PANNETIER Léonard
LEDIG Ian

Music theme recognition

Introduction

Dans le cadre de notre projet MSLP, nous avons pour objectif de développer un classificateur de styles musicaux. Nous prévoyons de constituer un ensemble de données audio musicales avec 4 styles de chaque musique prédéfinie (classique, techno, rap, rock) étiquetée, et d'entraîner un réseau de neurones pour prédire le style de musique d'une musique donnée.

Il existe déjà des Intelligences artificielles pour trouver le style d'une musique, comme Music Genre Finder, développé par Chosic, mais nous n'en avons pas trouvé de open source et performante à cet effet.

Modèles

Nous avons créé deux approches différentes :

- Perceptron multicouche
- Modèle gaussien

Ensemble de données

Nous avons créé l'ensemble de données nous-mêmes. Nous avons tout d'abord fait de premiers essais avec un échantillon de 25 musiques de chaque style, d'environ 3 minutes de longueur, soit 100 musiques au global. La répartition des morceaux est homogène (autant de morceaux pour chaque style).

Pour se procurer nos fichiers audios, nous avons utilisé le site YouTube ainsi qu'un autre site de conversion de vidéo Youtube en fichier au format mp3.

Nous avons utilisé ce dernier site pour télécharger l'ensemble de nos fichiers audio pour éviter tout problèmes liés à l'encodage et nous les avons transformés au format Mono car il n'est pas nécessaire de garder du Stéréo dans le cadre de nos entraînements de modèle.

Nous avons fait en sorte de prendre des musiques d'artistes différents pour éviter tout biais dans les résultats obtenus.

Cette fois aussi, nous avons rencontré des soucis concernant la précision donnée par les entraînements de nos modèles. Nous ne parvenions pas à avoir une précision suffisamment satisfaisante pour prédire le style de n'importe quelle musique donnée en entrée. Nous avons donc pris plusieurs décisions :

- Utiliser une transformation pour les données,
- Augmenter la taille du dataset,
- Découper chaque fichiers audios en plusieurs sous fichiers, ce qui augmente considérablement la taille de notre dataset.

Au début, nous donnions directement les données audio sans transformation mais les résultats étaient décevants, nous avons donc testé les transformations suivantes :

- La transformation de Fourier :

Chargement de l'ensemble de données audio, calcul de sa transformation de Fourier avec l'algorithme FFT (Fast Fourier Transform). Ces transformations sont ensuite utilisées comme entrée pour nos deux modèles.

- Le spectrogramme :

Chargement de l'ensemble de données audio, calcul de son spectrogramme qui est une représentation visuelle de la gamme de fréquences présentes dans un signal sonore au fil du temps. Ces spectrogrammes sont ensuite utilisés comme entrée pour nos deux modèles.

En essayant ces deux méthodes de transformations, nous avons décidé de garder le spectrogramme qui répond mieux à nos attentes.

On passe désormais d'un dataset de 100 morceaux à un dataset de 200 morceaux sachant qu'en les coupants, on a un dataset d'environ 15 000 sous-morceaux.

Auparavant, un morceau prenait en moyenne un vecteur d'environ 10 000 000. En le coupant, on obtient un vecteur d'environ 15 000, ce qui facilite le traitement de chaque fichiers.

Ainsi, en prenant des passages aléatoires de chaque morceaux pour en créer des sous-morceaux, nous avons la possibilité de jauger la quantité d'entrées que nous souhaitons avoir pour l'entraînement de nos modèles, sans pour autant avoir un ensemble de données massif.

Entraînement Perceptron multicouche

Nous avons décidé de réaliser un réseau de neurones convolutifs pour prédire le style des différentes musiques.

Le CNN prendra en entrée un vecteur de dimension N (la taille des sous-morceaux de musique) et renverra un vecteur de dimension 4 où chaque composante du vecteurs sera la probabilité que le morceau d'entrée appartienne à chaque classe.

Nous avons décidé de partir sur un CNN avec 4 couches convolutives 1D. C'est un bon compromis de performance et de légèreté du modèle. Avec plus de couche convolutive, l'entraînement est trop long sur nos machines. La première couche aura une taille de noyau de 10 et les autres de 3 qui est une valeur classique. La première couche devant extraire des caractéristiques globale de l'entrée prendre un taille de noyau importante permet de réduire la taille du modèle sans prendre en précision.

Entre chaque couche convolutive, nous avons fait une réduction de dimension avec une couche de max pooling 1D. Cela fait comme un filtre de plus en plus précis permettant d'extraire des information de plus en plus locales.

Les 2 dernières couches étant des couches linéaires pour prendre la décision de la classe en fonction des sorties des couches convolutives. Nous avons testé l'utilisation d'une couche softmax pour la dernière au lieu d'une couche linéaire et les résultats sont sensiblement équivalents.

Trois entrées ont été testé (les données audio brut, les transformées de fourier (fft) et les spectrogrammes). Les données brut ont donné des résultats médiocre (35% de précision pour le jeu de test). Les fft et les spectrogrammes ont été bien plus performant.

PANNETIER Léonard
LEDIG Ian

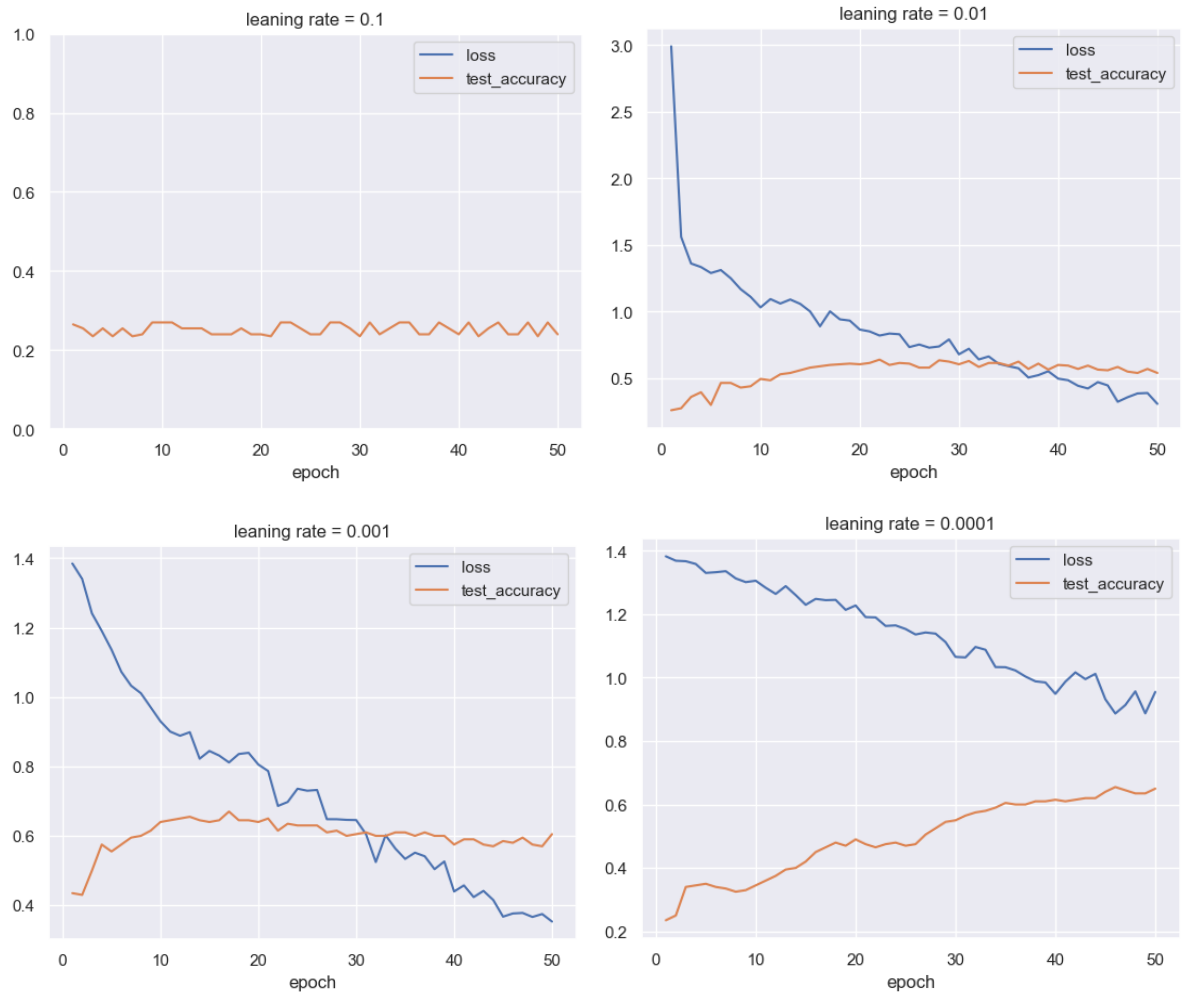
Optimisation des hyperparamètre :

Nous avons fait l'optimisation des paramètres en utilisant la transformée de fourier.

Il y a plusieurs paramètres sur lequel nous pouvons jouer :

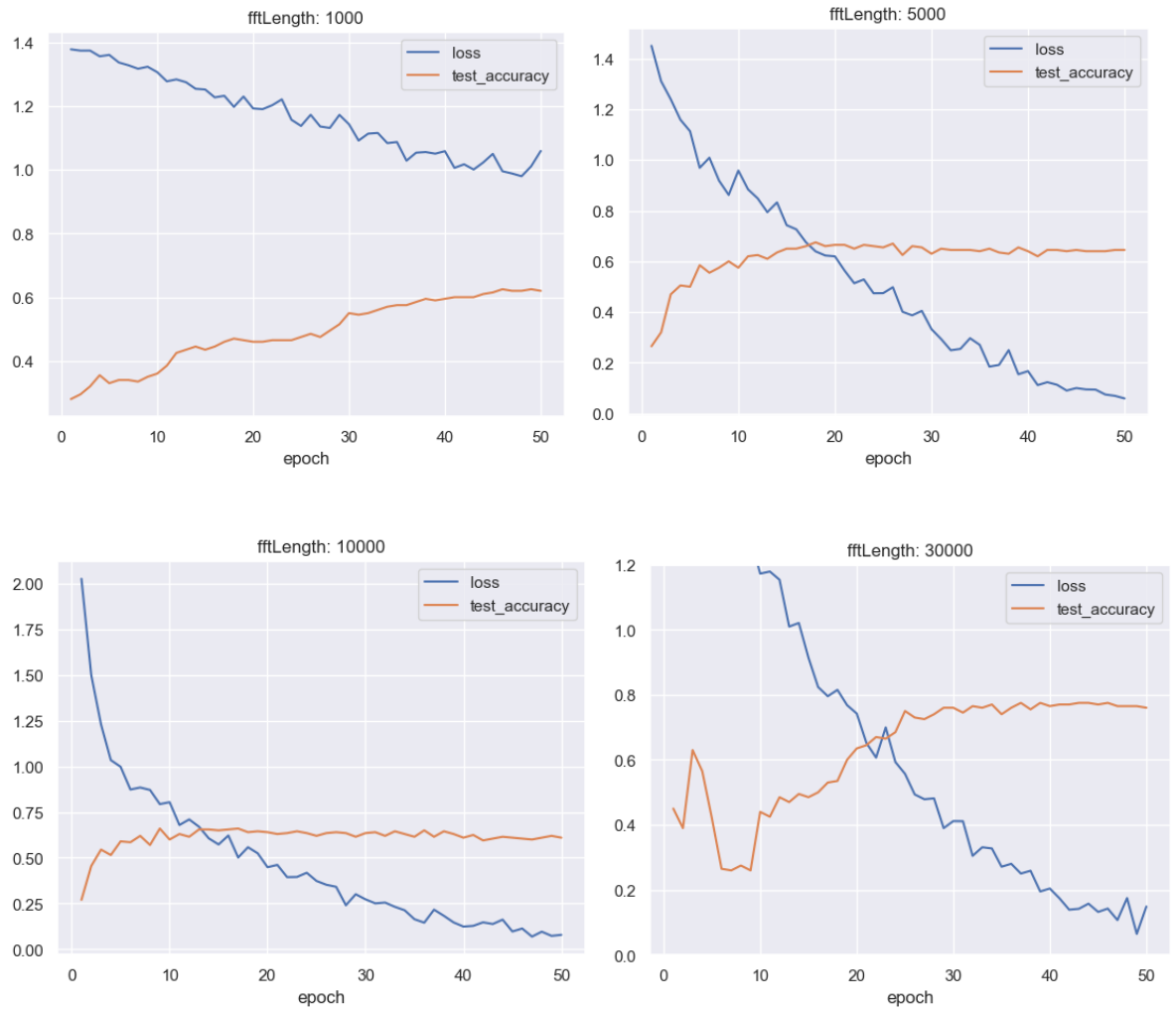
- La taille des données d'entrée (fftLength),
- La taille des mini lots (batch_size),
- Le nombre d'epoch d'entraînement,
- La taille du dataset,
- Le taux d'apprentissage (learning rate).

Commençons par le taux d'apprentissage, on veut un taux d'apprentissage le plus grand possible mais avec un modèle convergent, on trace donc la courbe de perte en fonction des epochs pour vérifier que le modèle converge pour différentes valeurs de taux d'apprentissage.



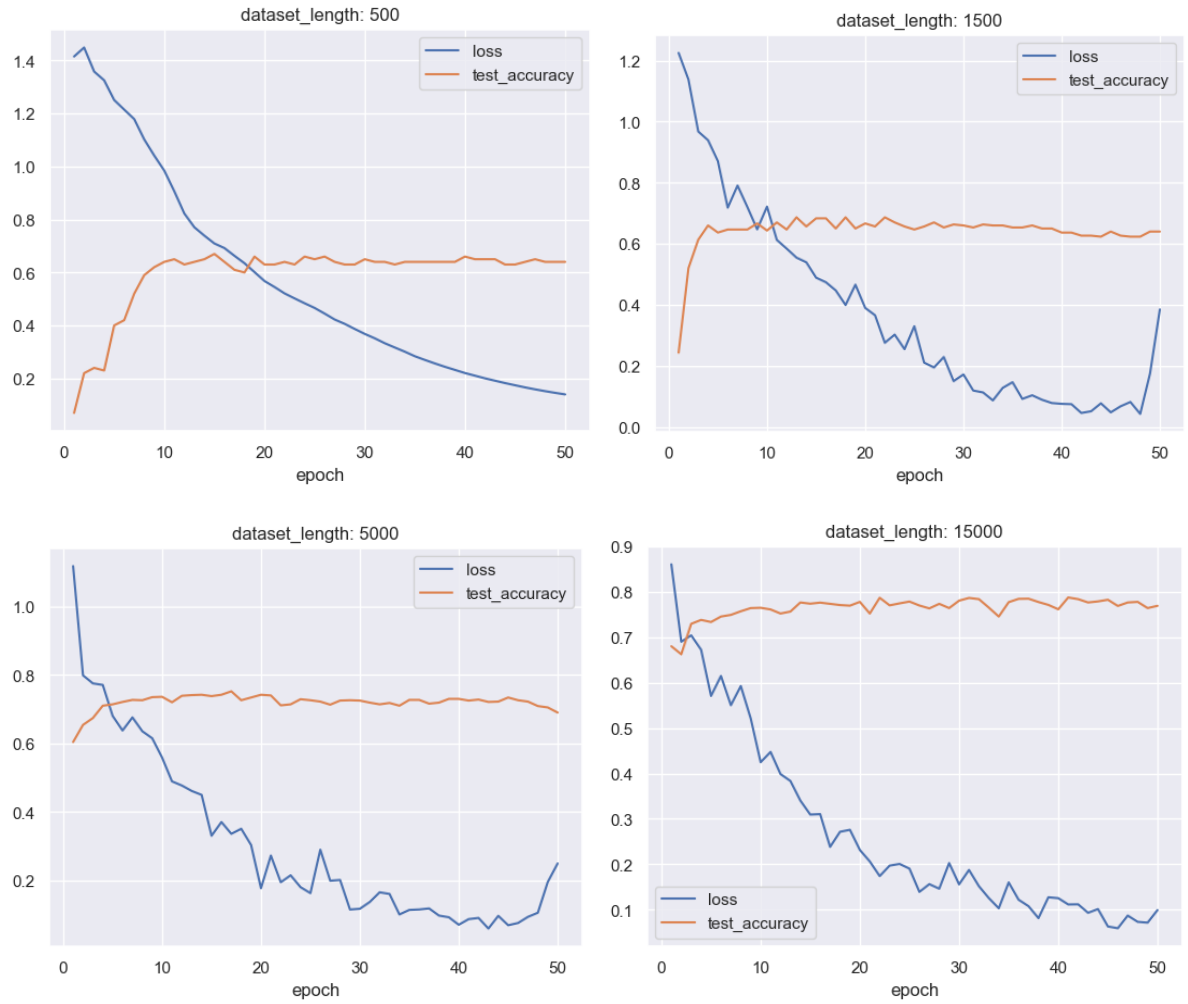
Nous remarquons qu'avec un taux d'apprentissage = 0.1 le modèle ne converge pas, avec un taux d'apprentissage = 0.01 le modèle converge mal, on va donc prendre taux d'apprentissage = 0.001.

Nous allons donc optimiser la taille du vecteur d’entrée.



Nous remarquons qu'augmenter la taille du vecteur d’entrée n’influe pas tant que ça sur les performances du modèle, pour garder un modèle léger, nous allons prendre un vecteur d’entrée de taille 5000.

Maintenant, optimisons la taille du dataset, c'est-à-dire le nombre d'entrées généré par audio.



Nous remarquons qu'à partir de 5000 les performances du modèle n'augmente plus, nous choisissons donc un dataset de 5000 sous-morceaux.

En ce qui concerne l'optimisation des hyperparamètres du spectrogrammes, c'est-à-dire la taille de la fenêtre de calcul et la taille des hop, nous avons décidé d'utiliser la fenêtre de hann ainsi qu'une hop length de 200 qui est une valeur classique.

Entraînement Gaussien

Il y a beaucoup moins d'hyper paramètres à optimiser, nous avons uniquement :

- La taille du dataset,
- Le type de matrice de covariance,
- La méthode d'initialisation,
- La taille du vecteur d'entrée.

Nous resterons ici sur 5000 comme taille de dataset pour pouvoir comparer les 2 modèles. Cependant la taille du vecteur d'entrée sera abaissé à 1000 car avec un vecteur trop grand, il faudrait une quantité de données déraisonnable pour entraîner le GMM. Ainsi nous ferons les comparatif avec les mêmes dataset pour les 2 modèles, juste les vecteurs d'entrée seront redimensionner.

En ce qui concerne la méthode d'initialisation il y a 2 choix possible, soit en initialisant aléatoirement les centres des clusters, soit en prenant les centres données par la méthodes des K-means, nous avons remarqué que en utilisant les K-means le modèle convergent plus vite c'est donc ce que nous allons utiliser.

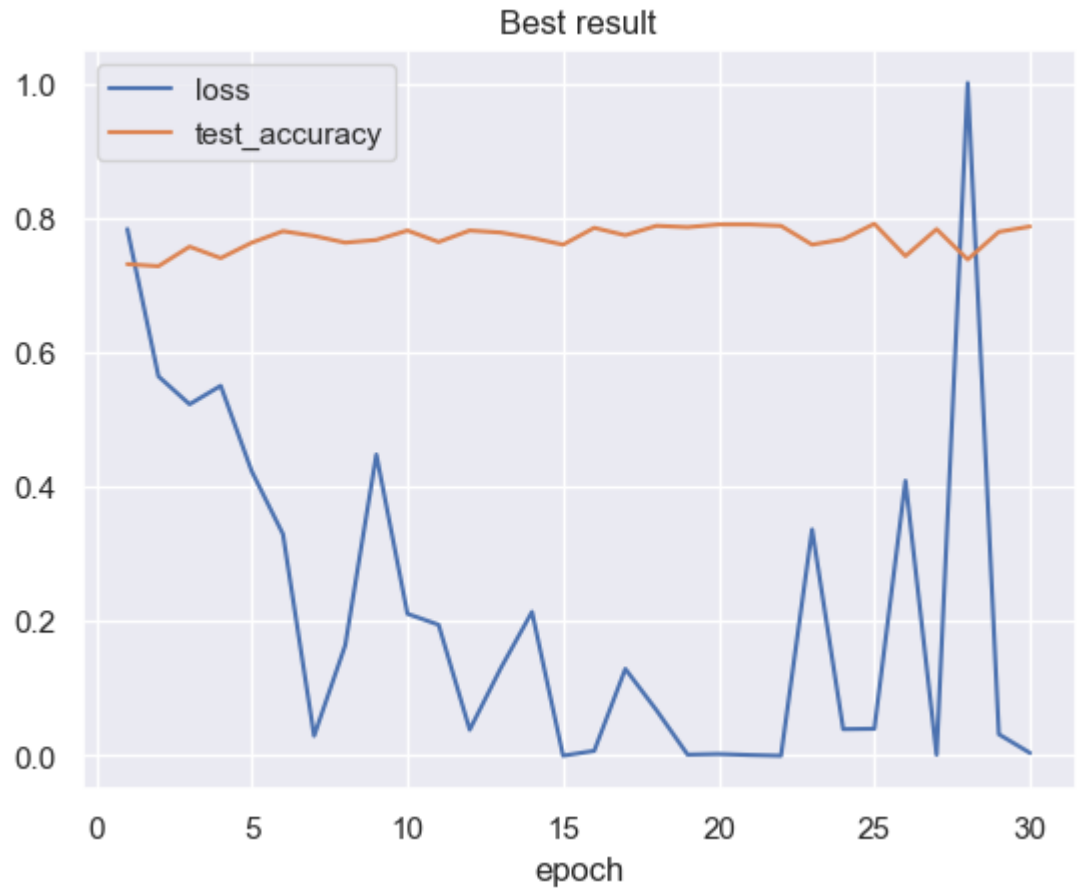
Pour les matrices de covariances, nous pouvons utiliser soit des matrices pleines différentes pour chaque clusters, soit pleines mais constantes pour les 4 clusters, soit une matrice diagonale par cluster, soit une unique variance par cluster.

Les 3 dernière méthodes ont donné des résultats médiocres, seules les 2 méthodes avec matrice pleines on données de bon résultats.

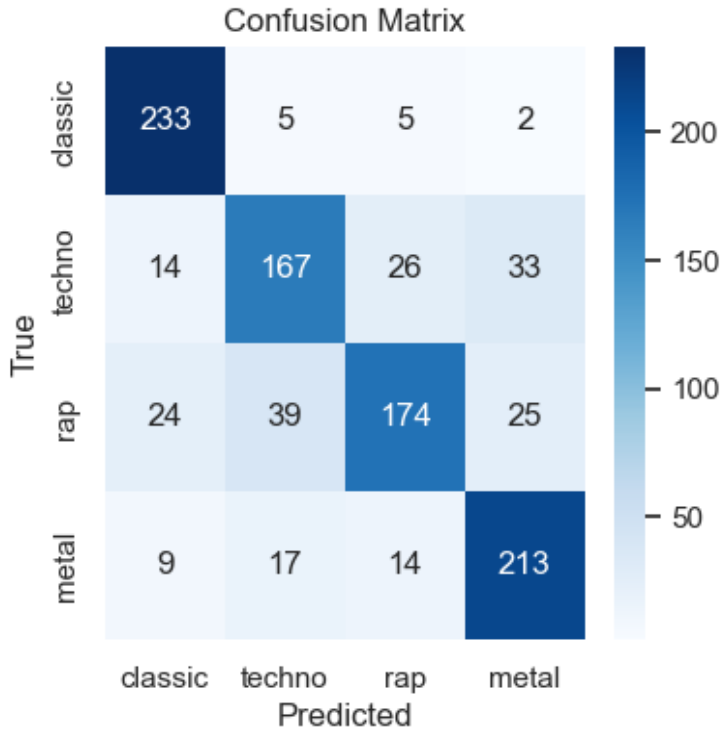
Cependant nous avions la puissance de calcul pour utiliser des matrices pleines pour chaque clusters, c'est donc ce que nous avons utilisé.

Analyse des résultats

En ce qui concerne le CNN, les performances étaient un peu meilleures en utilisant le spectrogramme que la transformé de Fourier (72% pour les FFT contre 78.7% pour le spectrogramme).

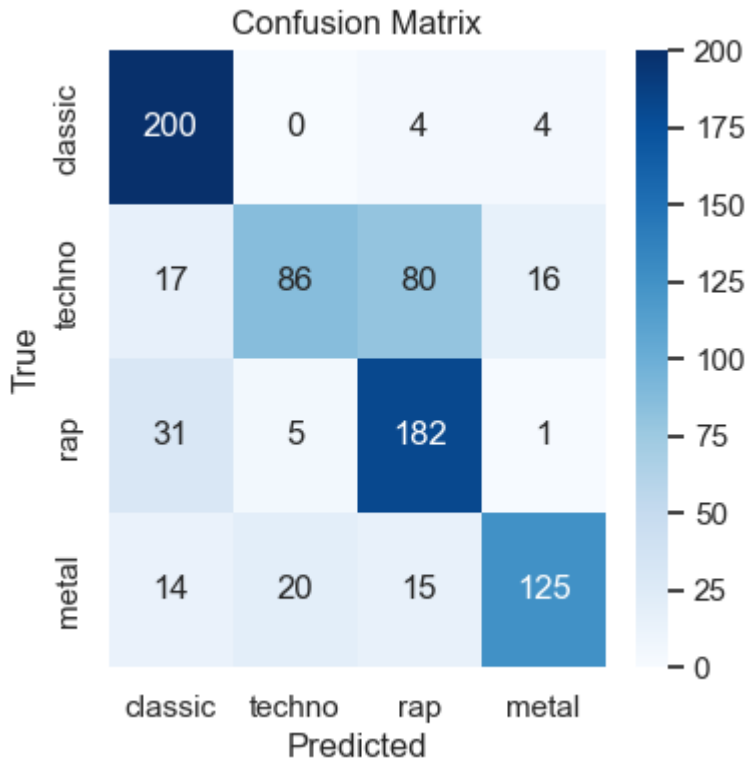


Nous avons également affiché la matrice de confusion de CNN pour voir si un des styles était moins bien reconnu que les autres.



Nous voyons bien que les 4 styles sont reconnus uniformément, notre CNN semble être bien équilibré.

En ce qui concerne le modèle gaussien, nous obtenons une précision de 69.7% et voici la matrice de confusion associé :



Nous remarquons que les résultats sont corrects bien que le modèle peine à repérer les morceaux de techno que le modèle confond avec des morceaux de rap.

Conclusion

Nous sommes contents du travail réalisé ainsi que des résultats obtenus. Le CNN à particulièrement bien fonctionné, nous pensons qu'il pourrait être encore plus performant en ayant des couches convolutives supplémentaires mais nous n'avions ni le temps, ni les ressource pour tester ceci.

Le modèle gaussien multivarié à légèrement moins bien fonctionner mais son entraînement reste plus rapide. Cependant pour des dimensionnalités aussi grandes les résultats sont moins stables et les données d'entrée doivent être en plus grand nombre, nous en concluons donc que ce type de modèle est moins adapté à ce genre de tâche.