Nema Motor RPM Estimation

Félix Bouchard (111 160 393)
Francis Verreault (111 158 102)
Loïc Amann (537 307 053)

Laval University

December 16, 2024

IFT-7030 Machine Learning for Signal Processing (MLSP)

1. ABSTRAIT

Les servomoteurs, avec leur contrôle précis et leur robustesse, sont devenus incontournables en robotique moderne. Cependant, leur coût élevé limite leur accessibilité pour les projets amateurs. Ce travail explore une alternative prometteuse : doter les moteurs pas-à-pas d'un système de rétroaction comparable à celui des servomoteurs, à moindre coût. L'objectif est de développer un prototype capable de déterminer en temps réel la vitesse de rotation (RPM) de moteurs pas-à-pas, permettant ainsi un ajustement dynamique des instructions et une amélioration significative de leur polyvalence, précision et résilience. Pour ce faire, nous proposons une approche basée sur l'analyse de signaux audio et les techniques d'apprentissage automatique afin d'estimer la vitesse de rotation en temps réel.

2. INTRODUCTION

Comme mentionné précédemment, l'objectif principal de ce projet est de développer un système de rétroaction abordable pour les moteurs pas-à-pas, en utilisant l'analyse audio et l'apprentissage automatique. Les microphones modernes permettent d'atteindre facilement des taux d'échantillonnage de 80 kHz, ce qui offre une résolution temporelle suffisante pour déterminer la position des moteurs pas-à-pas en temps réel, si notre système s'avère efficace. Cependant, pour notre prototype, nous allons nous concentrer sur des temps de réaction de l'ordre de 0,1 à 0,8 seconde. Cette simplification vise à réduire la complexité du problème et à minimiser les coûts associés au matériel et à l'entraînement des modèles. Notre approche s'articulera autour de quatre axes principaux : (1) la collecte des données audio, (2) le développement d'un modèle de noyau à régression logistique, (3) l'exploration d'un modèle de perceptron multicouche, et (4) l'analyse du spectrogramme audio pour extraire des caractéristiques pertinentes.

2.1. Collecte des données audio

La collecte de données a été réalisée à l'aide d'un microphone MI-305 et d'un moteur NEMA 42. Le microphone, fixé à l'extrémité du moteur, a enregistré ce dernier en fonctionnement continu pendant environ 100 heures. Un script Python (generationgcode.py) a été développé pour générer automatiquement des instructions G-code. Ces instructions commandaient au moteur de tourner dans un seul sens, avec une vitesse variant entre 0 et 120 pouces par minute.

Les modèles utilisés sont tous de type supervisé, nécessitant des étiquettes associées aux enregistrements audio. Pour générer ces données étiquetées, un plugin Mach3 a été développé. Ce plugin permet de synchroniser l'enregistrement audio avec le relevé des positions du moteur. Grâce à l'API offerte par Mach3, les positions ont été obtenues à une fréquence d'environ 10 Hz (toutes les 0,1 s). Ce dataset a été rendu public et est disponible sur hugginface :[2]

3. NOYEAU RÉGRESSION LOGISTIQUE

Le modèle de régression logistique à noyau (NRL) a été sélectionné en raison de sa capacité à capturer des relations non linéaires. En effet, le son émis par un moteur est influencé par de multiples facteurs tels que la vitesse, la charge, et l'usure, rendant la relation entre le son et les RPM complexe. Le NRL permet de projeter les données dans un espace de plus grande dimension où elles deviennent linéairement séparables.

Plus précisément, le NRL utilise une astuce, appelée "kernel trick", pour calculer les distances entre les points de données dans cet espace de haute dimension sans avoir à effectuer explicitement la transformation. Cette approche est particulièrement efficace pour traiter des relations non linéaires complexes.

L'implémentation de cette technique est détaillée dans le fichier KernelLogisticRegressionExploration.ipynb[1]
Les données audio ne sont pas directement utilisées dans le modèle. Premièrement, les données sont normalisées, puis des features MFCC, spectral centroid et RMS sont extraites à l'aide de Librosa. Finalement, ces features sont renormalisées avant d'être utilisées dans le modèle. À noter que l'utilisation de features mel-spectrogram avait pour effet de grandement réduire les performances du modèle.

3.1. Noyeau régression logistique - Résultats

Après optimisation des hyperparamètres en utilisant uniquement 12h d'audio avec des segments de 0.8s, les résultats obtenus sont les suivants :

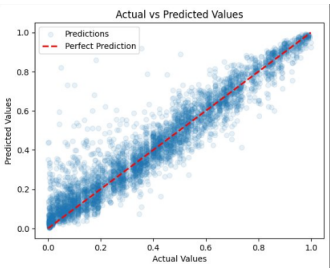

Fig. 1. Répartition NRL sur diagramme

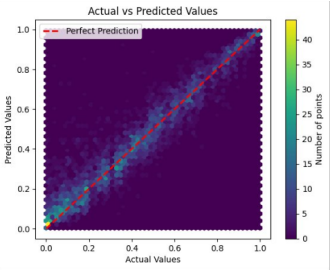

Fig. 2. Répartition avec une autre réprésentation

Métriques petit dataset	**Valeur**
Erreur absolue moyenne (MAE)	0.0566
Erreur quadratique moyenne (MSE)	0.0069
Racine de l'erreur quadratique moyenne (RMSE)	0.0832
Coefficient de détermination (R^2)	0.9042
Erreur absolue médiane (MedAE)	0.0399

Table 1. Métriques du modèle NRL 12h

En utilisant les même hyperparamètres et features de l'espace de dimension supérieur l'entrainement a été éffectuée sur tout le data 100h.

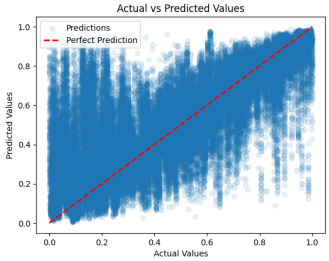

Fig. 3. Répartition NRL sur diagramme, dataset complet

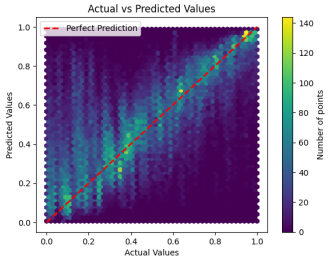

Fig. 4. Répartition avec une autre réprésentation

Métriques dataset complet	**Valeur**
Erreur absolue moyenne (MAE)	0.1505
Erreur quadratique moyenne (MSE)	0.0449
Racine de l'erreur quadratique moyenne (RMSE)	0.2120
Coefficient de détermination (R^2)	0.4049
Erreur absolue médiane (MedAE)	0.1023

Table 2. Métriques du modèle NRL sur tout le dataset

On remarque que les résultats sont beaucoup moins intéressant lorsqu'on utilise plus de données. C'est probablement dû aux hyperparamètres et features rerpésentant l'espace de dimension supérieur qui sont adapté que sur une partie des données.

3.2. Noyeau classification logistique - Résultats

Face aux performances insuffisantes du modèle de régression à noyau, la tâche a été simplifiée en un problème de classification à 100 classes. Cette simplification vise à évaluer si une amélioration significative des performances peut être obtenue en réduisant la complexité du problème. Le modèle de classification (KernelClassificationModel.py) a été entraîné sur environ 15% des données disponibles. Voici les résultats obtenus :

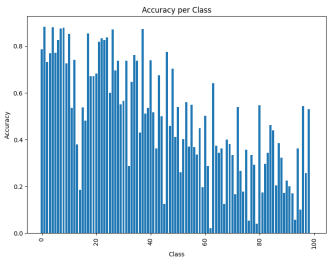

Fig. 5. Répartition de la précision

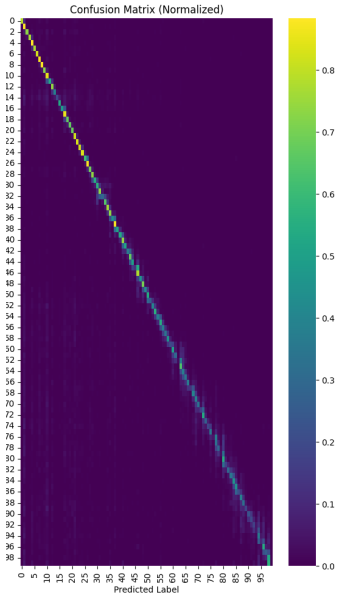

Fig. 6. Matrice de confusion normalisée

Métriques moyenne pour dataset partiel(12h)	**Valeur**
Accuracy	0.5515
Precision	0.5434
Recall	0.4869
F1-score (R^2)	0.4859

Table 3. Métriques du modèle NCL sur tous le dataset

On remarque à l'aide de la matrice de confusion que les résultats sont beaucoup plus prometteurs que ceux du modèle de régression. Les métriques moyennes ne mène pas aux même conclusion et il faudrait donc calculer des métriques pondérés pour représenter adéquatement les performances de façon numérique.

4. MODÈLE DE RÉSEAU NEURONAL MULTICOUCHE (MLP)

L'estimation des vitesses des moteurs NEMA à partir des signaux audio a été effectuée à l'aide d'un modèle de perceptron multicouche (MLP) développé avec PyTorch. Ce modèle se compose de trois couches linéaires : une couche d'entrée (`fc1`) comprenant 64 neurones, une couche cachée (`fc2`) avec 32 neurones, et une couche de sortie (`fc3`) comportant un seul neurone. Les fonctions d'activation ReLU ont été utilisées entre les couches pour introduire la non-linéarité, tandis qu'une fonction sigmoïde est appliquée à la sortie pour normaliser les prédictions dans l'intervalle [0, 1].

4.1. Prétraitement des données

Les données audio ont été enregistrées à une fréquence d'échantillonnage de 16 kHz et segmentées en tranches synchronisées avec les vitesses mesurées des moteurs. Chaque segment a été transformé en coefficients MFCC (Mel Frequency Cepstral Coefficients), extraits à l'aide de la bibliothèque Librosa. Ces coefficients fournissent une représentation compacte des caractéristiques audio pertinentes pour la tâche d'estimation. Les données ont ensuite été normalisées à l'aide d'un scaler pour garantir une échelle uniforme des caractéristiques.

4.2. Architecture et entraînement du modèle

Le modèle a été optimisé à l'aide de l'algorithme Adam, avec une fonction de coût basée sur l'erreur quadratique moyenne (MSE). L'entraînement a été réalisé sur un ensemble de données partitionné en 80% pour l'entraînement et 20% pour le test, avec un traitement par lots de 32 échantillons. Dix époques ont été effectuées pour minimiser la fonction de perte et ajuster les poids du réseau.

4.3. Performances et analyse des résultats

Les performances du modèle ont été évaluées sur l'ensemble de test à l'aide de plusieurs métriques standards, obtenant les résultats présentés dans la table 4.

Métriques	**Valeur**
Erreur absolue moyenne (MAE)	0.0211
Erreur quadratique moyenne (MSE)	0.0042
Racine de l'erreur quadratique moyenne (RMSE)	0.0651
Coefficient de détermination (R^2)	0.9421
Erreur absolue médiane (MedAE)	0.0094

Table 4. Métriques du modèle MLP sur l'ensemble de test

Ces métriques indiquent une forte corrélation entre les prédictions du modèle et les vitesses réelles, confirmant la fiabilité de l'approche MLP pour cette tâche.

4.4. Visualisation des résultats

Les figures 7 et 8 présentent graphiquement les performances du modèle. Ces figures illustrent la répartition des prédictions du modèle par rapport aux valeurs réelles, avec une ligne de parfaite prédiction indiquant l'idéal théorique. La densité des points près de cette ligne reflète la précision du modèle.

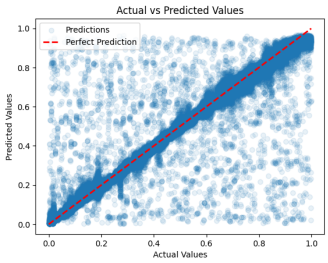

Fig. 7. Répartition de AST sur diagramme

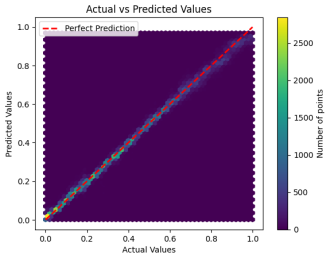

Fig. 8. Répartition avec une autre réprésentation

5. MODÈLE AUDIO SPECTROGRAM TRANSFORMER

Dans le cadre de ce projet, nous avons utilisé le modèle Audio Spectrogram Transformer (AST) pour prédire à partir de fichier audio. AST est un modèle de classification ou de régression sur des fichiers audio qui applique l'architecture Transformer à des spectrogrammes (représentation visuelle de signaux audio). Cette application de AST à notre sujet à impliquer une segmentation des fichiers audio (pour des questions de performances de computing), et l'entrainement du modèle pour prédire une variable continue (vitesse du moteur).

5.1. Objectif de l'implémentation de AST

Cette partie du projet a pour objectif d'appliquer un modèle Transformer adapté à l'audio pour résoudre un problème de régression où l'on prédit la vitesse des moteurs NEMA à partir d'enregistrements audio. Nous cherchons à étudier les variantions de résultats avec nos autres modèles, en appliquant à notre projet un modèle pré-existant.

5.2. Architecture du modèle AST

Le modèle AST (Audio Spectrogram Transformer) est une architecture basée sur le Transformer, adaptée pour le traitement de données audio. Le Transformer est une architecture largement utilisée en traitement du langage naturel et dans de nombreuses autres tâches de machine learning. Le modèle AST transforme l'audio en un spectrogramme (une représentation visuelle de l'intensité des différentes fréquences du son au cours du temps) et applique le Transformer à cette représentation.

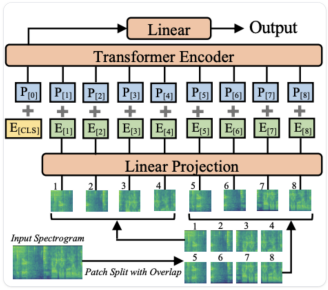

Fig. 9. Architecture du modèle AST

Les étapes suivantes sont typiquement suivies pour entraîner le modèle AST :

- Prétraitement de l'audio : Les fichiers audio sont transformés en spectrogrammes.
- Extraction des caractéristiques : L'extracteur de caractéristiques (feature extractor) est utilisé pour extraire des vecteurs de caractéristiques à partir des spectrogrammes.
- Entraînement du modèle : Un modèle AST pré-entraîné est utilisé et affiné pour prédire des classes spécifiques correspondant aux vitesses des moteurs.

5.3. Prétraitement des données

5.3.1. Segmentation des fichiers audio et Normalisation

Avant d'appliquer AST, les fichiers audios sont segmentés en petites portions. Cela est nécessaire pour traiter des fichiers de

grande taille et pour obtenir une entrée stable pour le modèle. Pour ce faire, nous utilisons la fonction $segment_audio$ qui divise l'audio en segments basés sur des timestamps provenant d'un fichier de métadonnées. Les segments sont ensuite utilisés pour calculer les vitesses du moteur associées à chaque segment. Enfin on normalise les segments et les vitesses pour extraire les features.

5.3.2. Extraction des spectrogrammes

Une fois les segments audios prêts, l'extracteur de caractéristiques d'AST est utilisé pour convertir ces segments en spectrogrammes. L'extracteur de caractéristiques est utilisé pour transformer chaque segment en un vecteur de caractéristiques (features) qui sera ensuite utilisé pour entraîner le modèle.

5.4. Entrainement

Le modèle Audio Spectrogram Transformer (AST) a été pré-entraîné sur AudioSet, un large ensemble de données étiqueté avec des événements sonores provenant de vidéos sur YouTube. AudioSet contient plus de 2 millions d'exemples audio couvrant 527 classes d'événements sonores. AST a été évalué sur différents benchmarks de classification audio, où il atteint de nouveaux résultats de pointe de 0,485 mAP sur AudioSet, 95,6% de précision sur ESC-50, et 98,1% de précision sur Speech Commands V2.

5.4.1. Chargement et préparation du modèle

Nous avons utilisé le modèle Audio Spectrogram Transformer (AST) pré-entraîné à partir de la bibliothèque transformers. Le modèle AST, initialement conçu pour des tâches de classification audio, est chargé à l'aide de la fonction $ASTForAudioClassification.from_pretrained$. Cependant, comme notre tâche consiste à prédire une variable continue (la vitesse du moteur), nous avons ajusté la couche de sortie du modèle pour effectuer une régression plutôt qu'une classification ($model.classifier = nn.Linear(in_features, 1)$). Nous avons remplacé la couche de classification par une couche linéaire, qui est plus adaptée aux prédictions continues. Le modèle est alors prêt à prédire les vitesses normalisées des moteurs à partir des spectrogrammes audios.

5.4.2. Optimisation et calcul de la perte

Pour l'optimisation, nous avons utilisé l'optimiseur Adam, qui est particulièrement efficace pour les tâches de régression, car il ajuste les taux d'apprentissage de manière dynamique. La fonction de perte choisie pour l'entraînement est l'erreur quadratique moyenne (MSE), qui est largement utilisée dans les tâches de régression. Cette fonction permet de calculer l'écart entre les valeurs prédites et les valeurs réelles (les vitesses des moteurs), et d'ajuster les poids du modèle en fonction de cette erreur.

5.4.3. Entrainement et évaluation

Le modèle prend beaucoup de ressource de calcul à l'ordinateur qui exécute le programme. Par rapport au deux autres modèles présentés, nous avons dû réduire le nombre d'epochs, la taille des batchs et réduire la qualité des spectrogrammes.

L'entraînement du modèle se fait sur plusieurs époques, où à chaque époque, les prédictions du modèle sont calculées et comparées aux valeurs réelles des vitesses normalisées. À chaque étape, l'erreur est calculée et les poids du modèle sont ajustés pour réduire cette erreur. À la fin de chaque époque, le modèle est évalué sur un jeu de test pour mesurer sa performance en termes d'erreur quadratique moyenne (MSE), qui est la métrique principale de notre tâche de régression.

5.5. Résultats

Métrique	**Valeur**
Mean Absolute Error (MAE)	0.0516
Mean Squared Error (MSE)	0.0040
Root Mean Squared Error (RMSE)	0.0633
R-squared Score (R^2)	0.9432
Median Absolute Error (MedAE)	0.0454

Table 5. Métriques du modèle AST sur l'ensemble de test

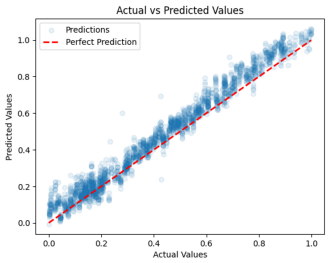

Fig. 10. Répartition de AST sur diagramme

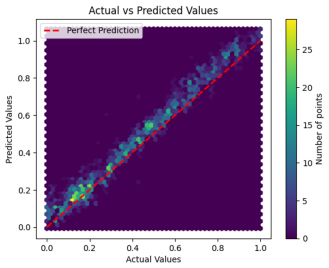

Fig. 11. Répartition avec une autre réprésentation

5.6. Améliorations

5.6.1. Fine-tuning

Le fine-tuning peut être amélioré en ajustant spécifiquement certaines couches du modèle, plutôt que d'entraîner l'ensemble du modèle. Par exemple, nous pourrions choisir de n'entraîner que les couches finales du modèle tout en laissant les premières couches fixes. Cela permettrait de mieux exploiter les connaissances déjà acquises par le modèle pré-entraîné tout en l'adaptant à notre tâche spécifique. Ce processus de fine-tuning plus ciblé peut réduire le temps d'entraînement tout en préservant la performance.

5.6.2. Augmentation des données d'entrainement

Nous avons observé que le modèle AST a été pré-entraîné sur des ensembles de données comme AudioSet, qui couvre une

large gamme d'événements sonores. Toutefois, il pourrait être pertinent d'envisager un pré-entraînement from scratch sur un ensemble de données plus spécifique à notre application. En utilisant des enregistrements audio de moteurs électriques provenant de notre propre dataset, complétés par des sons issus de vidéos YouTube ou d'enregistrements industriels (comme ceux provenant d'usines), nous pourrions enrichir le modèle avec des données acoustiques plus proches de celles qu'il rencontrera lors de l'application en production. Cela permettrait de mieux adapter les poids du modèle à notre domaine spécifique, optimisant ainsi les performances et améliorant la capacité du modèle à généraliser sur des sons similaires aux nôtres.

Une autre amélioration potentielle consiste à appliquer des techniques d'augmentation des données audio. Cela peut inclure des transformations telles que l'ajustement de la vitesse de lecture de l'audio, l'ajout de bruit de fond ou la modification de la hauteur des sons. L'augmentation des données permettrait d'améliorer la généralisation du modèle, en lui permettant d'apprendre à partir d'un éventail plus large de scénarios. Cette méthode est particulièrement utile lorsque le nombre d'exemples d'entraînement est limité.

6. CONCLUSION

En conclusion, les trois modèles ont produit des résultats prometteurs. Cependant, une étape importante n'a pas été abordée dans ce projet : vérifier si les modèles sont suffisamment rapides pour une utilisation en temps réel. En d'autres termes, il est essentiel de s'assurer que le temps nécessaire au modèle pour analyser les données est inférieur à la durée de ces données elles-mêmes.

References

[1] Loïc Amann Francis Verreault Félix Bouchard. *ProjectRepo*. 2024. URL: https://github.com/verreaultfrank/NemaMotorRpmEstimation.

[2] Francis Verreault. *Nema42 Audio DataSet*. 2024. URL: https://huggingface.co/datasets/captainfr4nk/StepperMotorSoundsWithLabels.