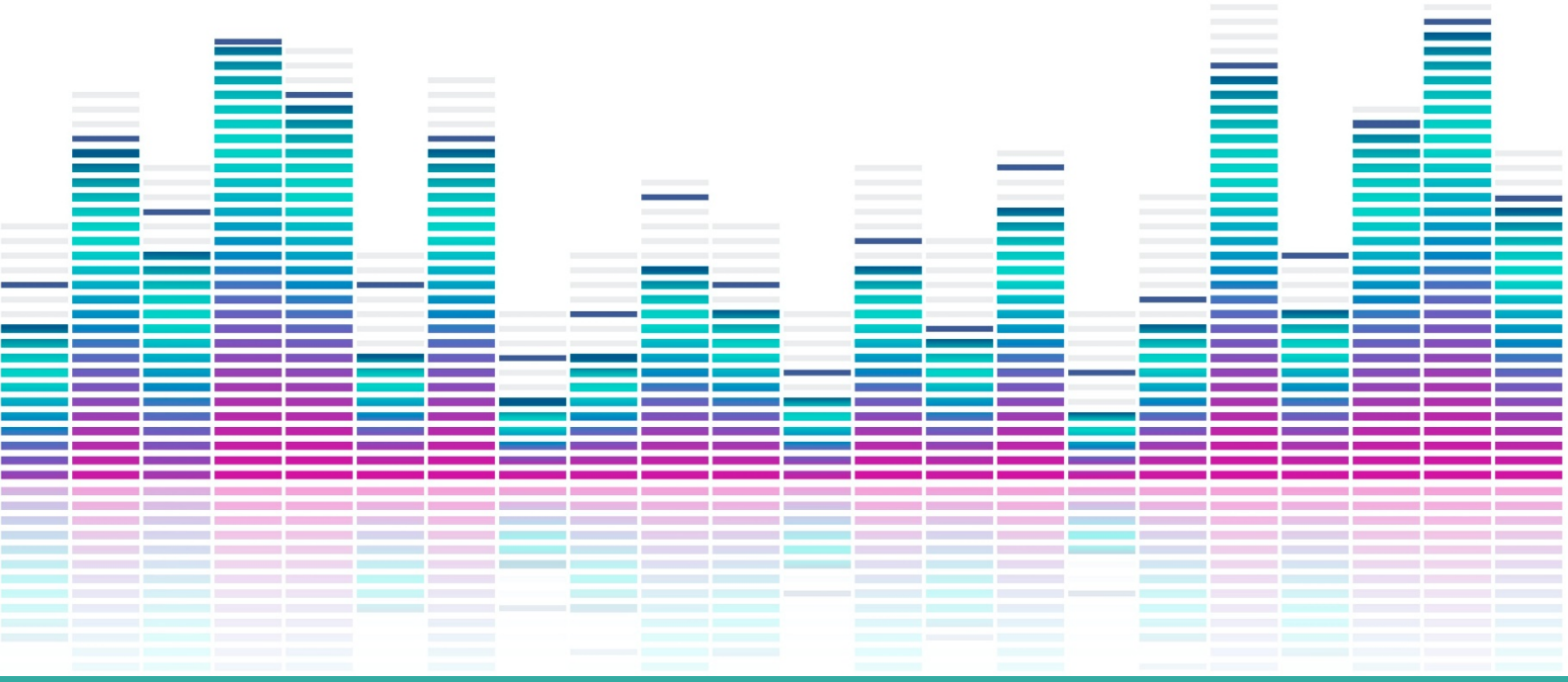
Anomalous Sound Detection



PROMOTION DATASCIENTIST MAI 2021

Astrid LANCREY
Corentin DESMETTRE
Yossef ABOUKRAT
Youcef MADJIDI

Table des matières

Contexte

La robotisation et l'automatisation des machines et des biens est un enjeu du 21-ème siècle. A cette fin, de plus en plus d'industries investissent pour rendre totalement autonomes des parties de leur process ou des tâches qui nécessitent habituellement la présence constante d'un opérateur. Les exemples les plus représentatifs dans l'industrie sont les robots de manutention et les chaînes d'assemblage automatiques. Concernant le quotidien, la course à la voiture autonome et aux drones de surveillance / livraison en sont des exemples les plus visibles.

La problématique associée à ces enjeux est l'auto-maintenance des machines ; tâche de surveillance réalisée dans le passé par l'opérateur, son absence induit un contrôle moins fréquent des paramètres et donc un risque de panne plus élevé. Dans cet objectif, l'enjeu du projet est de pouvoir détecter une anomalie dans une machine grâce à sa signature audio.

Problématique et objectif

Le principal défi de cette tâche est de détecter des sons anormaux qui sont par définition inconnus. En effet, dans les usines, les sons anormaux réels sont très diversifiés et nous ne pouvons avoir une liste exhaustive de ceux-ci. Cela signifie que nous devons détecter des sons anormaux inconnus qui n'ont pas été observés dans les données d'entraînement fournies. Aussi, ce problème ne peut être vu comme un simple problème de classification a deux classes (normal/anormal).

Nous tenterons donc au travers de différentes approche et model de détecter une anomalie dans une machine grâce à sa signature audio. Le livrable principal de ce projet se constituera d'un algorithme de contrôle et d'alerte implémentable dans l'appareil industriel et pourquoi pas par la suite l'intégrer dans un dispositif (type Raspberry Pi équipé d'un micro) configuré pour une surveillance en temps réel de la machine.

Data

Le jeu de donnée utilisé est celui disponible sur le site KAGGLE dans le cadre du challenge DCASE2020-Task2 qui représente 9,42 Go de fichiers audio au format WAV. On y retrouve des parties de différentes DataBases tel que ToyADMOS et de l'ensemble de données MIMII constituées des sons de fonctionnement normaux/anormaux de six types de machines jouets/réelles. Chaque enregistrement est un audio à canal unique (mono), d'une longueur d'environ 10 secondes qui comprend à la fois le son de fonctionnement de la machine cible et le bruit ambiant. Les six types de machines jouet/réelles suivants sont utilisés dans cette tâche :

Machine	Fichiers audio Total	Fichiers Anomalie	Fichiers Normaux
ToyConveyor	**6509**	**1059**	**5400**
ToyCar	**6459**	**1110**	**5399**
fan	**5550**	**1475**	**4075**
pump	**4205**	**456**	**3749**
valve	**4170**	**890**	**3204**
slider	**4094**	**479**	**3691**
Total	**30987**	**5469**	**25518**

Benchmark de l'existant, La stratégie de mesure et de contrôle des objectifs

1- Modèles utilisés en ASD (état de l'art)

La revue arXiv:2102.07820 qui recense une trentaine de publications supposées être les plus pertinentes dans le domaine du Machine Learning appliqué à l'ASD (Anomaly Sound Detection) a permis de mettre en avant les modèles les plus fréquemment utilisés pour répondre à ce type de problématique: il s'agit principalement de réseaux de neurones de type CNN (réseaux de neurones convolutionnels) et AE (AutoEncoder). Les jeux de données utilisés dans ces études sont en majorité les datasets ToyADMOS et MIMII qui sont inclus dans notre dataset DCASE2020-Task2. Les études sur ToyADMOS et MIMII ont toutes utilisé la courbe ROC et la métrique associée AUC pour évaluer leur modèle.

2- Modèles utilisés dans le cadre du challenge DCASE2020-Task2

D'après la revue consacrée spécifiquement aux études ayant développé les modèles les plus performants sur notre jeu de données DCASE2020-Task2 (Koyzumi & al, 2020) certaines se sont démarquées pour avoir utilisé une approche de classification tandis que les autres se sont focalisées sur la détection d'outliers par rapport à une "baseline".

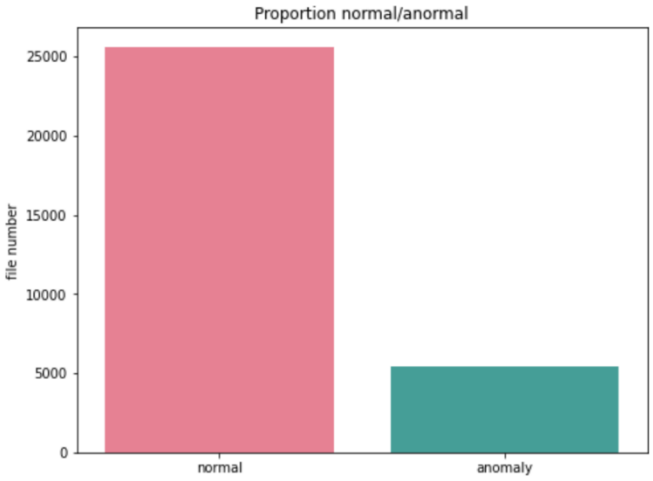
Le système de baseline repose sur un AutoEncoder qui permet de calculer un score d'anomalie. L'AE est un réseau de neurones de type FCNN (Fully Connected Neural Network) qui prend en input un spectrogramme log-mel. Il est entraîné sur des données "normales" et son optimisation consiste à minimiser au maximum l'erreur associée à la reconstruction du signal normal. Finalement le score d'anomalie est calculé à partir de cette erreur. Autrement dit l'AE est optimisé pour minimiser le score d'anomalie associé aux fichiers de label "normal" sur lesquels il est entraîné, dans le but de pouvoir détecter l'anormalité sur des données test de label "anormal".

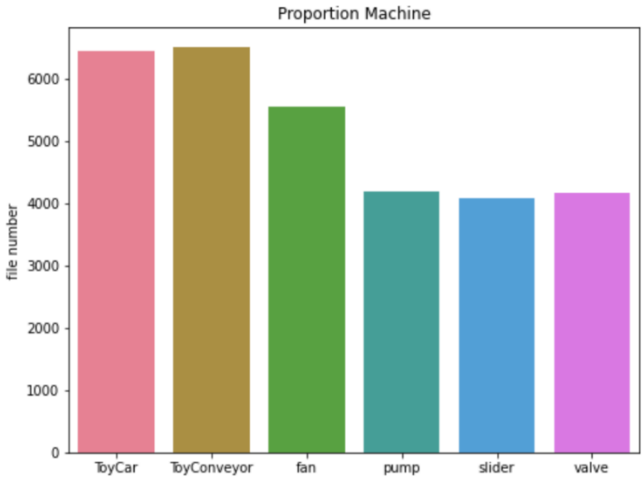
Dans le cas où les participants du Challenge ont adopté une méthode de classification via un DNN (Deep neural network), ils ont utilisé, pour une machine donnée, les fichiers de label "normal" d'autres machines comme des labels anormaux.

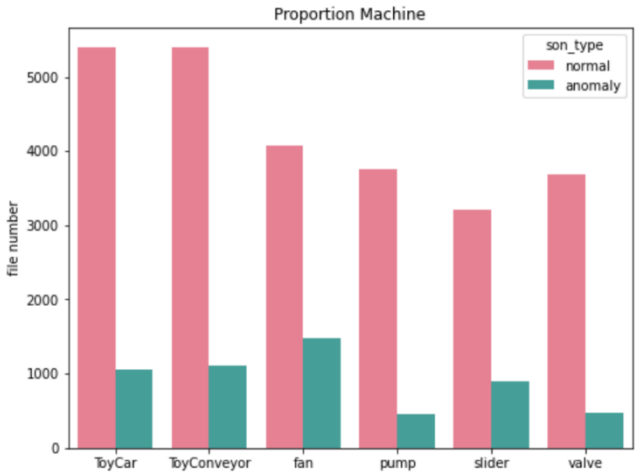
Enfin une dernière approche innovante, basée sur un AutoEncoder, a consisté à prévenir un potentiel effet indésirable de l'AE : à savoir associer des scores d'anormalité élevés à un ID de machine différent de celui avec lequel il a été entraîné et qui est pourtant de label "normal". Pour pallier ce problème deux équipes ont eu recours à deux stratégies différentes : la première a entraîné son AE avec un label "anormal" pour entraîner l'AE à ne pas reconstruire de signal "normal" dans ce cas (Daniluk & al., 2020). L'autre stratégie a consisté à inclure l'ID de la machine aux features à extraire ainsi le modèle ne reconstruit pas seulement les features du spectre audio mais aussi l'ID machine (Hayashi & al., 2020).

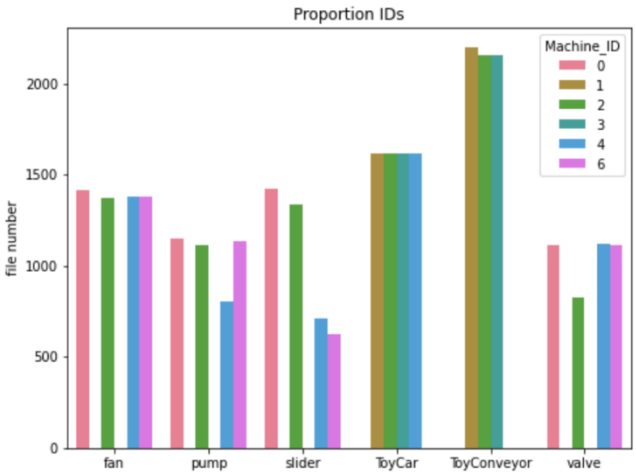
DataViz

Voici les quelques diagrammes de répartition des données de notre dataset (notebook Dataviz.ipynb).





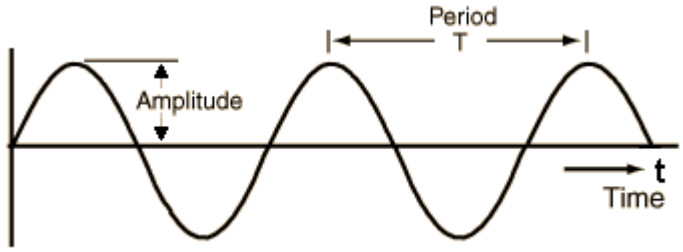


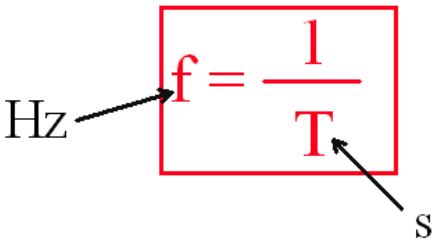


Projet

Avant-propos

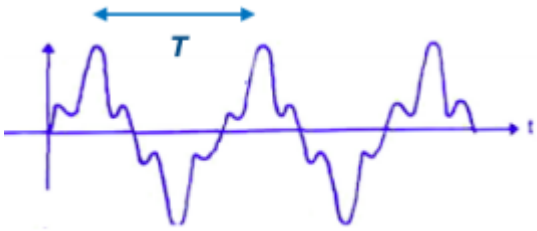
Un signal sonore est produit par des variations de la pression de l'air. Nous pouvons mesurer l'intensité des variations de pression et tracer ces mesures dans le temps. Ce signal est caractérisé par son amplitude, sa période et sa fréquence **:**



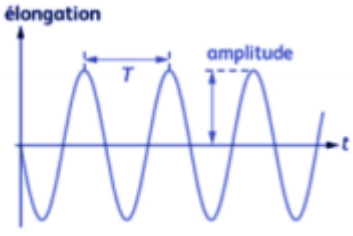


- Les signaux sonores se répètent souvent à intervalles réguliers, de sorte que chaque onde a la même forme. La hauteur montre l'intensité du son et est connue sous le nom d'amplitude.
- Le temps pris par le signal pour compléter une onde complète est la période. Le nombre d'ondes produites par le signal en une seconde s'appelle la fréquence. La fréquence est l'inverse de la période. L'unité de fréquence est le Hertz.

La majorité des sons que nous rencontrons ne suivent pas nécessairement des schémas périodiques aussi simples et réguliers. En effet, les signaux de fréquences différentes peuvent être additionnés pour créer des signaux composites avec des schémas répétitifs plus complexes. Tous les sons que nous entendons, y compris notre propre voix humaine, se composent de formes d'onde complexe comme celles présentées ci-dessous, qui pourraient par exemple correspondre au son d'un instrument de musique.

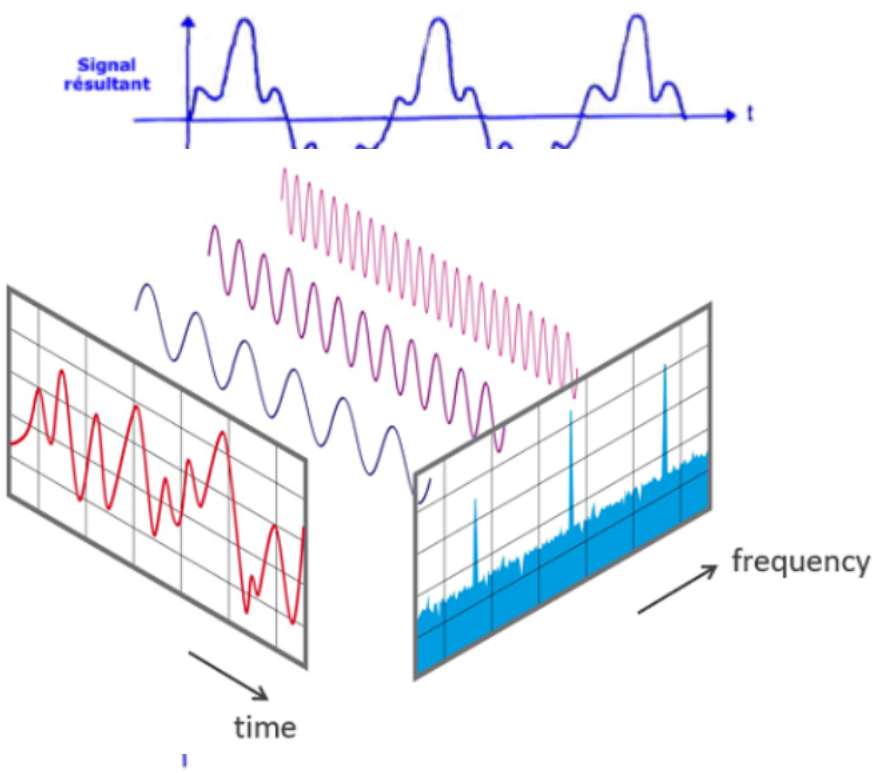


Son complexe



Son pur

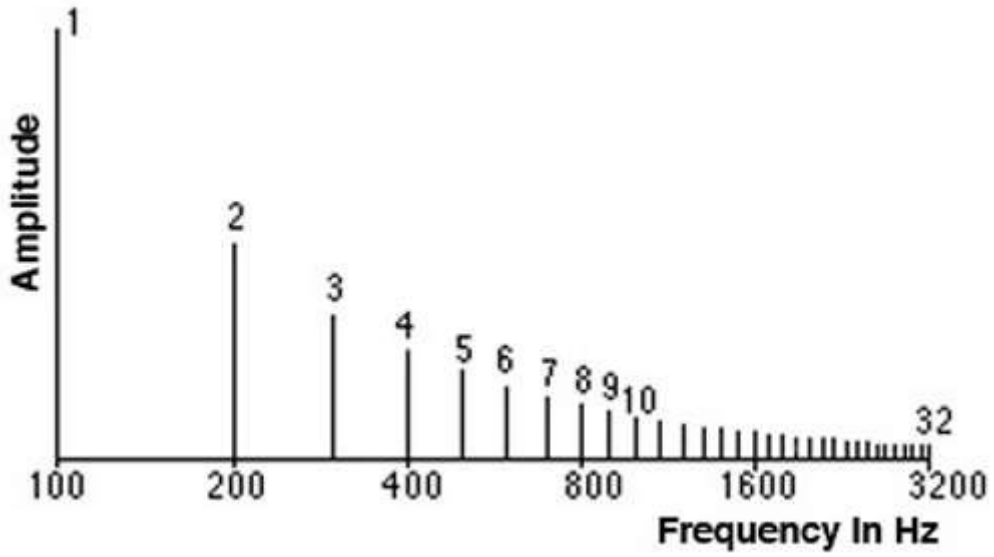
Il a été démontré pas Joseph Fourier que tout signal temporel périodique, comme un signal sonore, est en fait l'addition de signaux (ici nommé pure dans notre exemple) de différentes fréquences.



Ainsi, grâce à cette décomposition, il est possible de passer du domaine temporel où l'analyse mathématique du signal est très compliquée, au domaine fréquentiel où on s'intéresse qu'aux fréquences et où il est possible d'effectuer des analyses plus poussées ;

L'analyse spectrale ou analyse fréquentielle d'un son complexe consiste à représenter graphiquement les fréquences des différents composants sinusoïdaux du son. En abscisse on représente les fréquences et en ordonnée les amplitudes des composants du son.

Par exemple l'analyse spectrale du son complexe précédent donne le résultat ci-dessous :



Deep learning et analyse audio

Ces dernières années, le Deep Learning est devenu omniprésent et a connu un énorme succès dans le traitement de données audio. Avec le deep learning, les techniques traditionnelles de traitement de l'audio ne sont plus nécessaires, et nous pouvons nous appuyer sur une préparation standard des données sans nécessiter beaucoup de génération manuelle et personnalisée de caractéristiques.

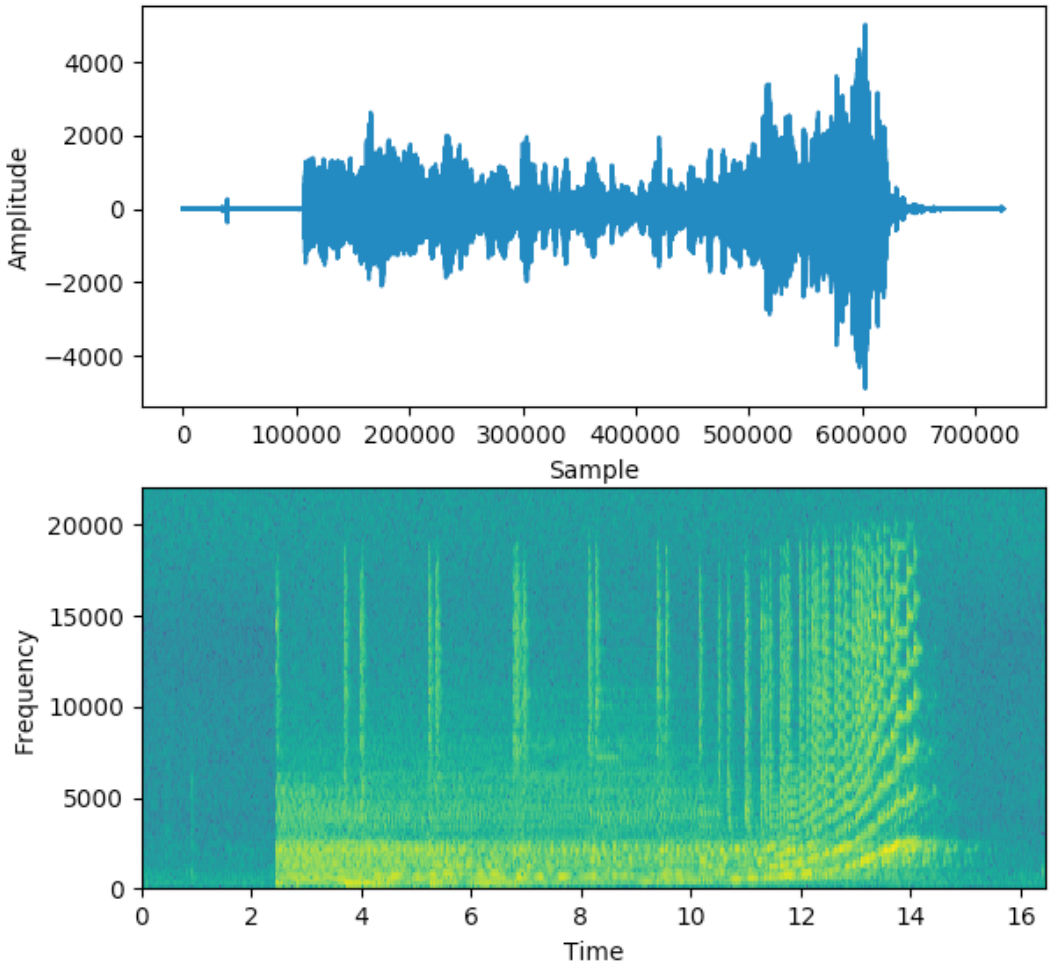
En effet, nous ne traitons pas réellement les données audios dans leur forme brute. Au lieu de cela, l'approche commune utilisée consiste à convertir les données audios en « image » et de les traiter avec les outils du Deep Learning. Cela se fait en générant des Spectrogrammes à partir de l'audio.

Le spectrogramme

Comme un signal produit des sons différents lorsqu'il varie dans le temps, les fréquences qui le composent varient également avec le temps. En d'autres termes, son spectre varie avec le temps.

Ainsi, Le spectrogramme d'un signal trace son spectre dans le temps et constitue une sorte de "photographie" du signal. Il représente le temps sur l'axe des x et les fréquence sur l'axe des y. C'est comme si nous prenions le spectre encore et encore à différents moments dans le temps, puis que nous les réunissions tous en un seul graphique. Il utilise différentes couleurs pour indiquer l'amplitude ou la force de chaque fréquence. Plus la couleur est vive, plus l'énergie du signal est élevée. Chaque "tranche" verticale du spectrogramme est essentiellement le spectre du signal à cet instant et montre comment l'énergie du signal est distribuée dans chaque fréquence présente dans le signal à cet instant.

Dans l'exemple ci-dessous, la première image montre le signal dans le domaine temporel, c'est-à-dire l'amplitude en fonction du temps. Cela nous donne une idée du niveau sonore ou silencieux d'un clip à un moment donné, mais cela nous donne très peu d'informations sur les fréquences présentes tandis que le spectrogramme (2nde image) affiche le signal dans le domaine de la fréquence.



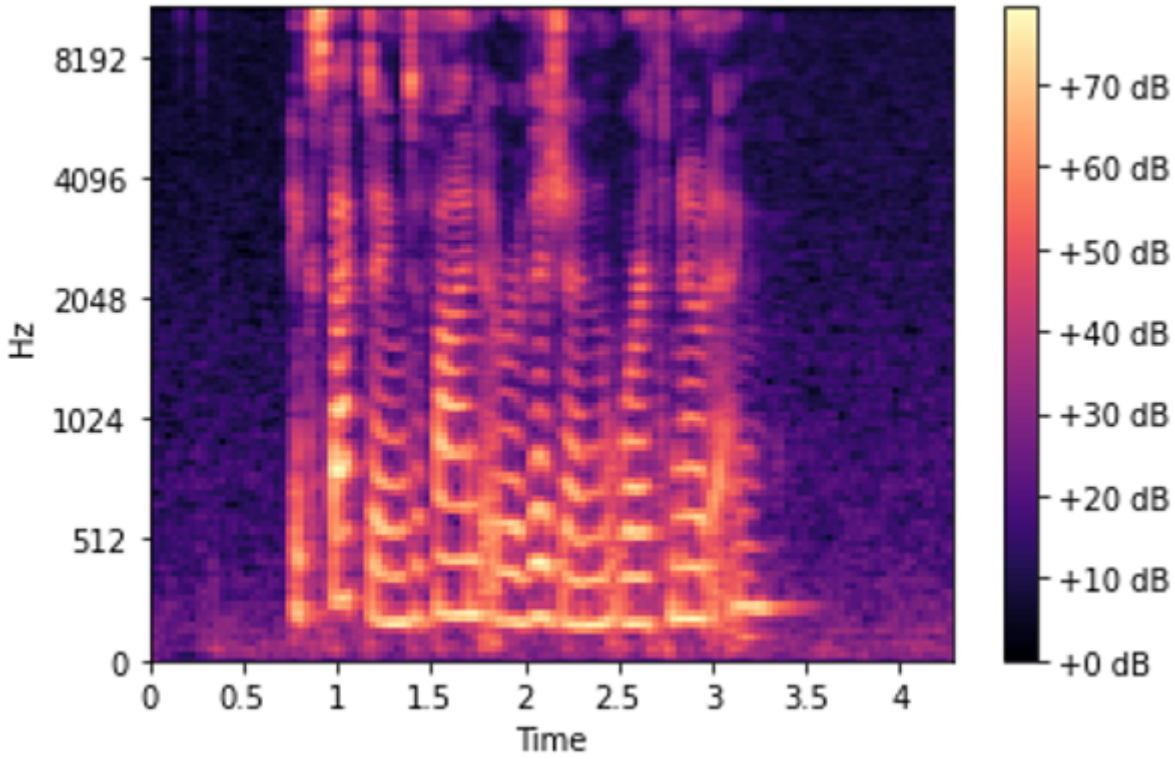
Utilisation du Mel Spectrogramme

Un spectrogramme Mel apporte deux changements importants par rapport à un spectrogramme normal qui trace la fréquence en fonction du temps.

- Il utilise l'échelle de Mel au lieu de la fréquence sur l'axe des ordonnées.
- Il utilise l'échelle des décibels au lieu de l'amplitude pour indiquer les couleurs.

En effet, les échelles Mel et Décibel ont été mises au point pour tenir compte de la sensibilité logarithmique de l'oreille humaine. Ainsi, ces échelles sont plus fidèles à la façon dont l'oreille perçoit le son, c'est pourquoi elles sont souvent utilisées dans l'analyse de son en Deep Learning. La librairie

Librosa de Python fournit tous les outils permettant une utilisation rapide et facile pour l'analyse spectrale. C'est pourquoi nous avons privilégié son usage pour notre projet.



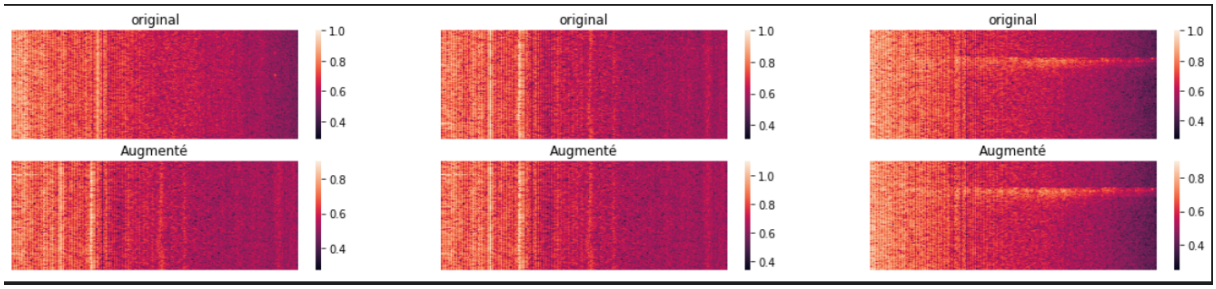
Classification du problème

Nous avons ici à faire à une problématique de détection d'anomalies qui diffère d'un problème de classification "classique" de par la sous-représentation d'une des deux classes de données à identifier : à savoir les fichiers son correspondant à des sons anormaux émis par une machine.

Pré-processing, Choix du modèle & Optimisation

Nous avions commencé par utiliser des transformations de Fourrier en testant différents nombres de dt puis nous avons finalement choisi d'augmenter le découpage du signal et passer en transformations LogMel afin de d'avantage mîmer le traitement du signal par l'oreille humaine. Tous les modèles présentés plus bas ont systématiquement été appliqués sur des spectrogrammes LogMel de 256 dt par 256 fréquences à la fois pour avoir une valeur de dt comprise entre les valeurs communément utilisées de 20 et 50 ms, ainsi que pour avoir une puissance de deux pour la valeur de dt et la valeur de fréquences , dans l'optique de ne pas changer la shape en passant de la partie Encoder à la partie Decoder d'un AuotEncoder (avons pour contrainte d'avoir une shape en sortie identique à la shape en entrée). Pour pouvoir obtenir cette combinaison nous avons utilisé une fonction maison appelée « downsizing » qui permet à la fois de normaliser les valeurs de décibels (qui correspondent aux valeurs de pixels de l'image spectrogramme) entre zéro et 1, ainsi que de sélectionner soit un seuil de fréquences soit un seuil de décibels (dans notre cas nous avons donc choisi 256 fréquences) rentré en argument par l'utilisateur.

Dans le but de limiter le sur-apprentissage des différents modèles testés nous avons systématiquement utilisé une fonction générateur « maison » qui permet de faire de l'augmentation de données à partir des données « train » : les nouvelles données sont générées par augmentation légère des données initiales. Les spectrogrammes représentés en exemple ci-dessous sont issus d'une homothétie de l'amplitude des LogMel comprise aléatoirement entre 0.1 et 10%. On peut constater sur les premiers et dernier spectre augmentés que, après augmentation, les valeurs ne s'étalent non plus sur une plage de 0.2 à 1 mais sur une plage de 0.2 à 0.8.



Pour générer un modèle qui permettrait de de détecter les anomalies nous avons **exploré** quatre approches distinctes :

1. Une approche d'AutoEncoder simple
2. Un classifieur hybride (Encoder-couches dense) du type de son normal vs anormal
3. Un classifieur hybride (Encoder-couches dense) des ID
4. Un classifieur RNN des ID

1- Autoencoder

En nous inspirant de la "baseline" qui est le modèle proposé par les coordinateurs du challenge DCASE2020, nous avons tout d'abord mis au point un Autoencoder que nous avons entraîné à reconstituer les spectrogrammes logMel issus des fichiers son de type normal. Le but étant d'obtenir des reconstitutions assez fidèles pour que la MSE calculée à partir des pixels du spectrogramme initial et après reconstitution par notre AE augmente lorsque les reconstitutions sont prédites sur des fichiers anormaux. Ainsi nous pourrions utiliser un seuil de valeur de MSE comme score d'anomalie qui si il est atteint conduit à la classification du fichier dans la classe « anomaly ».

Nous avons testé en parallèle un AE constitué de couches de convolution 2D et un AE similaire à celui utilisé dans la baseline constitué de couches de type dense. Nous présenterons dans une première sous-partie les distributions des MSE obtenues sur l'ensemble des fichiers de type normal et anormal au fil des optimisations testées sur un AE de convolution. Dans un second temps nous présenterons les résultats obtenus sur un AE dense.

a- Autoencoder de convolution

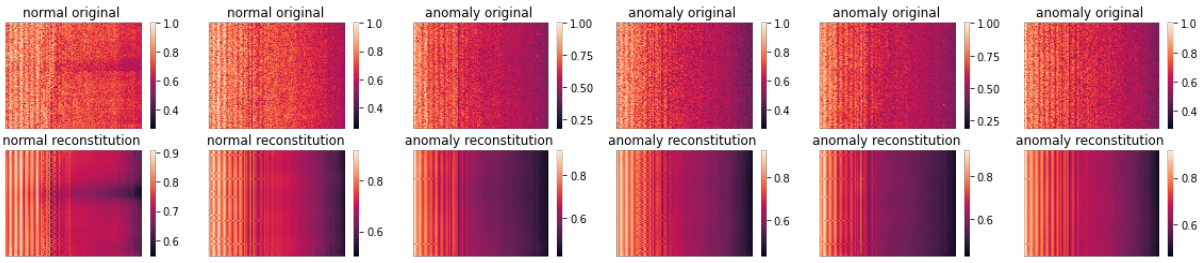
L'architecture de l'Autoencoder qui nous a permis d'obtenir les reconstructions des spectrogrammes LogMel les plus proches des originaux est la suivante (à gauche partie Encoder, à droite partie Decoder) :

conv2d_16 (Conv2D)	(None, 8, 8, 4)	148
up_sampling2d_5 (UpSampling2	(None, 16, 16, 4)	0
conv2d_17 (Conv2D)	(None, 16, 16, 8)	296
up_sampling2d_6 (UpSampling2	(None, 32, 32, 8)	0
conv2d_18 (Conv2D)	(None, 32, 32, 16)	1168
up_sampling2d_7 (UpSampling2	(None, 64, 64, 16)	0
conv2d_19 (Conv2D)	(None, 64, 64, 32)	4640
up_sampling2d_8 (UpSampling2	(None, 128, 128, 32)	0
conv2d_20 (Conv2D)	(None, 128, 128, 64)	18496
up_sampling2d_9 (UpSampling2	(None, 256, 256, 64)	0
conv2d_21 (Conv2D)	(None, 256, 256, 1)	577

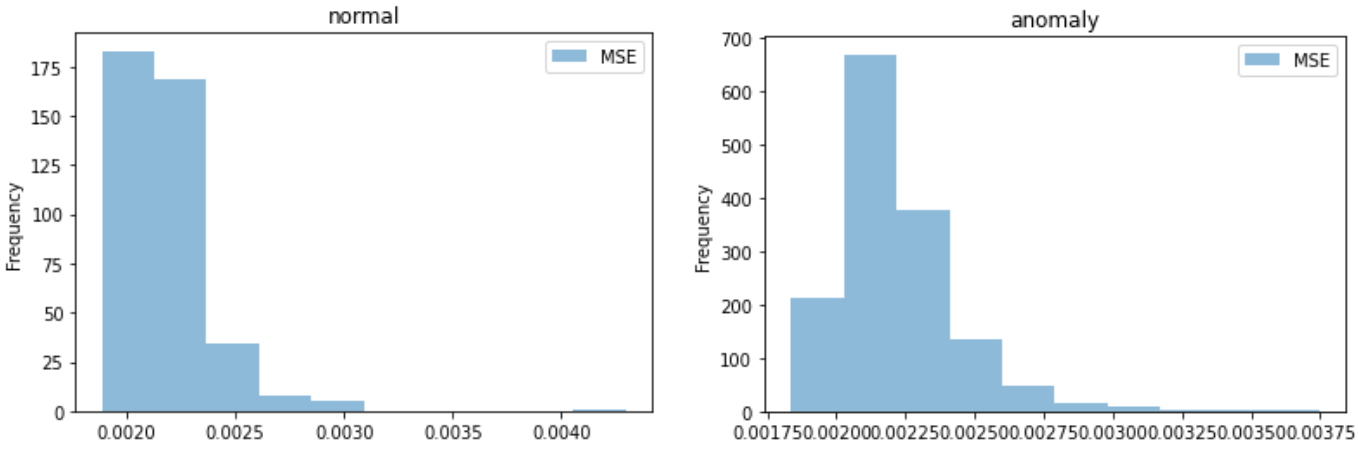
Total params: 50,505
Trainable params: 50,505
Non-trainable params: 0

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 256, 256, 1)]	0
conv2d_11 (Conv2D)	(None, 256, 256, 64)	640
max_pooling2d_5 (MaxPooling2	(None, 128, 128, 64)	0
conv2d_12 (Conv2D)	(None, 128, 128, 32)	18464
max_pooling2d_6 (MaxPooling2	(None, 64, 64, 32)	0
conv2d_13 (Conv2D)	(None, 64, 64, 16)	4624
max_pooling2d_7 (MaxPooling2	(None, 32, 32, 16)	0
conv2d_14 (Conv2D)	(None, 32, 32, 8)	1160
max_pooling2d_8 (MaxPooling2	(None, 16, 16, 8)	0
conv2d_15 (Conv2D)	(None, 16, 16, 4)	292
max_pooling2d_9 (MaxPooling2	(None, 8, 8, 4)	0

Un échantillon de reconstitutions obtenues à partir de cette architecture entraînée sur le jeu de donnés train de la machine fan est présenté ci-dessous pour deux fichiers normaux et quatre anormaux sélectionnés aléatoirement (les fichiers originaux en haut avec leurs reconstitutions respectives en bas):



Nous avons représenté les MSE calculées sur les pixels des spectrogrammes avant et après reconstitution par l'AE pour chaque fichier. Les distributions des valeurs obtenues sont représentées par les histogrammes ci-dessous :



Ces distributions reflètent l'incapacité de notre Autoencoder seul à différencier un fichier normal d'un anormal, les MSE des fichiers anormaux étant distribuées selon un range de valeurs très semblables à celui des fichiers normaux. Nous sommes donc tournés vers des approches hybrides basées sur la partie Encoder de l'AE présenté ci-dessus.

b- AutoEncoder dense

Nous avons, entre autres, testé l'architecture suivante d'un AutoEncoder dense :

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 256, 256, 1)]	0
dense (Dense)	(None, 256, 256, 64)	128
batch_normalization (BatchNo	(None, 256, 256, 64)	256
activation (Activation)	(None, 256, 256, 64)	0
dense_1 (Dense)	(None, 256, 256, 64)	4160
batch_normalization_1 (Batch	(None, 256, 256, 64)	256
activation_1 (Activation)	(None, 256, 256, 64)	0
dense_2 (Dense)	(None, 256, 256, 64)	4160
batch_normalization_2 (Batch	(None, 256, 256, 64)	256
activation_2 (Activation)	(None, 256, 256, 64)	0
dense_3 (Dense)	(None, 256, 256, 8)	520
batch_normalization_3 (Batch	(None, 256, 256, 8)	32
activation_3 (Activation)	(None, 256, 256, 8)	0
dense_4 (Dense)	(None, 256, 256, 64)	576
batch_normalization_4 (Batch	(None, 256, 256, 64)	256
activation_4 (Activation)	(None, 256, 256, 64)	0
dense_5 (Dense)	(None, 256, 256, 64)	4160
batch_normalization_5 (Batch	(None, 256, 256, 64)	256
activation_5 (Activation)	(None, 256, 256, 64)	0
dense_6 (Dense)	(None, 256, 256, 64)	4160
batch_normalization_6 (Batch	(None, 256, 256, 64)	256
activation_6 (Activation)	(None, 256, 256, 64)	0
dense_7 (Dense)	(None, 256, 256, 1)	65

Total params: 19,497
Trainable params: 18,713
Non-trainable params: 784

Malheureusement nous n'avons pu faire tourner le modèle que sur une dizaine d'epochs, celui-ci nécessitant une puissance de calcul trop importante pour nos machines. Nous avons donc été contraints de laisser de côté cette approche.

2- Modèle hybride de classification normal vs anormal

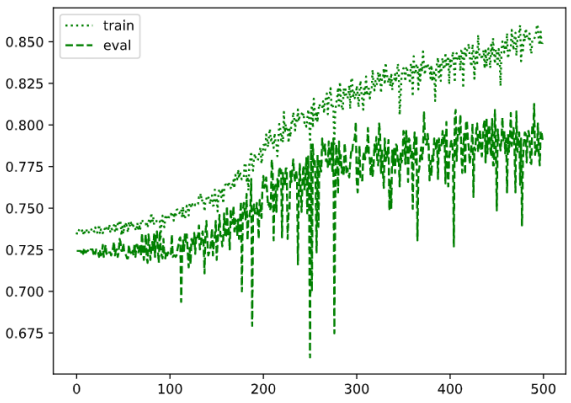
A défaut de pouvoir identifier les anomalies avec notre AutoEncoder seul, nous avons expérimenté un modèle de classification hybride en remplaçant la partie décoder (convolution, upsampling) par des couches de classification dense.

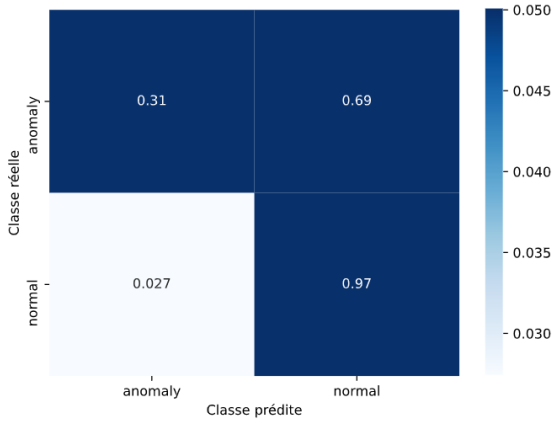
La démarche étant une étude de faisabilité, nous avons entraîné le modèle sur les données du train et du test, en nous servant de générateur de données pour atteindre au moins 30% d'anomalies dans l'échantillon final. D'un point de vue utilisation finale, cette démarche serait équivalente à entraîner un AutoEncoder sur une machine saine lors de la conception, puis d'entraîner le modèle décisionnel sur différentes anomalies de la même machine pour enfin ne ré-entraîner que la partie Auto-encoder sur la machine cliente saine finale.

Nous avons fait varier plusieurs paramètres :

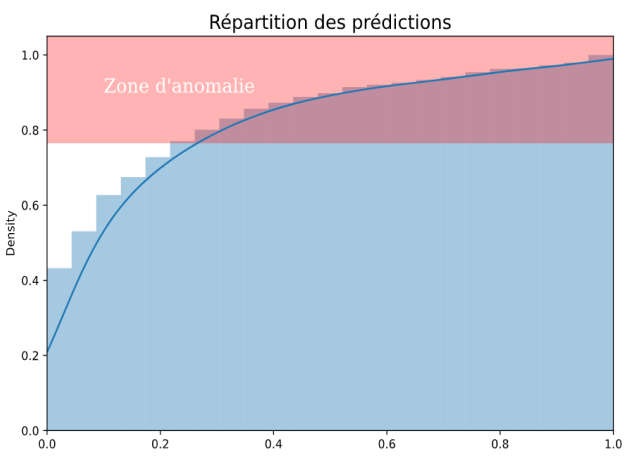
- 1 à 3 couches Denses cachées avant la couche de sortie
- 64 à 1024 Neurones par couche
- Avec et Sans Dropout
- Différentes activation finale et par couche (softmax, sigmoid, tanh, relu,leakyrelu)

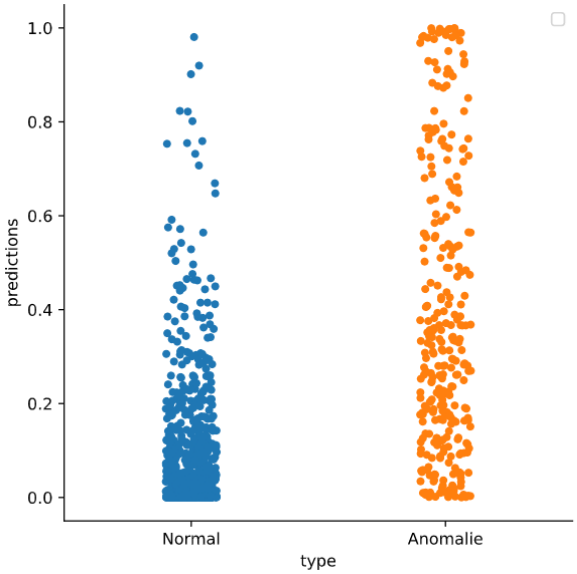
Entrainement :





Résultats :





La figure en bas à droite représente la distribution des valeurs de probabilité prédites par le modèle pour chaque la classe lorsque celle-ci correspond à la classe réelle (normal :bleu et anomalie: orange). En nous basant sur ces distributions, nous avons choisi de régler le seuil de décision à 0.4 au lieu de 0.5 pour la classe anomalie.

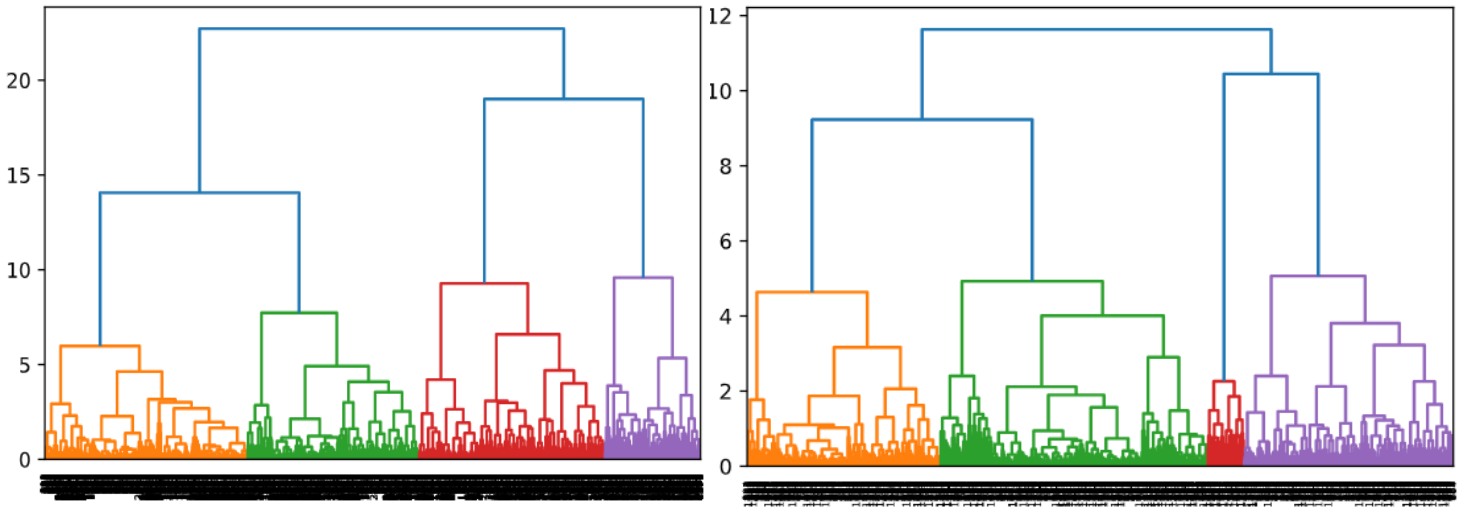
Cela nous a en effet permis de détecter 7% d'anomalies supplémentaires mais avec deux fois plus de fausses détections (5% au lieu de 2.5%).

Conclusion :

Ce modèle valide la capacité de prédiction normal/anormal sur notre jeu de donnée et avec nos outils de traitement.

3- Modèle de classification des ID de type hybride

A défaut de pouvoir identifier les anomalies avec notre AutoeEncoder seul, nous avons expérimenté un modèle de classification d'ID hybride : en raboutant le vecteur de compression de la partie Encoder de notre AE avec des couches denses pour faire de la classification à quatre classes. La finalité étant de faire la classification binaire du type normal vs anormal en nous basant sur les qualités de prédiction des ID : "Le modèle a raison" vs "le modèle s'est trompé". Avant d'utiliser le vecteur de compression de notre Encoder pour l'utiliser pour faire de la classification d'ID, nous avons effectué un clustering hiérarchique sur les features issues de ce dernier sur l'ensemble des prédictions de la machine "fan". Les résultats obtenus à partir des prédictions sur le train (donc sans anomalie) sont présentés à gauche. Ceux obtenus à partir des prédictions sur le jeu de donées test contenant des anomalies sont présentés à droite :



Nous pouvons identifier quatre clusters dont on peut supposer qu'ils correspondent aux quatre ID de la machine fan. Ce résultat confirme la faisabilité de détecter les classes des ID machines sur des données fortement compressées.

Une fois ce contrôle effectué, nous avons ensuite ajouté des couches denses derrière le vecteur de compression de l'Encoder, la dernière comportant 4 neurones et une fonction d'activation softmax dans le but de classifier nos ID à partir des features du vecteur de compression. Nous avons testé une première architecture avec autant de neurones sur les couches dense que de variables dans la couche flatten.

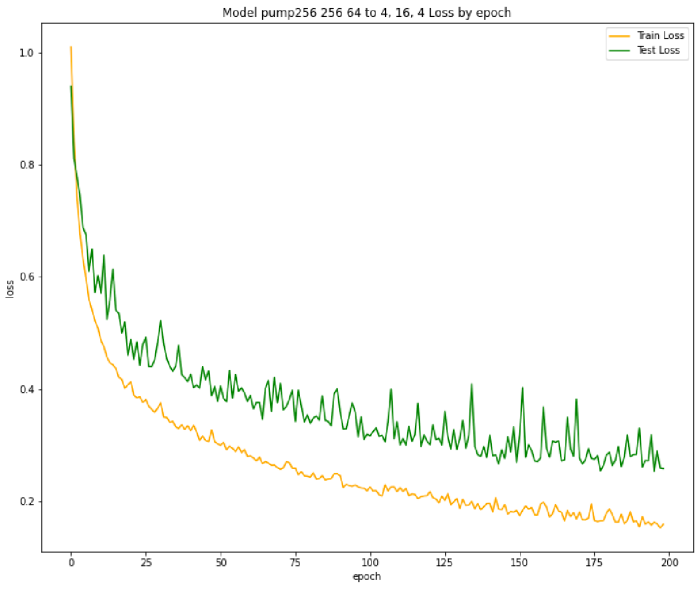
Nous avons fait varier plusieurs paramètres :

- 1 à 3 couches Denses cachées avant la couche de sortie
- 64 à 1024 Neurones par couche
- Avec et Sans Dropout
- Différents optimizers : Adam, Adagrad

L’exemple ci-dessous représente l’architecture utilisée sur les machines fan et pump à une seule couche dense cachée dans la partie classificateur :

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_11 (Conv2D)           (None, 256, 256, 64)      640
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 128, 128, 64)      0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 128, 128, 32)      18464
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 64, 64, 16)        4624
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 32, 32, 16)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 32, 32, 8)         1160
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 16, 16, 8)         0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 16, 16, 4)         292
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 8, 8, 4)           0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 256)               65792
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 1028
=================================================================
Total params: 92,000
Trainable params: 66,820
Non-trainable params: 25,180
_________________________________________________________________
```

L’évolution de la perte sur le train set et le set de validation au fil des epochs de l’entraînement de la machine pump est représentée ci-dessous :



La subtilité de classifier le type de son à partir de la qualité des prédictions “bonnes” vs “mauvaises” consiste à déterminer dans quel cas on considère que la prédiction est “bonne” et dans quel cas celle-ci est “mauvaise”. Dans un premier temps nous avons testé d’effectuer cette classification de manière très simple : si l’ID prédite n’est pas la bonne alors la prédiction est mauvaise et le fichier anormal. En générant une matrice de confusion entre vrai labels et labels prédits sur le jeu de données test des machines fan, slider et pump comportant des anomalies nous avons remarqué le problème suivant : le modèle se trompe pour seulement 30 à 50% des fichiers anormaux ce qui n’est pas assez pour pouvoir classifier le fichier comme anormal s’il est mal prédit pour son ID. Les résultats obtenus avec la machine pump sont présentés ci-dessous.

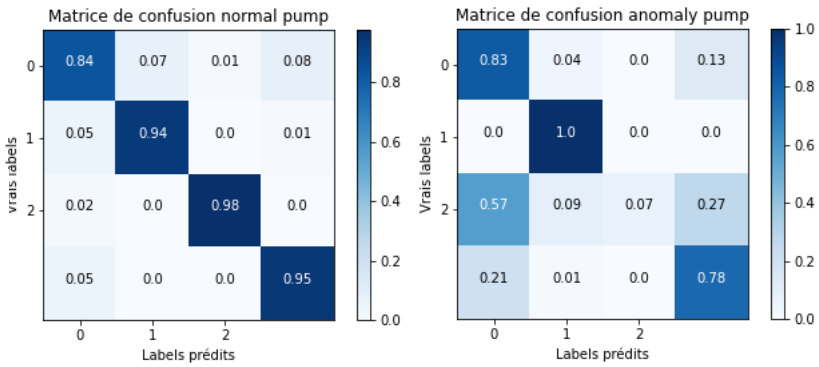
Pump :

Proportion de fichiers normaux mal classés : 7.25 %

Proportion de fichiers anormaux mal classés : 30.48 %

	precision	recall	f1-score	support
0	0.88	0.84	0.86	100
1	0.93	0.94	0.94	100
2	0.99	0.98	0.98	100
3	0.91	0.95	0.93	100
accuracy			0.93	400
macro avg	0.93	0.93	0.93	400
weighted avg	0.93	0.93	0.93	400

	precision	recall	f1-score	support
0	0.60	0.83	0.70	143
1	0.87	1.00	0.93	111
2	1.00	0.07	0.13	100
3	0.64	0.78	0.70	102
accuracy			0.70	456
macro avg	0.78	0.67	0.62	456
weighted avg	0.76	0.70	0.63	456

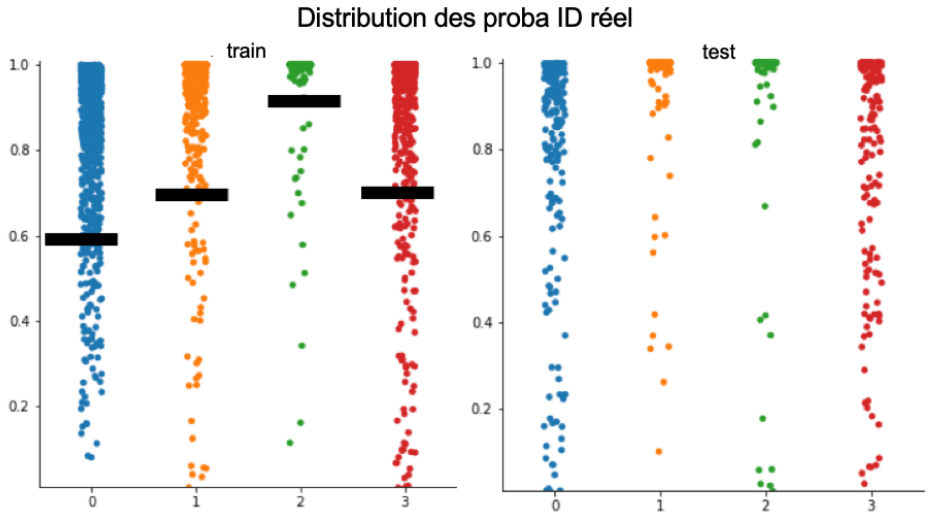


En générant une matrice de confusion entre vrai labels et labels prédits sur le jeu de données test des machines fan, slider et pump comportant des anomalies nous avons remarqué les problèmes suivants :

1. Le modèle devant prédire 4 classes, la probabilité d’une classe devient majoritaire au-dessus de 25%. L’impact d’une chute de probabilité n’aura pas forcément la même incidence sur le classement si la certitude associée à l’ID de base est forte ou faible. Le modèle ne prend donc pas en compte cette chute de probabilité
2. Le paradoxe de l'entraînement. Plus le modèle s'entraîne, plus il renforce ses certitudes sur la détection d’ID normaux, faisant de ce fait monter la densité de probabilité associée. Toutefois, cette certitude devenant forte, l’impact du son anormal n’est plus suffisant pour générer un défaut de classement.

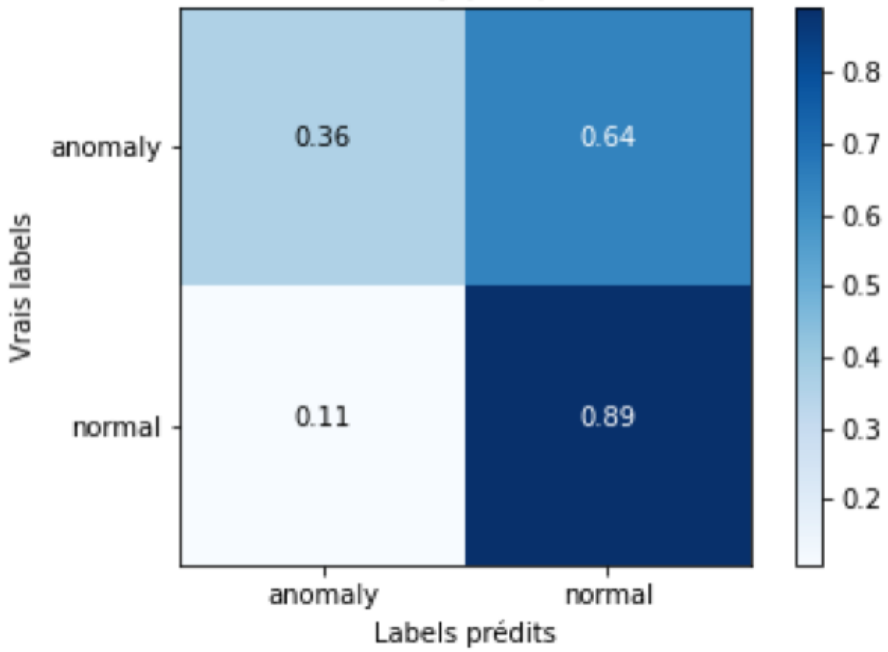
Dans le but de pouvoir travailler sur une chute de précision plutôt que sur une mauvaise prédiction nous avons visualisé les distributions de probabilité en sortie du modèle pour chaque véritable ID en entrée. Les figures ci-dessous représentent la valeur de la sortie (softmax) pour le label

associé en entrée respectivement pour les set train et test de pump. Dans ce cas-ci nous avons choisi de classifier comme anomalie les fichiers prédits par le modèle comme étant l'ID0 avec une probabilité < 0.6, les fichiers prédits par le modèle comme étant l'ID1 avec une probabilité < 0.7, les fichiers prédits par le modèle comme étant l'ID2 avec une probabilité < 0.9, et les fichiers prédits par le modèle comme étant l'ID3 avec une probabilité < 0.7.



La matrice de confusion des classes normal vs anomaly qui découle de ces seuils et le score AUC correspondant sont présentés ci-dessous. On peut conclure que le modèle ne détecte pas l'anomalie dans des proportions satisfaisantes.

AUC : 62.72 %

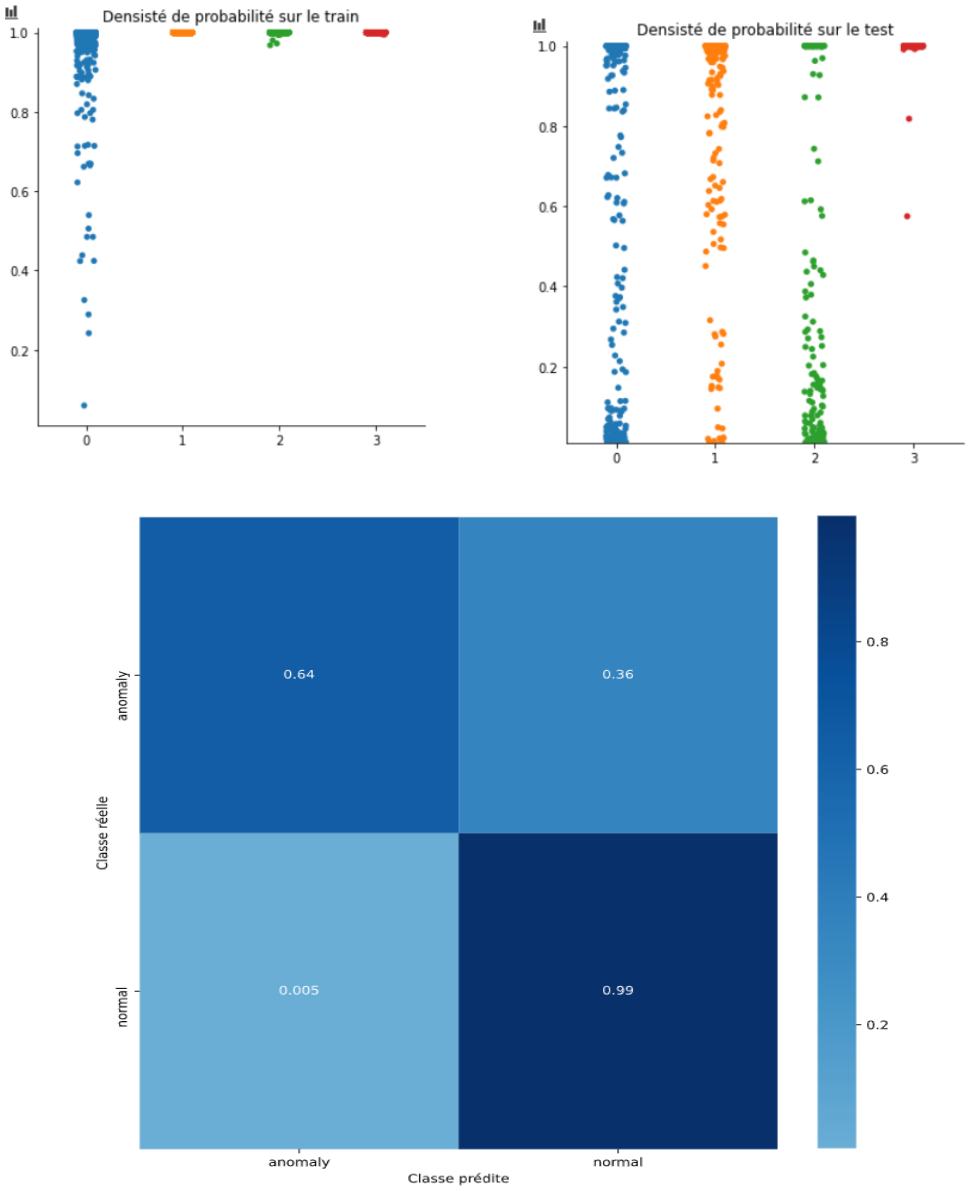


4- Modèle de classification des ID de type RNN à nbr_ID + 1 neurones en sortie

Afin d'améliorer la catégorisation et la fluidité de l'entraînement nous avons essayé de nous passer d'AutoEncoder. Pour cela nous nous sommes tournés vers des cellules qui prennent en compte une dimension supplémentaire : le temps. Nous avons donc utilisé des RNN en utilisant directement les données du train afin de conserver l'information de la temporalité. Nous avons ensuite fait varier plusieurs paramètres :

- 2 à 3 couches RNN cachées avant la couche de sortie
- Ajouter 1 à 2 couches de Dense entre les RNN et la sortie
- 64 à 1024 Neurones par couche
- Avec et Sans batch_normalization
- Avec et sans dropout / avec et sans recurent_dropout
- Différents optimizers : Adam, Adagrad

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru (GRU)                    (None, 256, 64)           61824
_________________________________________________________________
batch_normalization (BatchNo (None, 256, 64)           256
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 256, 64)           0
_________________________________________________________________
gru_1 (GRU)                  (None, 64)                24960
_________________________________________________________________
batch_normalization_1 (Batch (None, 64)                256
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 64)                4160
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 195
=================================================================
Total params: 91,651
Trainable params: 91,395
Non-trainable params: 256
```



Les répartitions sur le train et le test de la machine slider sont représentées ci-dessus en haut à gauche et à droite respectivement. Nous pouvons en déduire des seuils différenciés par ID. Sur les ID ‘1’, ’2’, ’3’ nous pouvons simplement mettre un seuil de 0.95. Concernant l’ID 0, la définition est plus difficile car la définition du seuil influera directement la détection de fausses anomalies. Les seuils finaux utilisés sont respectivement 0.4, 0.95, 0.95, 0.95 pour les ID ‘0’, ‘1’, ’2’, ’3’. Ces seuils nous ont permis d’obtenir la matrice de confusion ci-dessus entre vrais labels et labels prédits pour le type de son normal vs anormal.

Difficultés rencontrées lors du projet

Nous avons rencontré plusieurs difficultés, que nous pouvons classer dans différentes catégories:

Humaines

La première des difficultés est le travail d'équipe dans un temps restreint sur un domaine qu'aucun d'entre nous ne maîtrise. Il a fallu arriver à se connaître et à se comprendre avant de pouvoir se lancer dans le projet, ce qui en un mois est déjà un premier challenge en soit ! Heureusement, notre bonne humeur à tous et notre envie de travailler ensemble nous ont vite permis de surmonter cette première étape. Une fois ce cap passé nous avons pu commencer à utiliser nos expériences professionnelles très diverses pour amener des visions complémentaires aux problèmes.

Organisationnelles

Au-delà du rythme de la formation à tenir en parallèle du projet, nous avions tous des contraintes liées à la recherche d'emploi ou à un emploi à réaliser en parallèle. Ajoutez à cela les contraintes familiales de chacun et la nécessité de travailler à distance, et nous arrivons à la seconde difficulté. De ce fait, chacun avançait à un rythme différent sur les modules d'une semaine sur l'autre. Les modules étant tantôt très pertinents vis à vis du projet tantôt très éloigné, il était au début difficile d'avancer tous ensemble, nous avons donc mis en place des échanges réguliers via slack et zoom afin d'avoir tous le même niveau d'information sur le projet.
Nous avons également dû apprendre à utiliser des outils de code très différents (Jupyter, Collab, GitHub,VSC) tout en essayant de comprendre comment aller articuler notre projet et donc quels aller être nos futurs besoins. Nous avons réalisé un certain nombre d'aller retours sur des environnements de travail locaux et cloud selon les phases et les difficultés du projet. Cela a mis à rude épreuve l'organisation de chacun et du projet, devant à chaque fois naviguer entre les différents environnements et les données qui y étaient stockées.
Nous avons donc pu apprendre à utiliser plusieurs compilateurs différents, des systèmes locaux et sur le cloud, le tout avec et sans outils de calcul de différence.

Techniques

Au début nos algorithmes étaient simples, et notre connaissance limitée, techniquement nos limites étaient donc le temps de chargement du jeu de données (9Go) qui au-delà de la place prise sur le disque, demandait un temps de traitement de 30-50min avant de pouvoir être utilisé. Nous avons réussi avec le temps à contourner le problème de temps de traitement en mettant en place des fichiers npy prétraités, mais pas la problématique de la dimension du jeu de données.

Très vite la taille de ce dernier nous a posé un nouveau problème : l'impossibilité de le stocker dans son intégralité sur le cloud. Nos machines locales étant limitées en puissance de calcul, nous avons donc une double limitation qui nous a obligés à traiter les données machine par machine au lieu de traiter toute la donnée ensemble. De même cette capacité de calcul limitée nous a obligé à bâtir des modèles au plus juste de notre besoin, l'ajout de dimension nous renvoyant immédiatement une erreur de puissance disponible.

Cela nous a permis de découvrir deux axes importants de la DS, des systèmes afin de stocker des données prétraitées et les charger ainsi que la recherche d'hyper paramètres plutôt que le renforcement de la complexité du réseau.

Expérience

Enfin l'objectif premier du projet étant d'acquérir de l'expérience sur les processus de machine learning et sur les environnements associés il était naturel qu'au début aucun d'entre nous n'avait d'expérience solide dans ce domaine. Toutefois, sans 'sachant' dans l'équipe pour nous guider, le choix de la direction n'était pas évident et les aller retours se faisaient au fil des découvertes et des résolutions de problèmes.

Grâce aux erreurs réalisées au fil du projet nous savons un peu plus aujourd'hui comment nous aurions guider le démarrage, le temps et, les ressources que nous aurions accordé à chaque étape.

Suite du projet

Une piste d'ouverture pourrait être un dispositif (type Raspberry Pi équipé d'un micro) configuré pour réaliser cette tâche.