  0 73]]
```

➢ **SVC (Support Vector Classifier):**

```
Accuracy sur la nouvelle base de données: 0.9313725490196079
Classification Report sur la nouvelle base de données:
              precision    recall  f1-score   support

           0       1.00      0.80      0.89        70
           1       0.81      1.00      0.90        61
           2       1.00      1.00      1.00        73

    accuracy                           0.93       204
   macro avg       0.94      0.93      0.93       204
weighted avg       0.94      0.93      0.93       204

Confusion Matrix sur la nouvelle base de données:
 [[56 14  0]
 [ 0 61  0]
 [ 0  0 73]]
```

➢ **LogisticRegression :**

```
Accuracy sur la nouvelle base de données: 0.9166666666666666
Classification Report sur la nouvelle base de données:
              precision    recall  f1-score   support

           0       0.95      0.80      0.87        70
           1       0.81      0.95      0.87        61
           2       1.00      1.00      1.00        73

    accuracy                           0.92       204
   macro avg       0.92      0.92      0.91       204
weighted avg       0.92      0.92      0.92       204

Confusion Matrix sur la nouvelle base de données:
 [[56 14  0]
 [ 3 58  0]
 [ 0  0 73]]
```

➢ XGBClassifier :

```
Accuracy sur la nouvelle base de données: 0.9166666666666666
Classification Report sur la nouvelle base de données:
              precision    recall  f1-score   support

           0       0.98      0.77      0.86        70
           1       0.79      0.98      0.88        61
           2       1.00      1.00      1.00        73

    accuracy                           0.92       204
   macro avg       0.92      0.92      0.91       204
weighted avg       0.93      0.92      0.92       204

Confusion Matrix sur la nouvelle base de données:
 [[54 16  0]
 [ 1 60  0]
 [ 0  0 73]]
```

➢ conclusion :

Après l'application de la data augmentation sur les signaux de chutes de blocs, nous avons entraîné les mêmes modèles de machine learning, cette fois sur les signaux individuels et non sur les événements. Nous avons constaté une amélioration remarquable des performances des quatre modèles.

Les quatre modèles ont dépassé 90 % en termes d'accuracy, le modèle SVC atteignant 93 %. Comme pour l'entraînement sur les événements, les quatre modèles ont correctement classifié les signaux "autres" et les autotests, mais avec une classification nettement améliorée pour les chutes de blocs, atteignant un recall moyen de 79 %.

Pour améliorer davantage ces performances, nous avons envisagé d'utiliser le grid-search afin de trouver les paramètres optimaux pour un meilleur recall. Ce sera le prochain point abordé dans ce rapport.

➢ Optimisation des Modèles par Recherche en Grille:

Pour améliorer les performances de nos modèles de machine learning sur la classification des signaux de chutes de blocs, nous avons mis en œuvre une recherche en grille (grid search) pour optimiser les paramètres des modèles. La procédure est la suivante :

Nous avons commencé par charger et normaliser les données initiales. Les caractéristiques pertinentes ont été extraites et normalisées, puis les classes ont été rééquilibrées à l'aide de SMOTE pour compenser le déséquilibre des données.

Nous avons défini une fonction de scoring axée sur le rappel de la classe 0, représentant les signaux de chute de blocs. Ensuite, nous avons établi des grilles de paramètres pour

chaque modèle, comprenant des paramètres comme le nombre d'estimateurs, la profondeur maximale et les taux d'apprentissage.

Les modèles suivants ont été inclus : RandomForestClassifier, SVC, LogisticRegression, XGBClassifier et LGBMClassifier. Pour chaque modèle, une recherche en grille a été effectuée pour identifier les meilleurs hyperparamètres, maximisant ainsi le rappel de la classe 0.

Les résultats de la recherche en grille sont les suivants :

1. **RandomForestClassifier :**
    - **Paramètres optimaux :** `max_depth: None`, `min_samples_split: 2`, `n_estimators: 200`
    - **Meilleur score de rappel pour la classe 0 :** 0,9768 (±0,0307)
2. **SVC (Support Vector Classifier) :**
    - **Paramètres optimaux :** `C: 10`, `gamma: scale`
    - **Meilleur score de rappel pour la classe 0 :** 0,9818 (±0,0176)
3. **LogisticRegression :**
    - **Paramètres optimaux :** `C: 1`, `solver: liblinear`
    - **Meilleur score de rappel pour la classe 0 :** 0,9702 (±0,0237)
4. **XGBClassifier :**
    - **Paramètres optimaux:** `learning_rate: 0.1`, `max_depth: 3`, `n_estimators: 100`
    - **Meilleur score de rappel pour la classe 0 :** 0,9851 (±0,0218)

Ces résultats montrent que l'ajustement des hyperparamètres via la recherche en grille a permis d'améliorer significativement le rappel pour la classe 0.

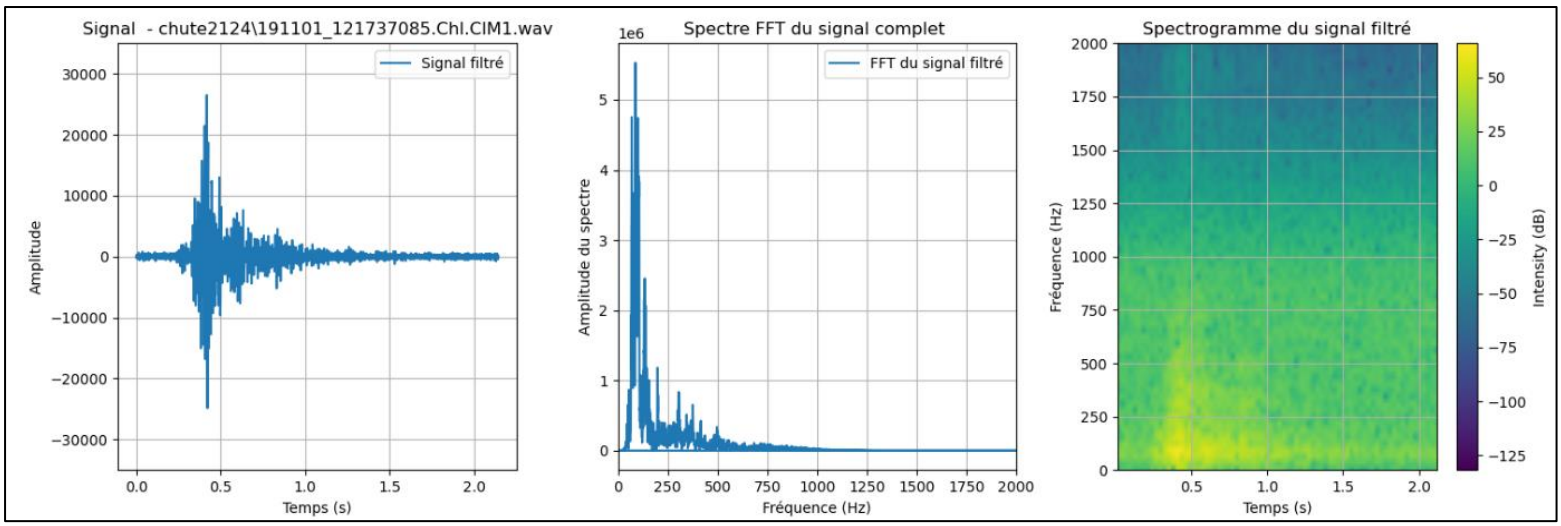
8. Utilisation des modèles de Deep Learning pour une classification automatique :

➢ 8.1) Modèle CNN Spectrogramme : analyse et évaluation :

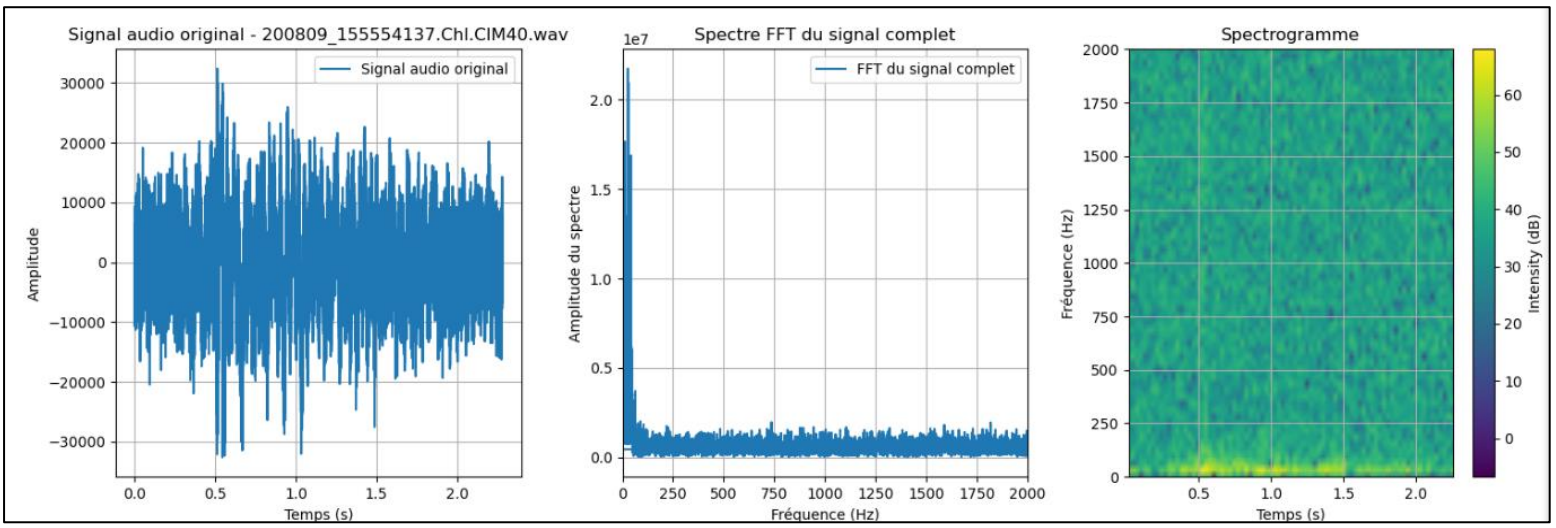
▪ 1. Contexte général :

Un spectrogramme est une représentation graphique qui illustre comment la distribution des fréquences d'un signal varie au fil du temps. C'est une visualisation bidimensionnelle où l'axe horizontal représente le temps, l'axe vertical représente la fréquence, et la couleur ou l'intensité indique l'énergie ou la puissance du signal à chaque fréquence et à chaque instant. Cette représentation est souvent utilisée dans le traitement du signal et l'analyse audio pour visualiser les caractéristiques temporelles et fréquentielles des signaux complexes tels que les sons et les vibrations.

✓ Pour une chute de bloc :



✓ Pour un bruit :



On peut remarquer une différence entre les deux spectrogrammes au niveau de chaque pixel. Cette représentation est particulièrement utile pour l'apprentissage en Convolutional Neural Networks (CNN) car elle permet de capturer des motifs complexes et abstraits présents dans les données audios. Les CNN sont capables d'apprendre des caractéristiques pertinentes à partir de ces spectrogrammes, en analysant les variations spatiales et temporelles qui représentent les structures significatives des signaux audio.

▪ 2. Construction d'une base de spectrogrammes pertinentes et prétraitement des données :

➢ Pour construire une base de données de spectrogrammes, nous avons commencé par collecter des enregistrements audios de trois dossiers différents représentant des catégories d'événements distinctes : "chute de bloc", "parasites" et "autotest". Chaque enregistrement a été lu, prétraité pour supprimer les décalages et filtré avec un filtre passe-haut pour éliminer les fréquences indésirables. Ensuite, nous avons converti les signaux filtrés en spectrogrammes, qui sont des représentations visuelles des signaux dans le domaine fréquentiel, et les avons enregistrés dans une base de données Pandas. Cette base de données a été sauvegardée dans un fichier pickle pour une utilisation ultérieure.

```
        audio_file = os.path.join(folder_path_list[etiquette], concatenated_events[event_id][i])
        Fs, x = audioBasicIO.read_audio_file(audio_file)
        x = remove_offset(x)
        if len(x) > 0:
            signal = x
            filtered_x = highpass_filter(signal, 50, Fs)

            # Calculer le spectrogramme
            f, t, Sxx = spectrogram(filtered_x, Fs)
            Sxx_log = 10 * np.log10(Sxx + 1e-8)  # Convertir en dB

            # Créer une entrée pour le DataFrame
            event_data = {
                "Evenement": event_id,
                "Etiquette": etiquette,
                "Spectrogramme": [Sxx_log]  # Mettre entre crochets pour conserver la structure
            }

            # Ajouter les données de l'événement au DataFrame
            new_database = pd.concat([new_database, pd.DataFrame(event_data)], ignore_index=True)

# Sauvegarder la base de données dans un fichier
new_database.to_pickle(database_name)
print(f"Base de données {database_name} sauvegardée avec succès.")
```

- Pour charger et prétraiter les données de spectrogrammes pour un modèle CNN, il est essentiel de redimensionner tous les spectrogrammes à une taille fixe, généralement (128, 128), pour assurer la compatibilité avec le modèle. Ensuite, les données sont préparées en convertissant les spectrogrammes en tableaux numpy et en appliquant le one-hot encoding aux étiquettes pour l'apprentissage supervisé. Enfin, les données sont divisées en ensembles d'entraînement, de validation et de test pour l'entraînement et l'évaluation du modèle. L'ensemble d'entraînement est utilisé pour ajuster les poids du modèle, tandis que l'ensemble de validation sert à optimiser les hyperparamètres, comme les taux d'apprentissage. Enfin, l'ensemble de test est réservé pour évaluer les performances finales du modèle sur des données inconnues, garantissant ainsi une évaluation impartiale de sa capacité à généraliser.

```
# Diviser les données en ensembles d'entraînement, de validation et de test
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

print(f"Taille de l'ensemble d'entraînement: {X_train.shape}")
print(f"Taille de l'ensemble de validation: {X_val.shape}")
print(f"Taille de l'ensemble de test: {X_test.shape}")
```

```
Taille de l'ensemble d'entraînement: (1929, 128, 128, 1)
Taille de l'ensemble de validation: (413, 128, 128, 1)
Taille de l'ensemble de test: (414, 128, 128, 1)
```

▪ 3. Définition du model CNN et sa compilation :

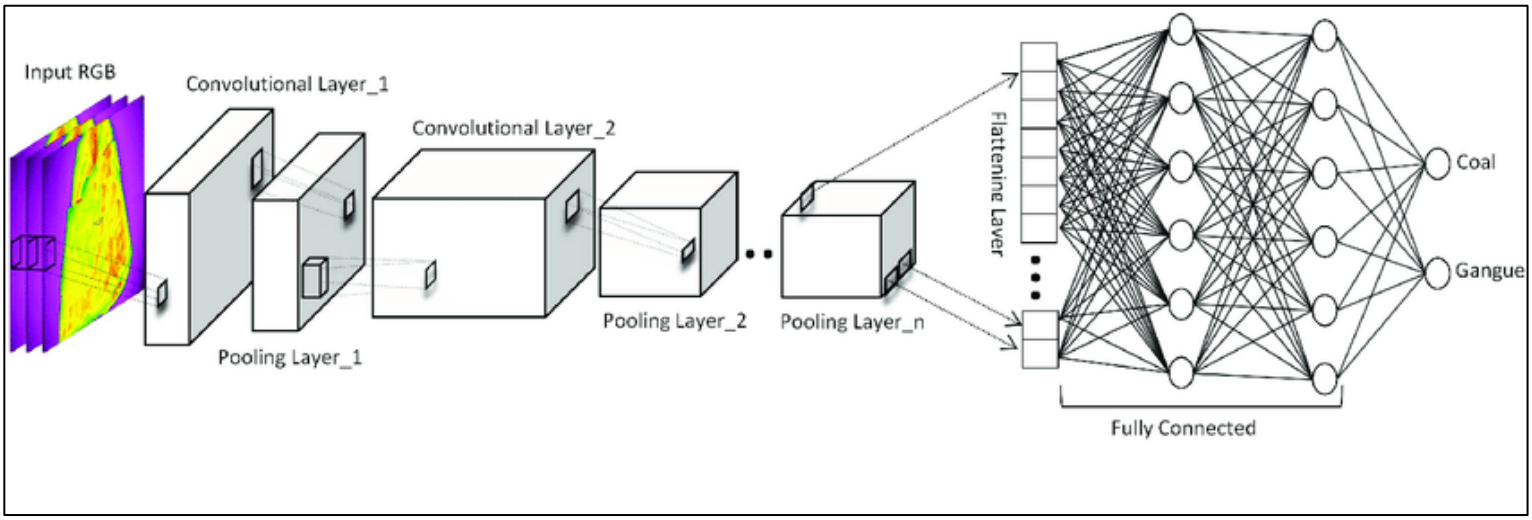
Un modèle Convolutional Neural Network (CNN) est une architecture spécifique de réseau de neurones profonds conçue principalement pour le traitement des images et des données structurées avec une organisation spatiale. Voici les composantes principales d'un CNN :

1. **Couche de convolution :** Cette couche applique des filtres (kernels) sur une région locale de l'image en effectuant une opération de convolution. Chaque filtre extrait des caractéristiques spécifiques de l'image, telles que les bords, les textures ou les motifs plus complexes.
2. **Couche de pooling :** Après la convolution, une couche de pooling est souvent utilisée pour réduire la dimensionnalité de la carte des caractéristiques (feature map) tout en conservant les informations essentielles. Le pooling peut être effectué par des opérations telles que le max pooling (sélection du maximum dans une région) ou la moyenne pooling (calcul de la moyenne dans une région).
3. **Couche d'activation :** Après chaque convolution et pooling, une fonction d'activation est appliquée pour introduire une non-linéarité dans le réseau. La fonction d'activation la plus couramment utilisée est la ReLU (Rectified Linear Unit), qui remplace les valeurs négatives par zéro et laisse les valeurs positives inchangées.
4. **Couche entièrement connectée (ou Dense) :** Les dernières couches d'un CNN sont généralement des couches entièrement connectées qui combinent les caractéristiques extraites par les couches précédentes pour effectuer une classification ou une régression. Ces couches sont similaires aux couches d'un réseau de neurones traditionnel.

5. **Couche de sortie :** La couche de sortie dépend de la tâche spécifique du modèle. Pour la classification, elle utilise une fonction d'activation comme softmax pour obtenir des probabilités sur les classes, tandis que pour la régression, elle peut utiliser une fonction linéaire ou une activation spécifique adaptée à la plage des valeurs attendues.
6. **Fonction de perte :** Pour entraîner le CNN, une fonction de perte (loss function) est utilisée pour mesurer l'écart entre les prédictions du modèle et les valeurs réelles. Pour la classification, la perte croisée (cross-entropy loss) est couramment utilisée, tandis que pour la régression, l'erreur quadratique moyenne (mean squared error) est souvent préférée.
7. **Optimiseur :** L'optimiseur ajuste les poids du réseau neuronal pour minimiser la fonction de perte lors de l'entraînement. Des optimiseurs populaires incluent SGD (Stochastic Gradient Descent), Adam, et RMSProp, qui ajustent les poids du réseau en fonction des gradients calculés par rétropropagation.

En résumé, un CNN est une architecture de réseau de neurones profonds qui exploite la capacité des convolutions locales pour extraire et hiérarchiser des caractéristiques dans des données structurées telles que les images. Il est particulièrement efficace pour le traitement des images en raison de sa capacité à capturer les motifs spatiaux et à réduire le nombre de paramètres requis par rapport aux réseaux de neurones traditionnels.

Représentation du modèle CNN avec ses couches



Dans notre cas, pour coder le modèle CNN, nous avons défini une architecture de réseau de neurones convolutifs en utilisant Keras. Le modèle commence par une couche de convolution avec 32 filtres de taille (3, 3) et une activation ReLU, suivie d'une couche de pooling pour réduire la dimensionnalité spatiale. Nous avons ensuite ajouté des couches de convolution supplémentaires avec 64 et 128 filtres respectivement, chacune suivie de couches de pooling et de dropout pour prévenir le surapprentissage. Après les couches de convolution, nous avons aplati les données pour les passer à une couche dense de 128 neurones avec une activation ReLU et une couche de dropout. Enfin, la sortie du modèle est une couche dense avec une activation softmax pour la classification multiclasse en trois catégories. Le modèle est compilé avec l'optimiseur Adam et la fonction de perte de l'entropie croisée catégorielle, et nous affichons un résumé de l'architecture du modèle.

Layer (type)	Output Shape	Param #
conv2d_83 (Conv2D)	(None, 126, 126, 32)	320
max_pooling2d_83 (MaxPooling2D)	(None, 63, 63, 32)	0
dropout_89 (Dropout)	(None, 63, 63, 32)	0
conv2d_84 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_84 (MaxPooling2D)	(None, 30, 30, 64)	0
dropout_90 (Dropout)	(None, 30, 30, 64)	0
conv2d_85 (Conv2D)	(None, 28, 28, 128)	73,856
max_pooling2d_85 (MaxPooling2D)	(None, 14, 14, 128)	0
dropout_91 (Dropout)	(None, 14, 14, 128)	0
flatten_32 (Flatten)	(None, 25088)	0
dense_64 (Dense)	(None, 128)	3,211,392
dropout_92 (Dropout)	(None, 128)	0
dense_65 (Dense)	(None, 3)	387

Total params: 3,304,451 (12.61 MB)
Trainable params: 3,304,451 (12.61 MB)
Non-trainable params: 0 (0.00 B)

- ## 4. Entrainement et évaluation du modèle :

Pour optimiser l'entraînement du modèle CNN, nous avons défini des callbacks, notamment l'early stopping, qui surveille la perte de validation et arrête l'entraînement si elle ne s'améliore pas après 5 époques consécutives, tout en restaurant les meilleurs poids du modèle. Ensuite, nous avons entraîné le modèle en utilisant les données d'entraînement et de validation sur 50 époques avec une taille de lot de 32. L'early stopping permet de prévenir le surapprentissage et d'assurer que le modèle final utilise les paramètres les plus performants observés durant l'entraînement.

```
# Définir les callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
# Entraîner le modèle
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=50,
    batch_size=32,
    callbacks=[early_stopping])
```

```
Epoch 1/50
61/61 ━━━━━━━━━━━━━━━━━━━━ 23s 272ms/step - accuracy: 0.7548 - loss: 9.9466 - val_accuracy: 0.9104 - val_loss: 0.4173
Epoch 2/50
61/61 ━━━━━━━━━━━━━━━━━━━━ 16s 267ms/step - accuracy: 0.8736 - loss: 0.3503 - val_accuracy: 0.9322 - val_loss: 0.4124
Epoch 3/50
61/61 ━━━━━━━━━━━━━━━━━━━━ 16s 261ms/step - accuracy: 0.9048 - loss: 0.2737 - val_accuracy: 0.7094 - val_loss: 0.6272
Epoch 4/50
61/61 ━━━━━━━━━━━━━━━━━━━━ 16s 262ms/step - accuracy: 0.9163 - loss: 0.2263 - val_accuracy: 0.9467 - val_loss: 0.2108
```

- **Callback** :

Les callbacks dans Keras sont des objets qui peuvent effectuer certaines actions à différentes étapes du processus d'entraînement du modèle. Par exemple, vous pouvez utiliser des callbacks pour arrêter l'entraînement lorsque la précision ne s'améliore plus (early stopping), sauvegarder le modèle à la fin de chaque epoch, ajuster dynamiquement le taux d'apprentissage, etc.

- **Early Stopping :**

L'early stopping est une technique utilisée pour arrêter l'entraînement du modèle lorsque la performance sur les données de validation ne s'améliore plus après un certain nombre d'epochs. Cela permet d'éviter le sur-apprentissage (overfitting) et de trouver le modèle optimal. Voici comment il fonctionne dans Keras : -monitor : La métrique à surveiller, souvent val_loss (la perte sur les données de validation). -patience : Le nombre d'epochs à attendre après la dernière amélioration avant d'arrêter l'entraînement. -restore_best_weights: Si True, restaure les poids du modèle à l'epoch avec la meilleure performance de validation.

- **Epochs :**

Une epoch est une passe complète à travers l'ensemble de données d'entraînement. Pendant une epoch, le modèle voit chaque échantillon de l'ensemble de données exactement une fois. Le nombre d'epochs est un hyperparamètre que vous devez définir. Par exemple, si vous définissez 50 epochs, cela signifie que le modèle verra l'ensemble de données d'entraînement 50 fois.

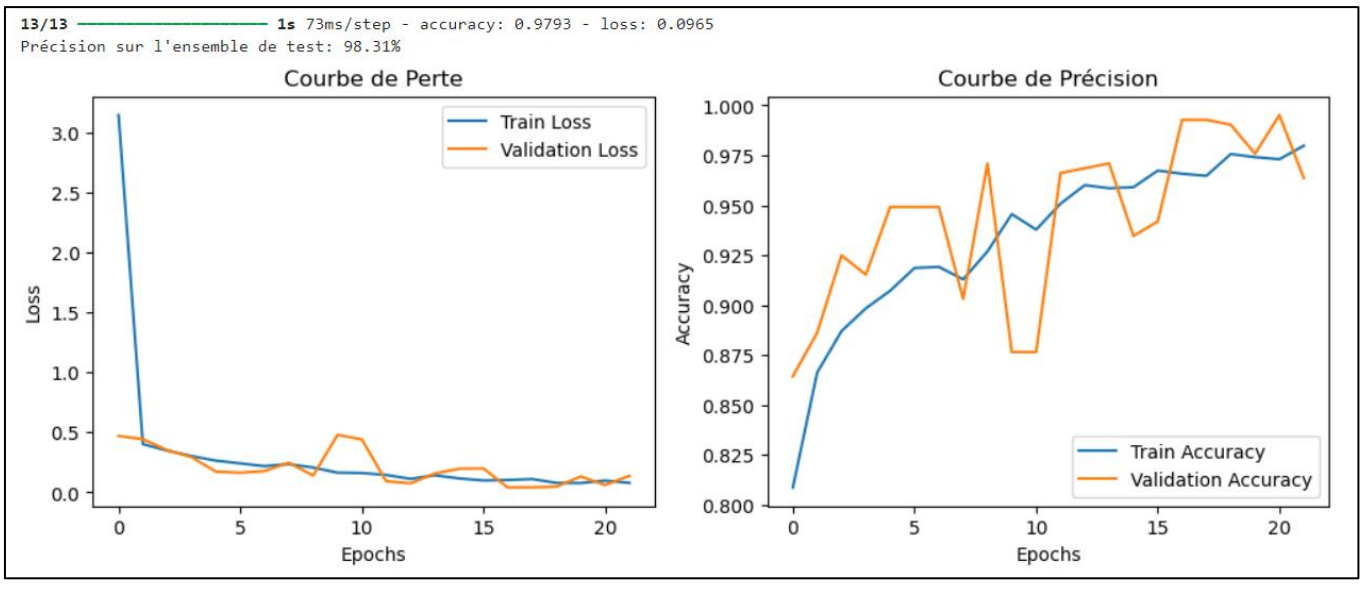
- **Batch Size :**

Le batch size est le nombre d'échantillons d'entraînement passés à travers le réseau avant de mettre à jour les poids du modèle. Pendant l'entraînement, l'ensemble de données est divisé en petits lots (batches), et le modèle est mis à jour à la fin de chaque lot. Un batch size de 32 signifie que le modèle sera mis à jour après avoir vu 32 échantillons.

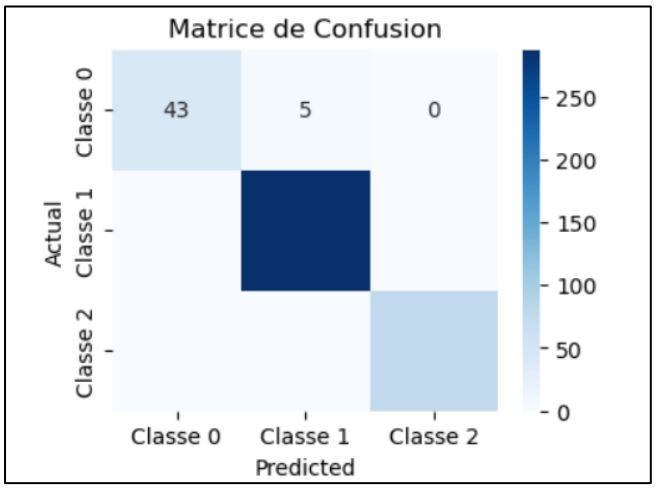
- **Poids** :

Un poids (ou coefficient) dans un réseau de neurones est un paramètre qui est ajusté lors de l'entraînement du modèle pour optimiser les prédictions. Les poids sont utilisés pour déterminer l'importance relative de chaque caractéristique d'entrée et comment ces caractéristiques sont combinées pour produire la sortie.

- **5.Résumé sur les performances du modèle sur l'ensemble deTest :**



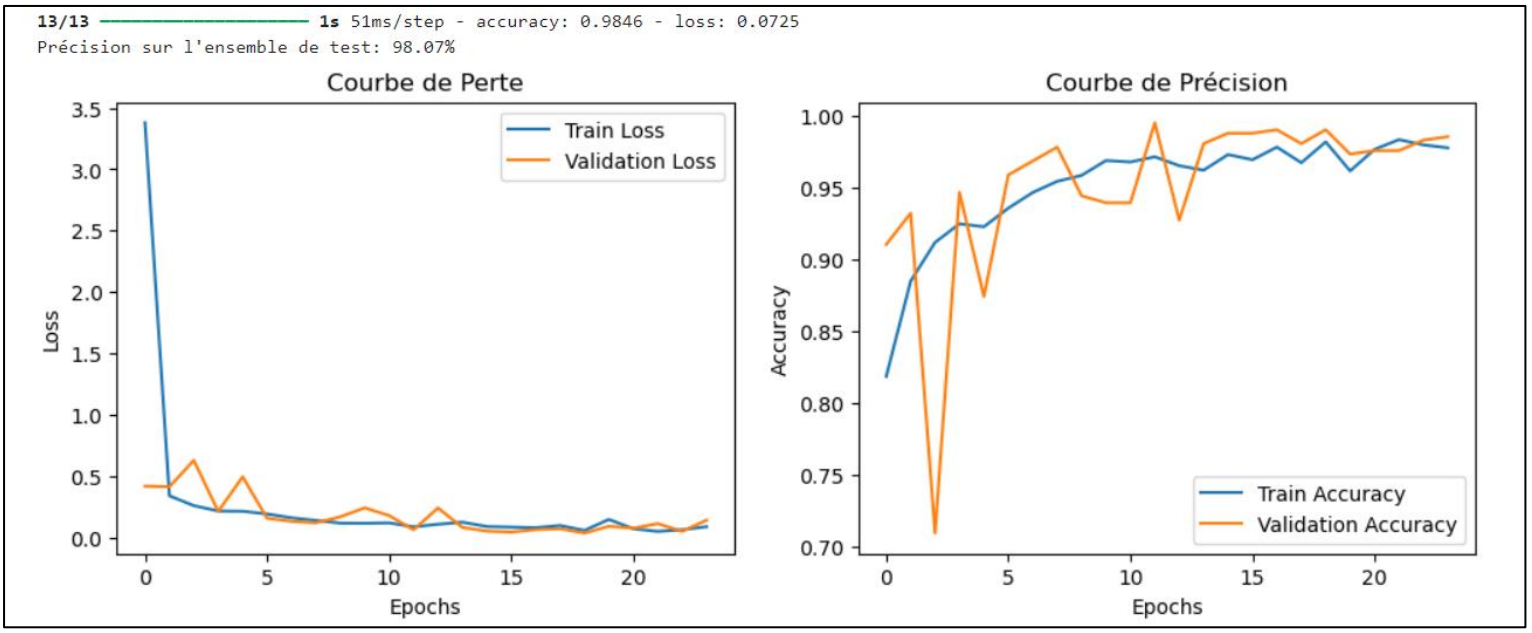
	precision	recall	f1-score	support
Classe 0	0.98	0.90	0.93	48
Classe 1	0.98	0.99	0.99	289
Classe 2	0.99	1.00	0.99	77
accuracy			0.98	414
macro avg	0.98	0.96	0.97	414
weighted avg	0.98	0.98	0.98	414



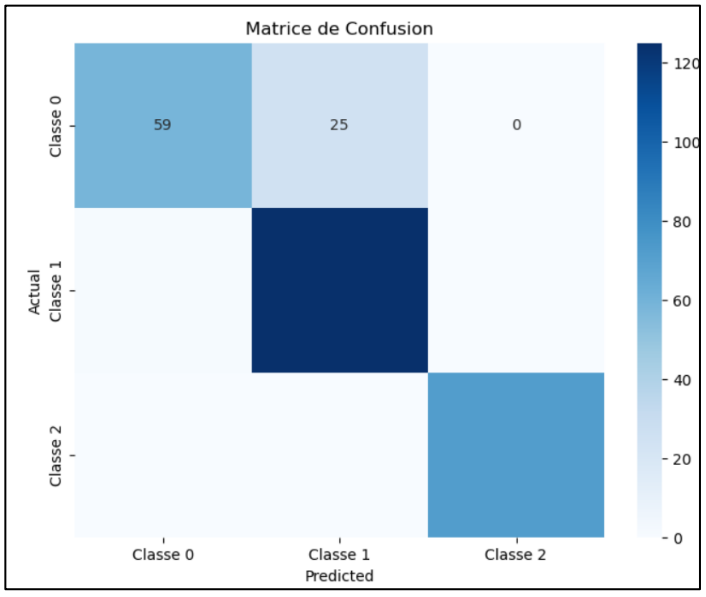
Résumé :

Le modèle a atteint une précision de 98 % sur l'ensemble de test, ce qui est plutôt satisfaisant, avec un rappel de 90 % pour la première classe. Cela signifie qu'il a réussi à identifier 90 % des chutes de blocs, ayant manqué 5 cas sur 48. Ces résultats montrent que le CNN basé sur les spectrogrammes est un modèle très performant, même avec une base de données relativement petite pour le Deep learning. Il reste à tester la nouvelle base de 2124 données sur ce modèle afin d'évaluer ses performances sur des données inédites.

- **6. Résumé sur les performances du modèle sur la nouvelle base 21/24 :**



	precision	recall	f1-score	support
Classe 0	0.98	0.70	0.82	84
Classe 1	0.83	0.99	0.91	126
Classe 2	1.00	1.00	1.00	72
accuracy			0.91	282
macro avg	0.94	0.90	0.91	282
weighted avg	0.92	0.91	0.90	282



Résumé :

Après avoir testé le modèle sur la nouvelle base de données, il a réussi à maintenir des performances relativement satisfaisantes, malgré le fait qu'il n'ait pas été entraîné sur les 2124 nouvelles données. Le rappel pour la première classe a toutefois chuté de 90 % à 70 %, ce qui est compréhensible, car la nouvelle base contient des chutes de blocs difficiles à détecter, même à l'oreille. De plus, même les modèles d'apprentissage automatique avec des caractéristiques bien définies n'ont pas réussi à identifier ces signaux subtils. Dans l'ensemble, le modèle a tout de même montré de bonnes performances.

➢ 8.2) Modèle CNN Temporel : analyse et évaluation :

▪ 1. Contexte général :

Pour le modèle CNN temporel, les données ont été préparées à partir d'une base de données contenant des signaux temporels issus de divers événements. Chaque signal a été prétraité pour supprimer le bruit et filtrer les fréquences non pertinentes, assurant ainsi une représentation propre des événements. Le modèle CNN est défini avec plusieurs couches convolutionnelles suivies de couches de pooling pour extraire les caractéristiques temporelles importantes. La structure comprend également des couches de dropout pour prévenir le surapprentissage, suivies de couches entièrement connectées pour la classification multi-classe. Le modèle est compilé avec l'optimiseur Adam et une perte de catégorie croisée sparse, tandis que l'évaluation se concentre sur la précision comme métrique principale. Les performances du modèle sont évaluées sur un ensemble de test distinct, avec une analyse approfondie incluant des courbes de perte et de précision pour suivre l'apprentissage du modèle au fil des epochs. De plus, une matrice de confusion est générée pour visualiser et quantifier les prédictions du modèle par rapport aux étiquettes réelles, fournissant ainsi un aperçu complet de sa capacité à classifier correctement les événements temporels.

▪ 2. Construction d'une base pertinente et prétraitement des données :

```
print(f"Taille de l'ensemble d'entraînement: {X_train.shape}")
print(f"Taille de l'ensemble de validation: {X_val.shape}")
print(f"Taille de l'ensemble de test: {X_test.shape}")
```

```
Taille de l'ensemble d'entraînement: (985, 20000, 1)
Taille de l'ensemble de validation: (211, 20000, 1)
Taille de l'ensemble de test: (212, 20000, 1)
```

Nous disposons de 985 signaux, chacun composé de 20 000 valeurs, pour l'entraînement du modèle, ainsi que de 211 signaux pour le paramétrage et la validation, et de 212 signaux de test. 20 000 valeurs représente environ 4.5 seconde pour une fréquence de 4400 Hz.

- **3. Définition du model CNN :**

Layer (type)	Output Shape	Param #
conv1d_9 (Conv1D)	(None, 4998, 32)	128
max_pooling1d_9 (MaxPooling1D)	(None, 2499, 32)	0
dropout_7 (Dropout)	(None, 2499, 32)	0
conv1d_10 (Conv1D)	(None, 2497, 64)	6,208
max_pooling1d_10 (MaxPooling1D)	(None, 1248, 64)	0
dropout_8 (Dropout)	(None, 1248, 64)	0
conv1d_11 (Conv1D)	(None, 1246, 128)	24,704
max_pooling1d_11 (MaxPooling1D)	(None, 623, 128)	0
dropout_9 (Dropout)	(None, 623, 128)	0
flatten_4 (Flatten)	(None, 79744)	0
dense_8 (Dense)	(None, 128)	10,207,360
dropout_10 (Dropout)	(None, 128)	0
dense_9 (Dense)	(None, 3)	387

Total params: 10,238,787 (39.06 MB)
Trainable params: 10,238,787 (39.06 MB)
Non-trainable params: 0 (0.00 B)

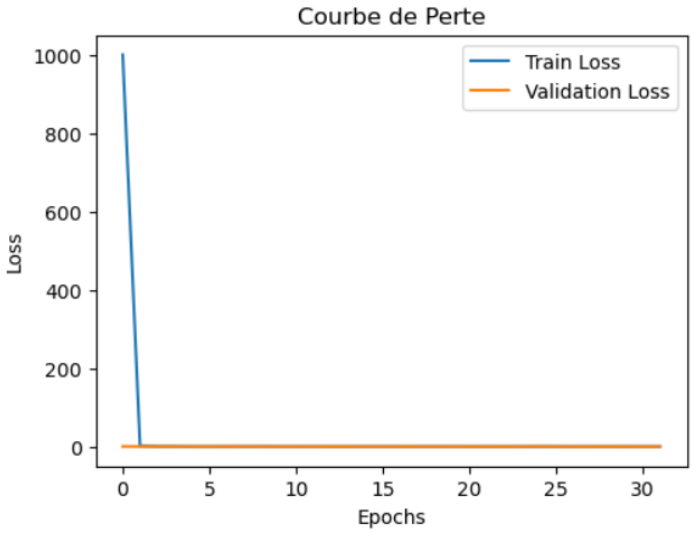
Ce model se compose de plusieurs couches, chacune avec des caractéristiques spécifiques

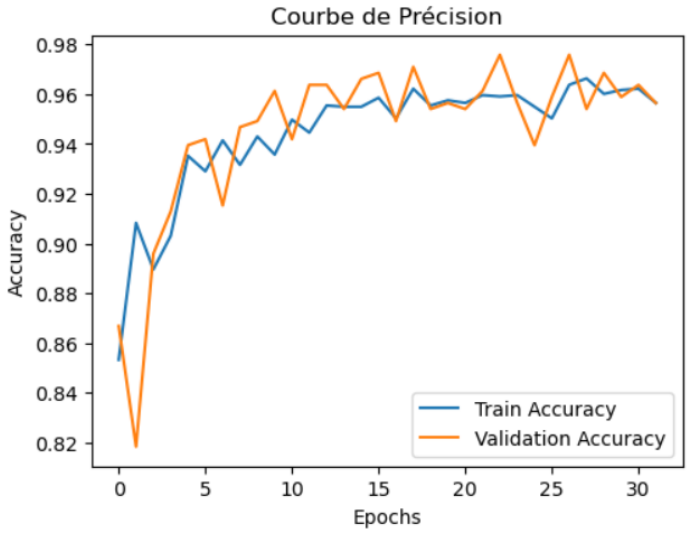
1. **Conv1D et MaxPooling1D** : Le modèle commence par une couche de convolution 1D avec 32 filtres, suivie d'une couche de max-pooling 1D, réduisant la dimension spatiale de moitié. Cette séquence est répétée avec des convolutions 1D supplémentaires (64 et 128 filtres respectivement) et des couches de max-pooling.
2. **Dropout** : Après chaque couche de max-pooling, une couche de dropout est ajoutée pour prévenir le surapprentissage en ignorant aléatoirement une fraction des neurones pendant l'entraînement.
3. **Flatten** : Une couche de flatten convertit les données 3D en un vecteur 1D pour les préparer à l'entrée dans les couches denses.
4. **Dense** : Une couche dense avec 128 neurones et une fonction d'activation ReLU suit la couche flatten, ce qui permet de capturer des interactions complexes entre les caractéristiques extraites par les couches convolutionnelles. Une autre couche de dropout est appliquée ici pour encore réduire le surapprentissage.
5. **Output Dense** : La couche de sortie est une couche dense avec 3 neurones, correspondant probablement à une classification en trois catégories, utilisant une fonction d'activation softmax.

Ce modèle comporte un total de 10,238,787 paramètres, tous étant entraînables. La taille totale du modèle est de 39.06 Mo.

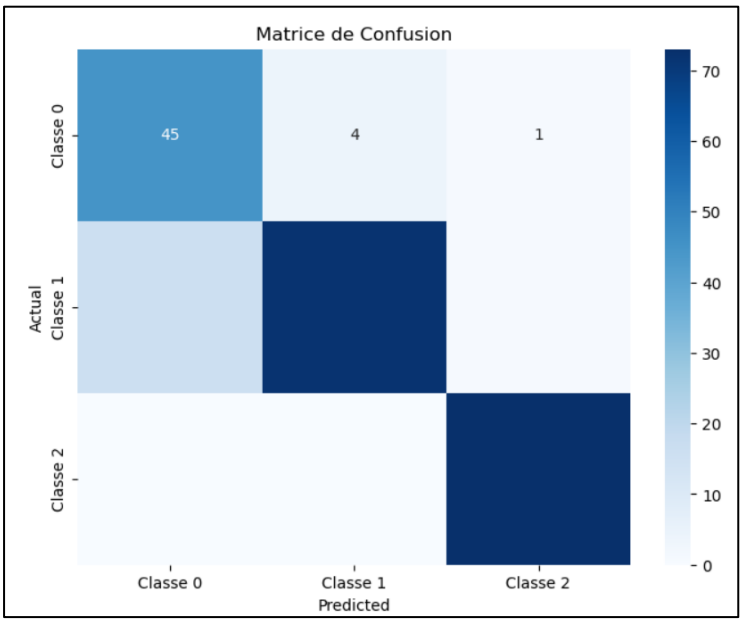
▪ 4.Entrainement et évaluation du modèle sur l'ensemble de Test :

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9757 - loss: 0.0928
Précision sur l'ensemble de test: 97.34%
```





	precision	recall	f1-score	support
Classe 0	0.74	0.90	0.81	50
Classe 1	0.95	0.81	0.87	89
Classe 2	0.97	1.00	0.99	73
accuracy			0.90	212
macro avg	0.89	0.90	0.89	212
weighted avg	0.91	0.90	0.90	212

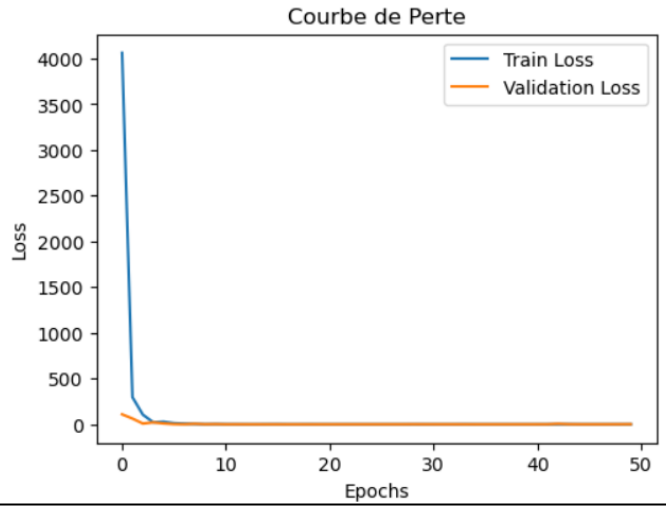


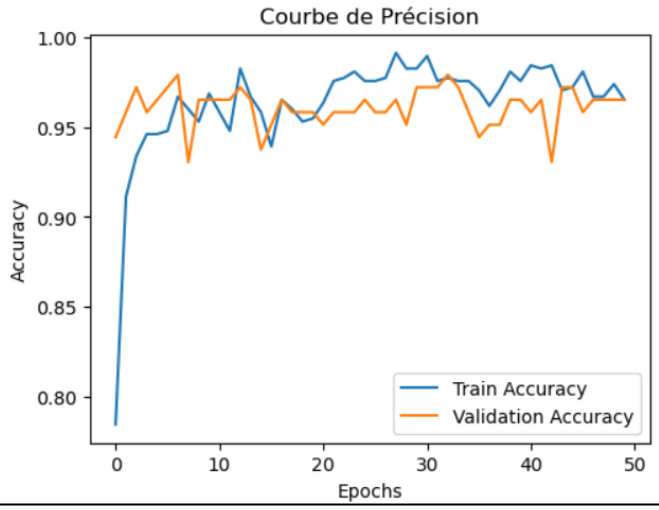
Résumé :

Le modèle a atteint une précision de 97 % sur l'ensemble de test, ce qui est plutôt satisfaisant, avec un rappel de 90 % pour la première classe similaire au CNN spectrogramme. Ces résultats montrent que le CNN basé sur la représentation temporelle est aussi un modèle très performant, même avec une base de données relativement petite pour le Deep Learning. Il reste à tester la nouvelle base de 2124 données sur ce modèle afin d'évaluer ses performances sur des données inédites.

- **5.Entrainement et évaluation du modèle sur la nouvelle base21/24 :**

```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.8194 - loss: 31.1304
Précision sur l'ensemble de test: 86.52%
```

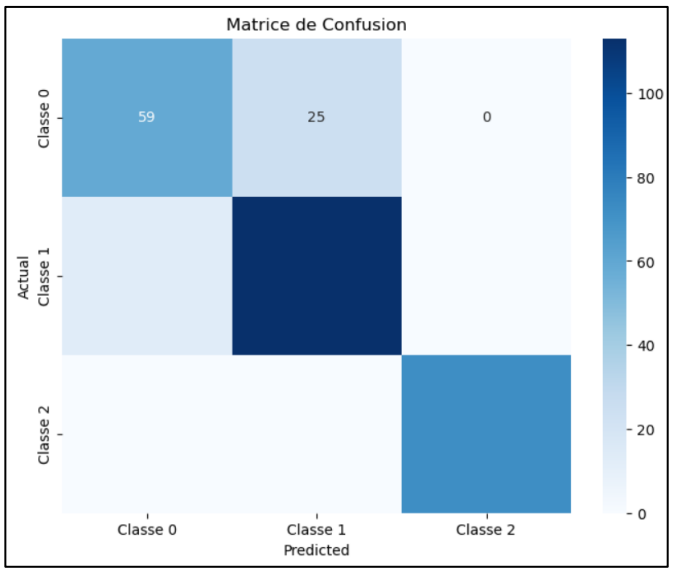





```
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
              precision    recall  f1-score   support

     Classe 0       0.82      0.70      0.76        84
     Classe 1       0.82      0.90      0.86       126
     Classe 2       1.00      1.00      1.00        72

    accuracy                           0.87       282
   macro avg       0.88      0.87      0.87       282
weighted avg       0.87      0.87      0.86       282
```



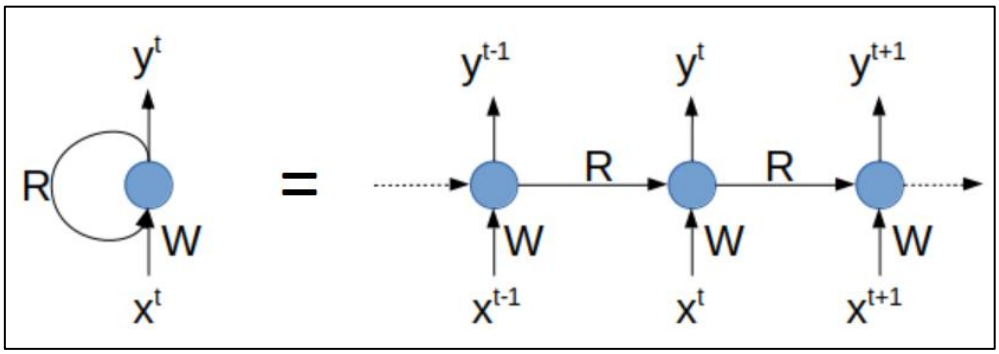
Résumé :

Après avoir testé le modèle sur la nouvelle base de données, il a réussi à maintenir des performances relativement satisfaisantes de 86%, malgré le fait qu'il n'ait pas été entraîné sur les 2124 nouvelles données. Le rappel pour la première classe a toutefois chuté de 90 % à 70 %, ce qui est compréhensible, car la nouvelle base contient des chutes de blocs difficiles à détecter, même à l'oreille. De plus, même les modèles d'apprentissage automatique avec des caractéristiques bien définies n'ont pas réussi à identifier ces signaux subtils. Dans l'ensemble, le modèle a tout de même montré de bonnes performances mais pas comme le CNN spectrogramme. cela est du au fait que le spectrogramme contient plus d'informations fréquentielles que la présentation temporelle.

➢ 8.3) Modèle RNN Temporel : analyse et évaluation :

▪ 1. Contexte général :

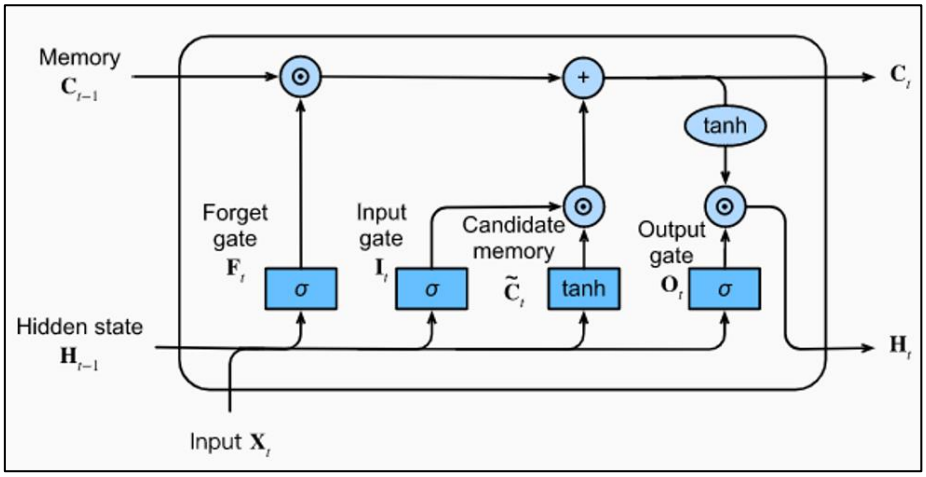
Un modèle récurrent simple est un type de réseau de neurones conçu pour traiter des séquences de données en utilisant des mécanismes de mémoire. Ces modèles sont particulièrement efficaces pour les données temporelles et séquentielles, où chaque élément de la séquence dépend des éléments précédents. Voici une définition plus détaillée :



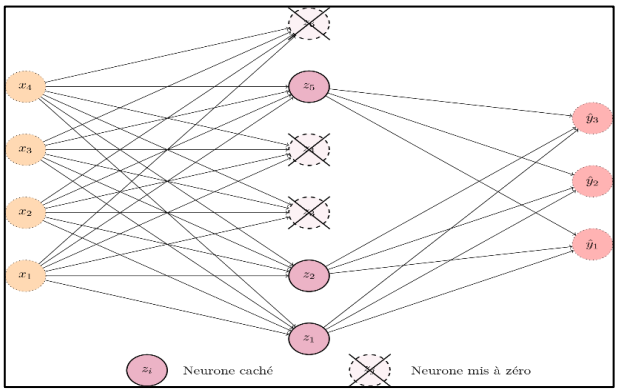
Un modèle récurrent simple est constitué de quelques couches de neurones récurrents, comme les LSTM (Long Short-Term Memory) ou les GRU (Gated Recurrent Units), suivies par une ou deux couches denses (fully connected) pour la classification finale.

Structure Typique

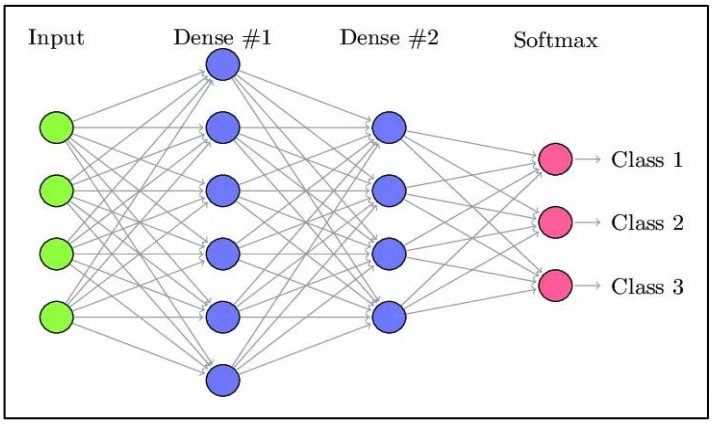
1. **Couche LSTM/GRU**: Ces couches capturent les dépendances temporelles dans les séquences de données. Les LSTM et GRU sont des variantes des réseaux de neurones récurrents (RNN) classiques, spécialement conçues pour surmonter le problème des dépendances à long terme en conservant des informations sur des périodes plus longues.



2. **Couche Dropout** : Une couche Dropout est souvent ajoutée après la couche récurrente pour éviter le surapprentissage. Elle fonctionne en désactivant de manière aléatoire un certain pourcentage des neurones pendant l'entraînement.



3. **Couche Dense (Fully Connected)** : Cette couche prend les sorties de la couche récurrente et les passe à travers une série de neurones entièrement connectés pour effectuer la classification. Une fonction d'activation, comme ReLU (Rectified Linear Unit), est souvent utilisée pour introduire de la non-linéarité.



4. **Couche de Sortie** : La couche finale utilise une fonction d'activation softmax pour produire une probabilité pour chaque classe, permettant la classification des séquences.

$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Conclusion

Les RNN sont particulièrement adaptés à la classification des signaux acoustiques grâce à leur capacité à modéliser les dépendances temporelles dans les données séquentielles. Les variantes avancées des RNN, comme les LSTM et les GRU, sont souvent utilisées pour améliorer la performance, en capturant des relations à long terme dans les signaux acoustiques.

- **2. Les difficultés rencontrées :**

Parmi les difficultés rencontrées dans le développement du RNN pour la classification, la durée d'entraînement excessive est l'une des plus importantes. Au début, nous avons construit une base de données des signaux temporels pour les trois classes : chute de bloc, autres, et autotest, avec une fréquence d'échantillonnage de 4400 Hz et une durée fixée à 10 secondes, soit 44000 valeurs par signal. La taille de l'ensemble d'entraînement était de (1626, 44000). Nous avons ensuite créé un modèle RNN basé sur LSTM pour l'entraînement.

Les LSTM (Long Short-Term Memory) sont une variante des réseaux de neurones récurrents (RNN) spécialement conçue pour résoudre le problème des dépendances à long terme. Ils peuvent conserver des informations sur de longues périodes, ce qui les rend particulièrement efficaces pour traiter des séquences temporelles complexes.

Cependant, en raison de la complexité du modèle et de son nombre de paramètres dépassant les 50 000, nous n'avons pas réussi à effectuer l'entraînement. Chaque epoch nécessitait au moins une heure, ce qui signifie qu'un seul parcours complet de l'ensemble d'entraînement prenait une heure. Avec un minimum de 50 epochs fixés, l'entraînement total aurait pris 50 heures.

```
# Entraînement du modèle
history = model.fit(X_train, y_train_one_hot, epochs=50, batch_size=32, validation_data=(X_val, y_val_one_hot))
```

```
Epoch 1/50
 3/42 ——————————— 1:18:49 121s/step - accuracy: 0.4236 - loss: 0.6897
```

- **3. Des solutions envisageables :**

Pour résoudre ce problème, nous avons adopté une stratégie pour réduire le temps d'entraînement.

Premièrement, nous avons supprimé toute la classe des autotests, qui comptait environ 600 signaux. Ces signaux étant faciles à détecter, ils n'avaient pas la priorité pour la classification automatique.

Deuxièmement, nous avons rééchantillonné les signaux de 4400 Hz à 1500 Hz, ce qui a permis de réduire la quantité de valeurs par signal de 44000 à 15000. Cela a significativement diminué le nombre total de données, soit une réduction de $600 \times 44000 + 1300 \times (44000 - 15000) = 64\,100\,000$ valeurs pour la base de données d'entraînement.

```
Taille de l'ensemble d'entraînement: (1326, 15000)
Taille de l'ensemble de validation: (442, 15000)
Taille de l'ensemble de test: (442, 15000)
```

Troisièmement, nous avons adopté un modèle RNN plus simple avec un nombre de paramètres réduit à 15000. Et nous avons diminué le nombre d'Epochs de 50 à 25 .Ces modifications ont permis de réduire considérablement le temps d'entraînement, tout en maintenant la performance du modèle.

- **4. Définition du model RNN :**

Model: "sequential_4"

Layer (type)	Output Shape	Param #
lstm_8 (LSTM)	(None, 15000, 32)	4,352
dropout_8 (Dropout)	(None, 15000, 32)	0
lstm_9 (LSTM)	(None, 32)	8,320
dropout_9 (Dropout)	(None, 32)	0
dense_8 (Dense)	(None, 16)	528
dense_9 (Dense)	(None, 2)	34

Total params: 13,234 (51.70 KB)
Trainable params: 13,234 (51.70 KB)
Non-trainable params: 0 (0.00 B)

Le code définit un modèle de réseau de neurones récurrents (RNN) en utilisant Keras. Il utilise l'API `Sequential` pour empiler les couches, commençant par une couche LSTM avec 32 unités qui retourne des séquences, suivie d'une couche de Dropout pour éviter le surapprentissage. Ensuite, une deuxième couche LSTM avec 32 unités est ajoutée, suivie d'une autre couche de Dropout. Le modèle inclut ensuite une couche Dense avec 16 neurones et une activation ReLU, et se termine par une couche Dense avec une activation softmax pour la classification des catégories. Le nombre total de parmaetres est 13 234. Le modèle est compilé avec l'optimiseur Adam et une perte `categorical_crossentropy`, et il est configuré pour suivre la métrique de précision.

$$L = -\frac{1}{m}\sum_{i=1}^{m} y_i \cdot \log(\hat{y}_i)$$

- **5. Entrainement et évaluation du modèle :**

L'entraînement du modèle a duré environ 6 heures. Après l'évaluation sur l'ensemble de test, il s'est avéré que le modèle n'a pas correctement classifié les chutes de bloc. Plusieurs facteurs pourraient en être la cause, notamment le manque de précision dû à la réduction de la fréquence d'échantillonnage de 4400 Hz à 1500 Hz, ainsi que l'inadéquation possible du modèle et de ses paramètres pour ce type de signaux.

Pour remédier à cela et réduire le temps d'entraînement, nous avons adopté une autre approche. Nous avons utilisé une fonction d'intervalle utile qui extrait uniquement la partie utile du signal, en fixant la durée à 5 secondes tout en maintenant la même fréquence d'échantillonnage. De plus, nous avons simplifié le modèle en optant pour un RNN plus basique avec moins de paramètres.

Model: "sequential_14"

Layer (type)	Output Shape	Param #
masking_5 (Masking)	(None, 22000, 1)	0
simple_rnn_5 (SimpleRNN)	(None, 32)	1,088
dense_27 (Dense)	(None, 16)	528
dense_28 (Dense)	(None, 3)	51

Total params: 1,667 (6.51 KB)
Trainable params: 1,667 (6.51 KB)
Non-trainable params: 0 (0.00 B)

Conclusion :

Le modèle n'a pas réussi à classer les signaux de chutes de blocs, malgré de nombreuses tentatives de modification. Je pense que ce modèle n'est pas adapté pour effectuer une classification automatique entre les chutes de blocs et les non-chutes de blocs. Par conséquent, j'ai abandonné ce type de modèle. Je ne sais pas pourquoi ce modèle ne fonctionne pas.

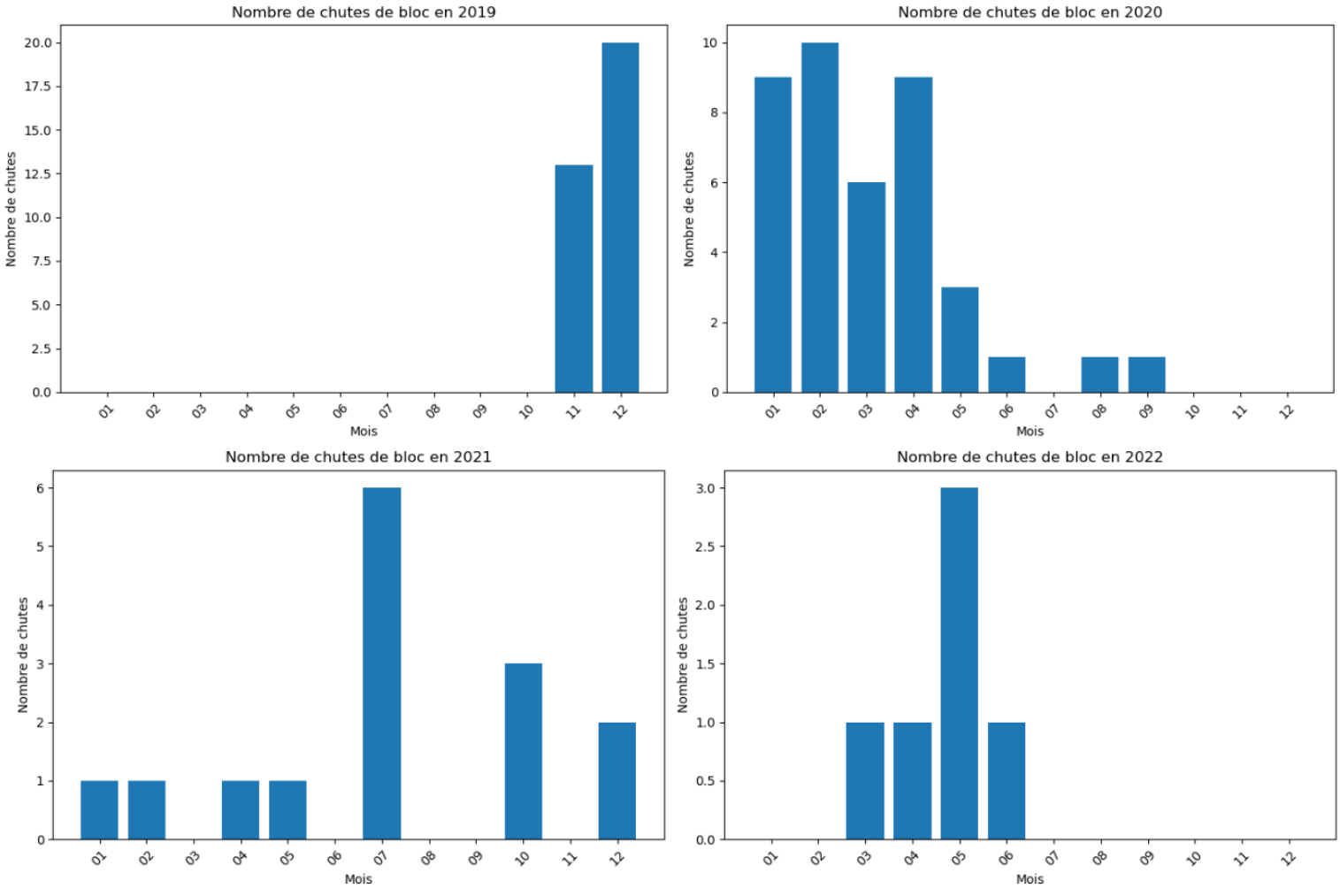
9. Analyse des corrélations pluviométriques :

- **Contexte général :**

Après avoir collecté les données pluviométriques du site de Château Landon pour les années 2019, 2020, 2021, 2022, 2023 et 2024 dans un fichier Excel, nous avons visualisé les précipitations mensuelles et quotidiennes pour ces années. Ensuite, nous avons extrait les dates des événements de chutes de blocs en utilisant un algorithme nommé `date_extract (filename)`. Cela nous a permis de construire une base de données des chutes de blocs avec leurs dates correspondantes. Nous avons ensuite visualisé ces résultats pour chaque mois et chaque jour de ces années. Ce qui est quelque peu ambigu, c'est que nous n'avons enregistré aucune chute de bloc depuis 2022.

- **Visualisation et analyse des résultats :**

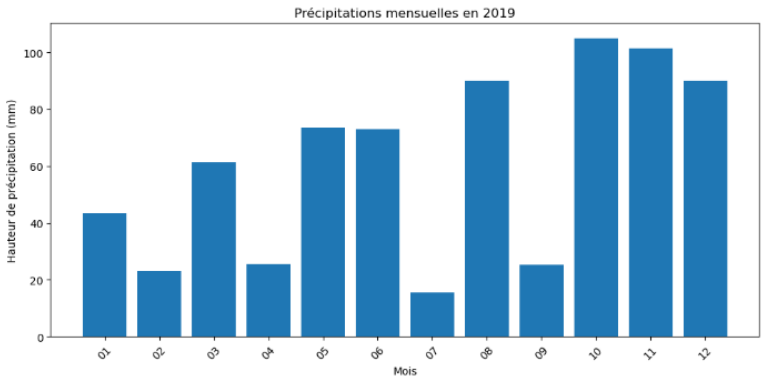
1. Visualisation des chutes de bloc :

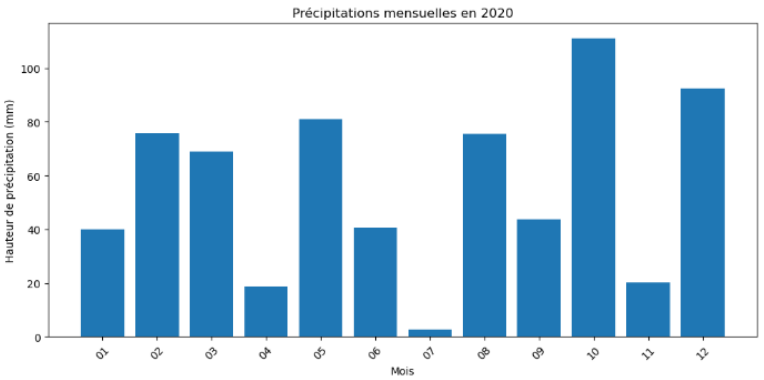


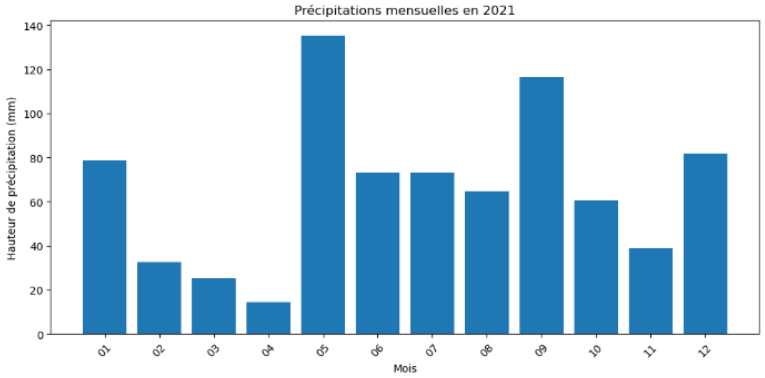
Les enregistrements débutent en novembre 2019, avec une forte augmentation des chutes en novembre et décembre, suggérant des conditions favorables ou une surveillance accrue en fin d'année. En 2020, les chutes de blocs sont fréquentes au premier trimestre, culminant en février avec une diminution progressive à partir de mai. L'année 2021 présente des

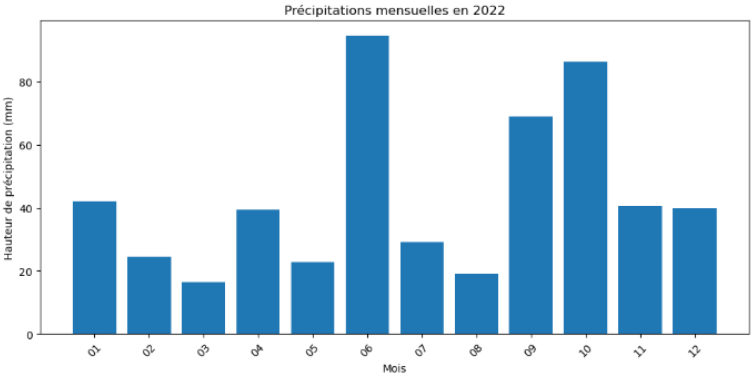
chutes réparties plus uniformément avec des pics notables en juillet et en octobre, indiquant peut-être des variations saisonnières ou des événements spécifiques influençant ces occurrences. En 2022, les chutes sont concentrées principalement de mars à juin, avec un pic en mai, mais il est remarquable qu'aucune chute n'a été enregistrée après juin.

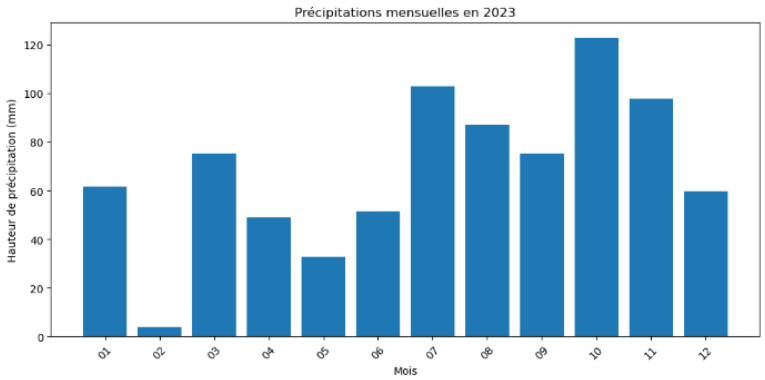
2. Visualisation des pluviométries :

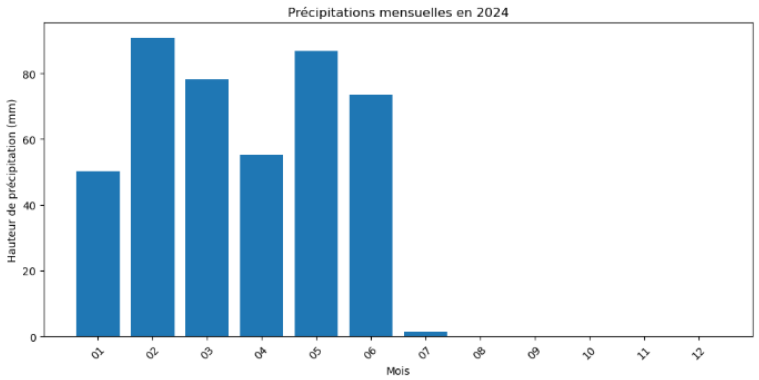






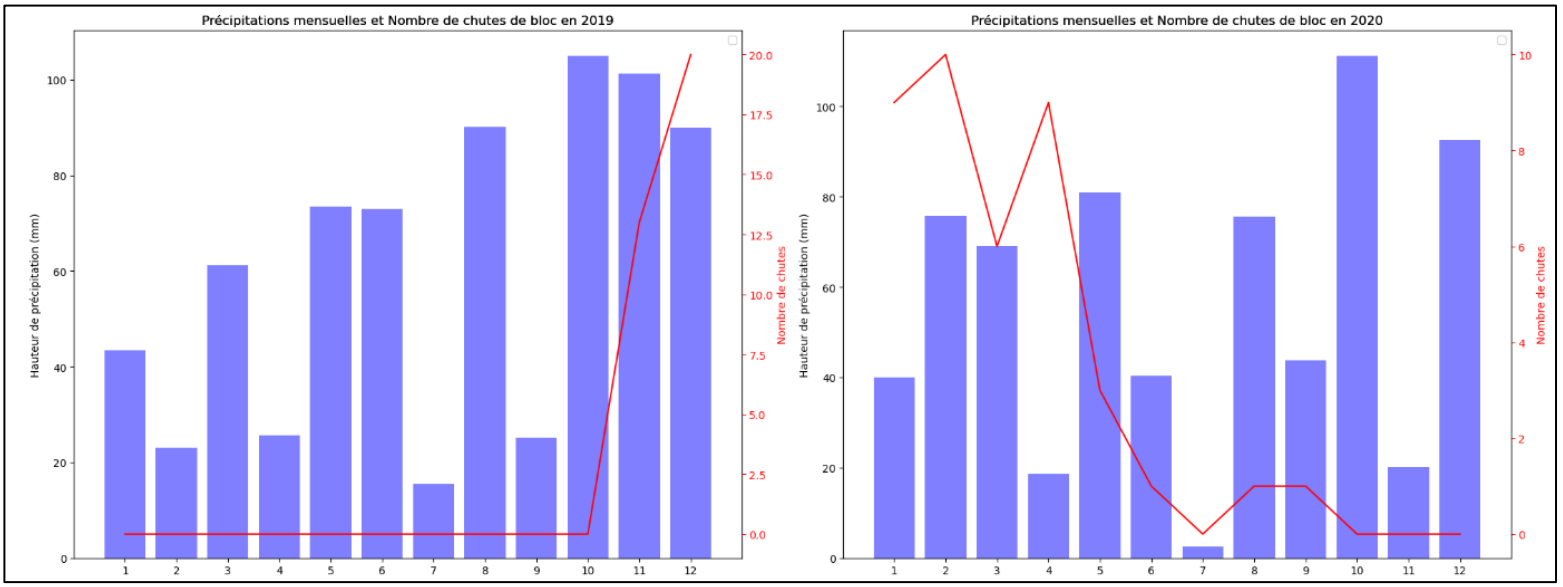


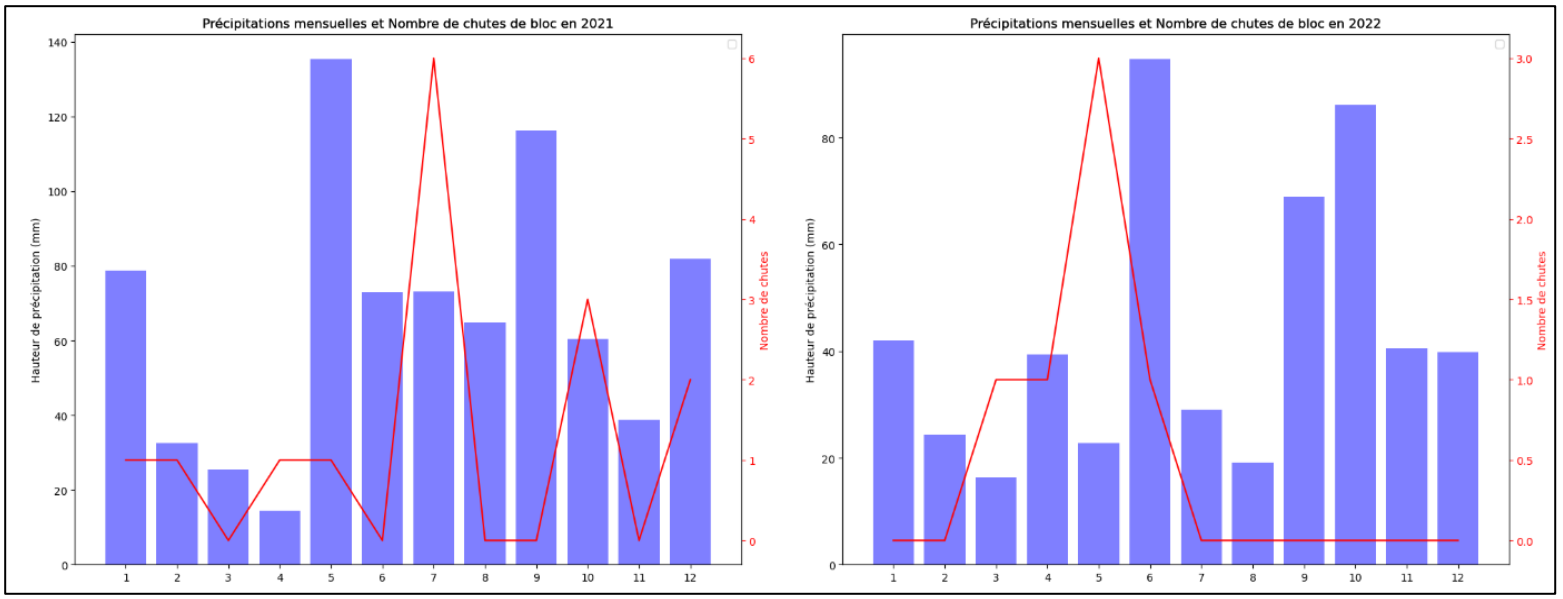




En 2019, les précipitations augmentent progressivement au fil de l'année avec des pics notables en juillet, août, octobre et novembre, indiquant des conditions climatiques potentiellement favorables à des précipitations plus abondantes durant ces mois. L'année 2020 montre une variabilité significative, avec des précipitations élevées en janvier et février, une diminution en été, et un pic marqué en octobre. En 2021, les précipitations sont relativement bien réparties, avec un pic particulièrement élevé en avril, ce qui pourrait être attribué à des événements météorologiques spécifiques au printemps. Enfin, 2022 présente des précipitations irrégulières, avec des pics en mai et octobre, et une diminution notable durant les mois d'été.

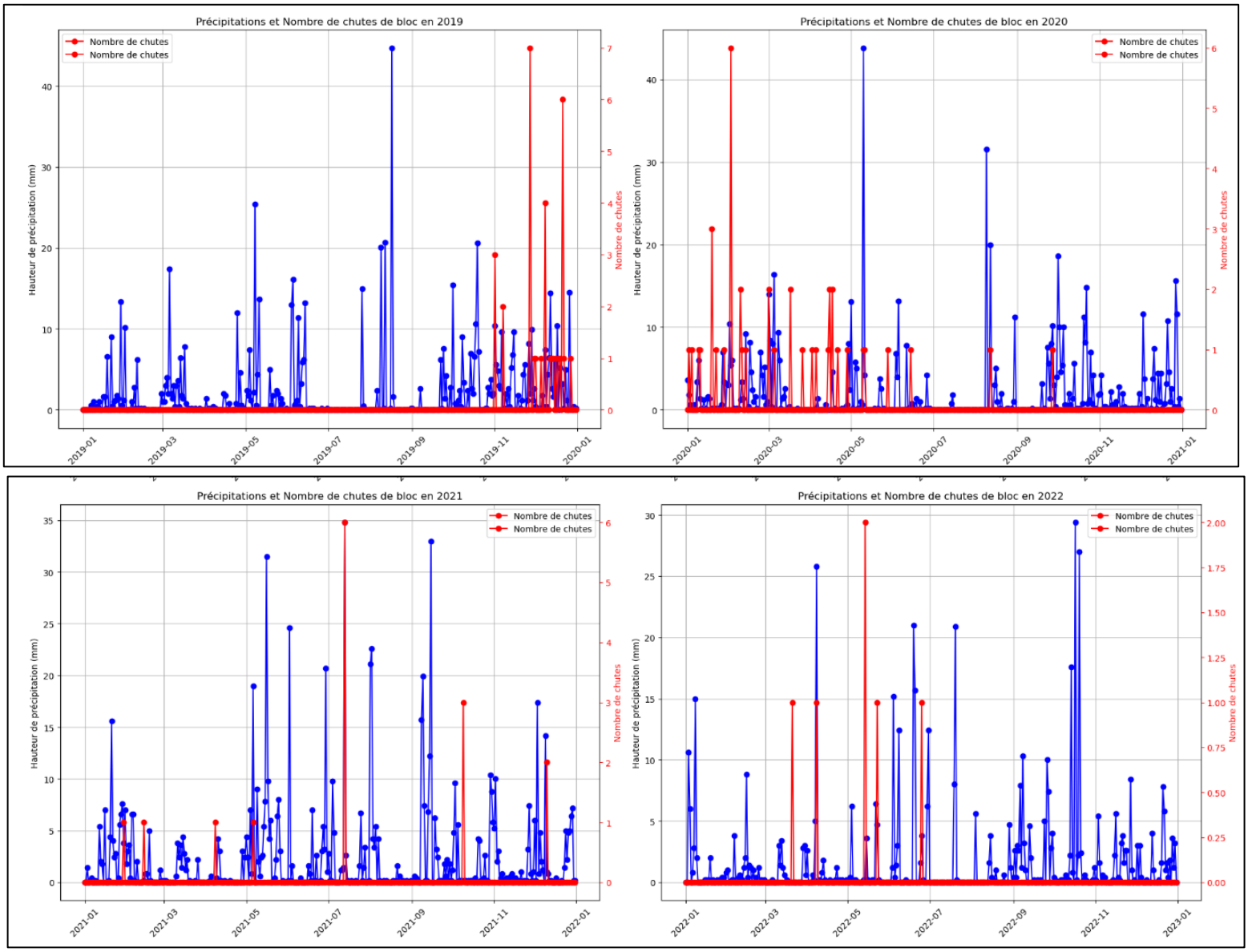
3. Compositions des deux visualisations pour chaque mois :





Les visualisations des précipitations mensuelles (en bleu) et des chutes de blocs (en rouge) dans la carrière de Royer pour les années 2019 à 2022 montrent des tendances intéressantes. En 2019, il y a une nette augmentation des chutes de blocs en novembre et décembre, coïncidant avec une hausse significative des précipitations durant ces mois. Cela suggère une corrélation positive entre les précipitations et les chutes de blocs pour cette période. En 2020, bien que les précipitations soient élevées en janvier et octobre, les chutes de blocs sont principalement concentrées en janvier, février et mars, indiquant une réaction rapide des blocs aux précipitations. En 2021, une forte augmentation des chutes de blocs en juin est visible malgré une précipitation modérée, tandis que les mois de mai et octobre montrent une précipitation élevée avec une correspondance moins claire des chutes. Enfin, en 2022, les chutes de blocs sont concentrées entre mars et juin, avec un pic en avril, corrélant bien avec les précipitations élevées de mars à juin. Ces analyses suggèrent que, bien que les précipitations influencent généralement les chutes de blocs, d'autres facteurs peuvent également jouer un rôle, comme la structure géologique et la saturation préalable du sol.

4. Compositions des deux visualisations par jour :



Les visualisations des précipitations journalières (en bleu) et des chutes de blocs (en rouge) pour les années 2019 à 2022 montrent des tendances intéressantes et permettent d'analyser la corrélation entre les précipitations et les chutes de blocs dans la carrière de Royer.

Année 2019

En 2019, une augmentation notable des chutes de blocs est observée en novembre et décembre, coïncidant avec une hausse significative des précipitations durant ces mois. Cette corrélation positive suggère que les précipitations jouent un rôle important dans le déclenchement des chutes de blocs. En effet, l'augmentation de la hauteur de précipitation semble précéder ou coïncider avec les événements de chute, indiquant que les sols saturés ou les infiltrations d'eau peuvent déstabiliser les blocs rocheux.

Année 2020

En 2020, les précipitations sont élevées en janvier et octobre. Toutefois, les chutes de blocs sont principalement concentrées en janvier, février et mars. Cette observation montre une réaction rapide des blocs aux précipitations, surtout en début d'année. L'augmentation des chutes de blocs en janvier, suite à de fortes précipitations, suggère une saturation rapide des sols. Cependant, l'absence de nombreuses chutes de blocs en octobre, malgré les précipitations élevées, pourrait indiquer que d'autres facteurs comme la température, le gel et le dégel ou la saturation préalable du sol peuvent également influencer la stabilité des blocs.

Année 2021

En 2021, une forte augmentation des chutes de blocs est visible en juin, malgré une précipitation modérée ce mois-là. Cela peut suggérer que la stabilité des blocs a été affaiblie par des précipitations antérieures ou par d'autres facteurs environnementaux. De plus, les mois de mai et octobre montrent des précipitations élevées, mais avec une correspondance moins claire des chutes de blocs, indiquant que les précipitations ne sont pas le seul facteur déterminant les chutes. Il se peut que la combinaison de précipitations antérieures, la saturation des sols et d'autres conditions géologiques soient nécessaires pour déclencher les chutes.

Année 2022

En 2022, les chutes de blocs sont concentrées entre mars et juin, avec un pic en avril qui corrèle bien avec les précipitations élevées de mars à juin. Cette tendance montre une fois de plus la relation entre les précipitations et les chutes de blocs. Cependant, des épisodes de précipitations en dehors de cette période (par exemple en septembre) n'ont pas entraîné autant de chutes, suggérant qu'après une période de haute activité, les sols et les structures rocheuses peuvent atteindre un point de saturation au-delà duquel les précipitations supplémentaires n'ont pas d'effet immédiat.

Conclusion

Ces analyses montrent que, bien que les précipitations influencent généralement les chutes de blocs, d'autres facteurs doivent également être pris en compte, comme la structure géologique, la saturation préalable du sol, les variations saisonnières de température, et d'autres conditions environnementales. La corrélation entre les précipitations et les chutes de blocs est complexe et multidimensionnelle, nécessitant une approche intégrée pour mieux comprendre et prévoir ces événements.

Conclusion finale :

Ce stage a été très formateur sur le plan des connaissances. Il m'a permis de développer des compétences techniques essentielles pour l'analyse de données et de m'épanouir dans le domaine de l'intelligence artificielle. La nécessité de cette classification automatique des signaux acoustiques sur le site de Château Landon nous a incités à rechercher des caractéristiques innovantes et utiles pour distinguer les différentes classes de signaux acoustiques.

Ce stage m'a également offert une excellente opportunité de m'entraîner à la gestion et au traitement des données : construction de bases de données pertinentes, analyse des colinéarités entre les composantes, et augmentation de données. J'ai bénéficié d'une grande liberté pour tester divers modèles de classification, que ce soient des modèles de machine Learning sur des bases de données Excel avec certaines caractéristiques (comme RandomForestClassifier, SVC, LogisticRegression, XGBoost) ou des modèles de deep learning sur des bases de données basées sur des spectrogrammes ou des vecteurs temporels (comme le CNN spectrogramme, le CNN temporel, et le RNN temporel).

Ce stage m'a permis de comparer les performances de ces modèles et d'adopter des stratégies pour les améliorer, comme le grid search et l'obtention de bases de signaux pertinentes. Il m'a également aidé à développer des compétences linguistiques en français, notamment lors des réunions régulières avec mes tuteurs et lors de ma présentation au cours du premier mois de stage.

En outre, ce stage m'a aidé à développer des soft skills telles que la gestion du temps, la patience, surtout dans les moments difficiles (comme lorsque j'ai été gêné par une allergie et un manque de motivation), l'écoute des autres et la défense de mes idées. J'ai également eu l'occasion de rencontrer un grand nombre de chercheurs et d'ingénieurs avec qui j'ai pu échanger des expériences.

En résumé, ce stage a été extrêmement bénéfique pour mon développement personnel et professionnel. Je remercie mes tuteurs de stage de m’avoir accordé cette expérience constructive à l’INERIS.