Hector PITEAU **Groupe 4**
Lilian HERTEL

Projet de Modélisations mathématiques

I. Introduction

Notre projet consiste à prédire le type d'un instrument de musique parmi les types suivantes : [bass, brass, flute, guitar, keyboard, mallet, organ, reed, string, synth, vocal]. Nos sons provenant de fichiers wave. Il va donc s'agir de réaliser une classification supervisé, en utilisant du "model-based learning", c'est à dire, un apprentissage dans lequel on construit un modèle qui peut être exprimé de façon mathématique comme un ensemble d'hypothèses de relations entre les différentes variables que l'on lui donne en entrée. Et supervisé car tous nos sons sont déjà labellisés, c'est à dire que l'on connaît déjà le type d'instrument d'un son.

A. Seuil de satisfaction :

Puisque nous devons différencier 11 classes d'instruments différents, si le modèle trouve plus d'une fois sur 11 fois, alors ce ne sera plus du hasard. Soit s'il arrive à une précision de plus de 1/11 = 0.0909 = 9.09% . Si nous dépassons les 9.09% de précision, notre modèle sera en mesure de prédire avec une certaine fiabilité le type d'instrument d'un fichier wave.

II. L'origine des données

Nos données proviennent d'un Dataset nommé "NSynth Dataset" du site de tensorflow.org. Ce dataset contient en réalité 3 sets, un set de 289,205 sons destiné à l'entraînement, 12678 destiné à valider les modèles, et 4096 destinés au test. Il faut savoir que ces trois sets contiennent des sons tous différents, il n'y en a pas deux identiques. Le set "Train" étant déjà très bien fournis, nous avons décidé de travailler seulement avec lui. Nous aurons donc déjà 289,205 sons à notre disposition.

L'ensemble contient aussi un unique json contenant un ensemble d'informations sur chaque fichier wave. Tel que pour chaque fichier on connaît :

- **note** [int64] A unique integer identifier for the note.
- **note_str** [bytes] A unique string identifier for the note in the format <instrument_str>-<pitch>-<velocity>.
- **instrument** [int64] A unique, sequential identifier for the instrument the note was synthesized from.
- **instrument_str** [bytes] A unique string identifier for the instrument this note was synthesized from in the format
- **pitch** [int64] The 0-based MIDI pitch in the range [0, 127].
- **velocity** [int64] The 0-based MIDI velocity in the range [0, 127].
- **sample_rate** [int64] The samples per second for the audio feature.
- **audio*** [float] A list of audio samples represented as floating point values in the range [-1,1].
- **qualities** [int64] A binary vector representing which sonic qualities are present in this note.

- **qualities_str** [bytes] A list IDs of which qualities are present in this note selected from the sonic qualities list.
- **instrument_family** [int64] The index of the instrument family this instrument is a member of.
- **instrument_family_str** [bytes] The ID of the instrument family this instrument is a member of.
- **instrument_source** [int64] The index of the sonic source for this instrument.
- **instrument_source_str** [bytes] The ID of the sonic source for this instrument.

III. Exploration des données

Une étape importante de ce projet est l'exploration de données. C'est à dire découvrir les différentes informations que nous disposons à propos de ces sons. Le site du DataSet nous propose un fichier JSON, dans lequel est référencé tous les sons dans le DataSet et auxquels il attache différentes informations supplémentaires que vous pouvez voir. Comme par exemple [...] Cependant nous souhaitons prédire un instruments seulement depuis le son lui même, nous n'allons donc pas utiliser toutes ces informations.

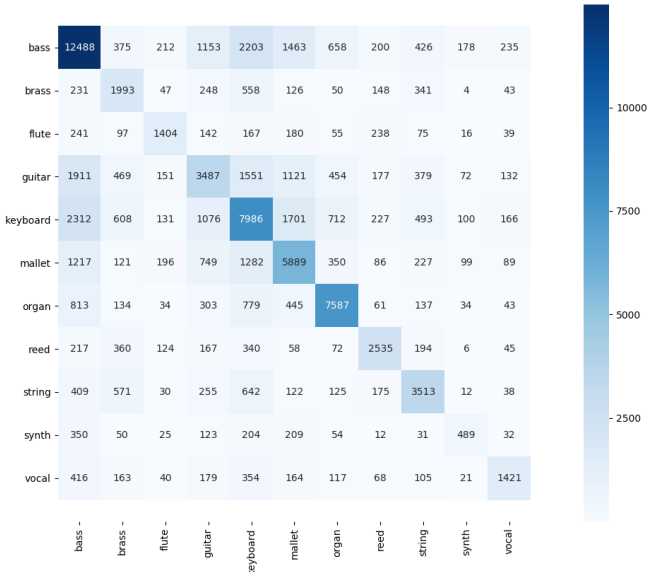
Les fichiers sont en format wave, ils durent 4 secondes, et ont un échantillonnage de 16 000 Hz. Nous ne pouvons pas traiter cette immense quantité de données, c'est à dire 16000*4=64000 floats. Il est impossible de considérer traiter nos 280000 fichiers entièrement, il nous faut donc trouver un moyen d'extraire un petit ensemble d'informations décrivant très précisément le son afin de pouvoir par la suite les classer de la façon la plus précise possible à notre échelle.

a. Exploration avec Transformée de Fourier et fréquences max.

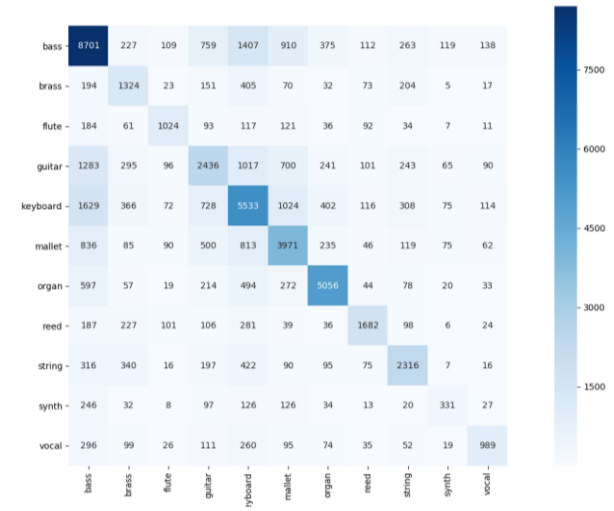
Dans un premier temps, nous avons décidé d'extraire les fréquences les plus importantes dans les sons. Nous en sélectionnons 10 afin de décrire le son. Pour ce faire nous réalisons une transformée de fourier sur chaque fichier en utilisant la fonction issue de **scipy**. Cette transformée de fourier nous permet d'accéder à la représentation du son sous la forme de quantité de fréquence comme nous pouvons le voir sur la **figure 1** **[AJOUTER UNE IMAGE D'UNE TRANSFORMÉE DE FOURIER]**. Comme nous pouvons le voir, certaines fréquences sont dominantes et ressortent beaucoup, ce sont les composantes principales. A cela s'accompagne un très grand nombre de bruits qui à la fois constituent le son, mais qui peuvent aussi être des fréquences parasites.

i. Premiers essaies :

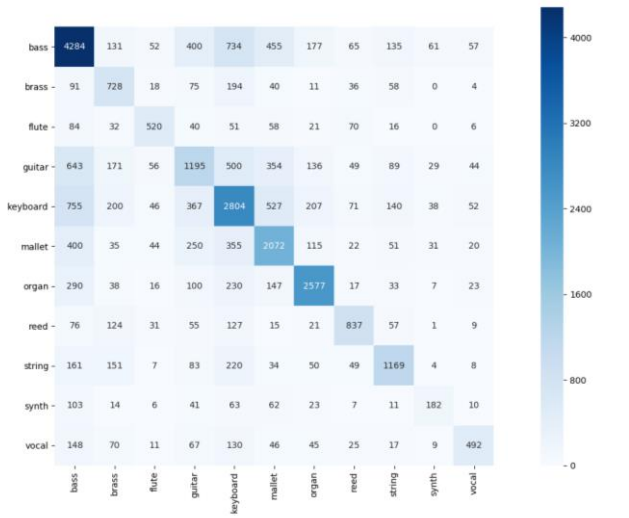
Nous avons précédemment extrait les **10 composantes** principales de chaques son que nous avons stocké dans un fichier JSON. Chaque ensemble de **10 valeurs** est bien évidemment accompagné de son label (décrivant la classe d'instrument). Nous séparons nos données en deux groupes distincts. Un groupe d'entraînement constitué de **202443 sons**, qui sera utilisé pour entraîner le modèle. Et un groupe de **86762 sons**, utilisé pour tester notre modèle et ainsi évaluer sa précision. Soit une répartition de **70%** des sons pour entraîner notre modèle.



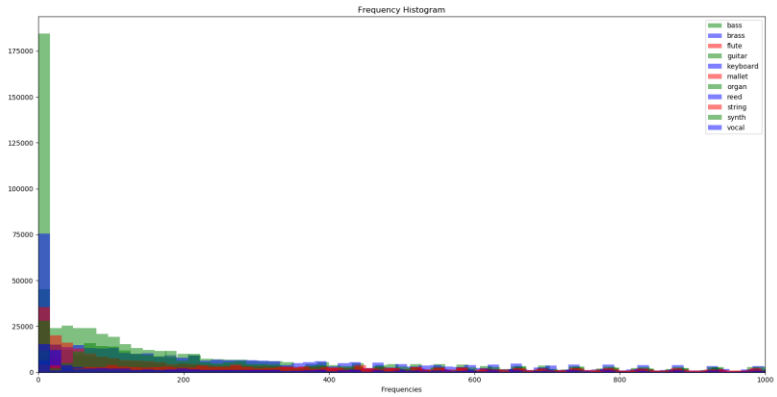
Dans un premier temps nous testons nos données avec un DecisionTreeClassifier, qui est un modèle qui utilise un arbre de décision pour classer les données et faire par la suite des prédictions. En utilisant 70% des sons pour l'entraînement nous obtenons sur l'échantillon de test, **56%** de positifs.
Ce qui est assez satisfaisant, car au moins une fois sur deux le modèle prédit de façon juste la classe d'instrument.



Nous réessayons avec cette fois ci avec 80% des données pour l'entraînement et 20% pour le test soit 231364 de train et 57841 de test. Nous obtenons **57,68%** de bonnes réponses sur l'échantillon de test comme nous pouvons le voir sur la **Figure 2**.



Nous réessayons de nouveau avec 90% des données pour le train et 10% pour le test, soit 260284 sons pour l'entraînement et 28921 pour le test, et nous obtenons **58,29%** de positifs pour l'échantillon de test comme vu sur la **Figure 3**.



Dans un premier temps, nous affichons un histogramme qui nous permet de visualiser l'occurrence des fréquences dans nos classes d'instruments, voir Figure x.

Comme nous pouvons le voir, il y a beaucoup de valeurs proche de zéro. Cela signifie que notre algorithme extrait beaucoup de pics de fréquences élevés autour de 0, ce qui est plus qu'étrange. Pour vérifier cela, nous allons compter le nombre de valeur égale à zéro pour chaque classe d'instruments, voir Figure x.

Classe d'instrument :	**Nombre de 0 :**	**Nombre total de valeurs :**	**pourcentage de 0 :**
bass	56084	654740	8,57
brass	1548	126750	1,22
flute	4866	87730	5,55
guitar	9624	326900	2,94
keyboard	21117	518210	4,07
mallet	3495	342010	1,02
organ	10143	344770	2,94
reed	2311	139110	1,66
string	46	194740	0,02

synth	2879	55010	5,23
vocal	7858	102080	7,70

Comme nous pouvons le voir, la part de zéro prends une place très importante. Nous avons donc plusieurs solutions, soit nous retravaillons l'algorithme afin qu'il trouve des pics de fréquences seulement au dessus de 0. Soit nous remplaçons les 0 par une valeur moyenne. Soit nous supprimons les valeurs 0.

1.1.1. Suppression des sons avec des fréquences à 0 :

Dans un premier temps, nous supprimons les sons qui possède au moins une fréquence max à zéro, et nous relançons une simulation : Résultat sur les tests : **54,41%**

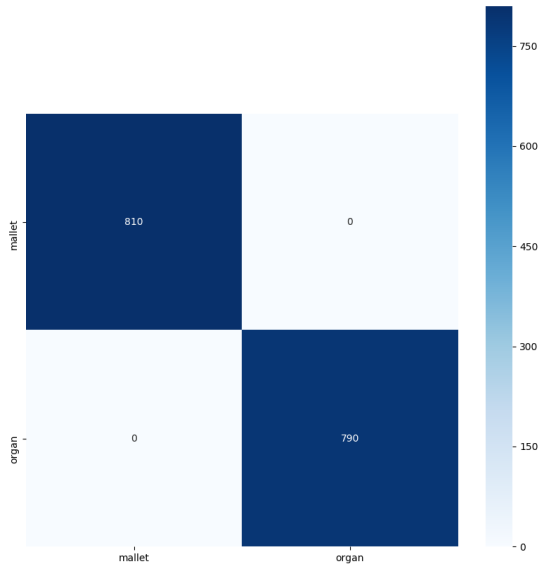
2. Améliorations :

Comme nous avons pu le voir le livre "Hands on Machine learning with Scikit Learn [...]", les algorithmes ont du mal si les valeurs utilisés à l'entraînement sont très étendues. Dans notre cas, il est vrai que certains pics de fréquences seront autour des **200Hz** alors que d'autres iront jusqu'à **7995Hz** d'après max()

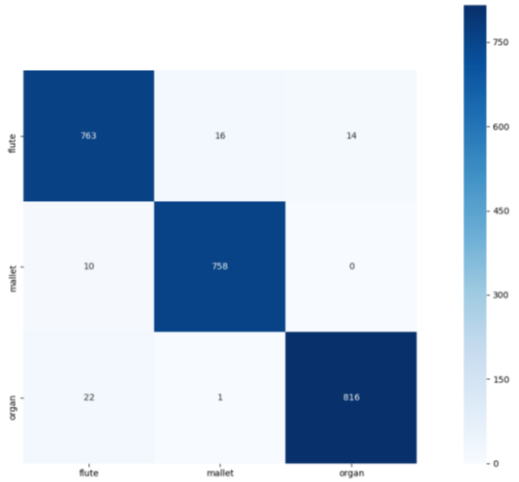
b. Exploration avec des Spectrogrammes

i. Exploration initiale

1. Decision Tree



Avec deux instruments, "Organ" et "Mallet", où nous prenons 4000 samples chacuns, avec une répartition de 80/20 pour les le set de train/test. Nous obtenons une précision sur les test de 100%, comme nous pouvons le voir sur la Figure X ci contre.

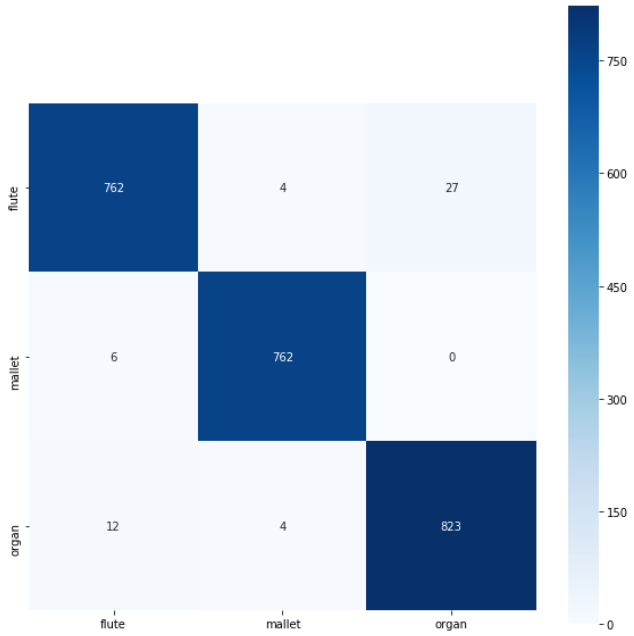


Avec trois instruments, soit ‘flute’, ’organ’, ’mallet’, et 4000 samples de chacun, et la même répartition nous obtenons une précision sur le set de test de 97.4%.

ii. Ajout de la normalisation des spectrogrammes

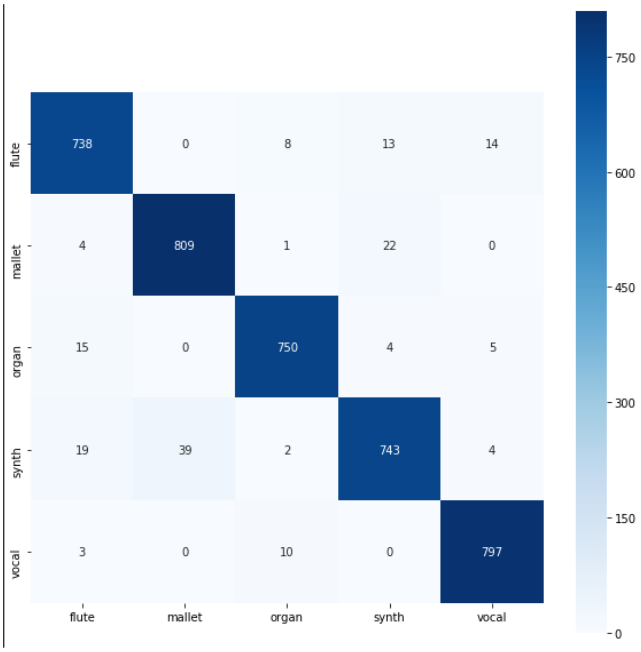
1. Decision Tree

- Avec trois instruments 4000/i (lilian) :



Instruments : flute, mallet, organ
Précision obtenue : 0.97875
Matrice de confusion : ci-contre

- Avec cinq instruments 4000/i (lilian) :



Instruments : flute, mallet, organ, synth, vocal
Précision obtenue : 0.95925
Matrice de confusion : ci-contre

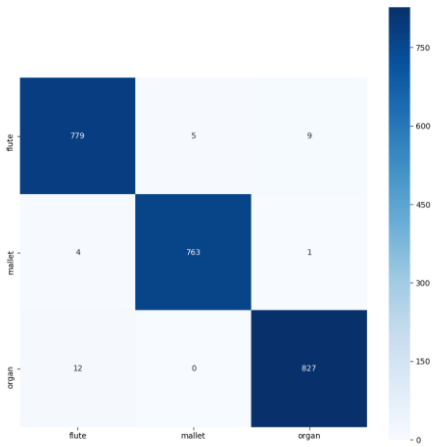
iii. Ajout de la réduction de la résolution des spectrogrammes

Pourquoi réduire la résolution ?

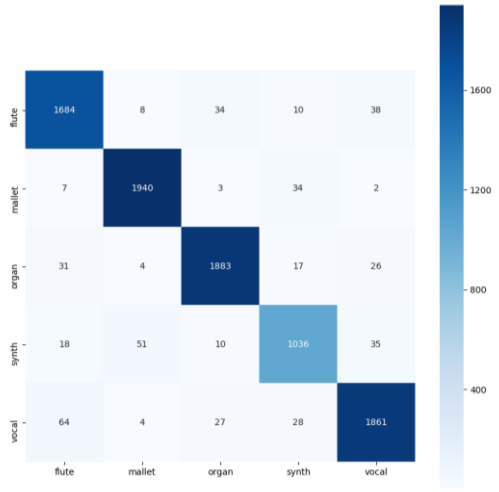
Dans un premier temps, nous réalisons un spectrogramme sur notre son, et nous en tirons un tableau à deux dimensions qui représente l'intensité d'une fréquence précise à un moment précis. Seulement ce tableau de deux dimensions a une taille de **129*285** ce qui nous donne **36765 valeurs**. Donc un son possède 36765 pour le décrire, ce qui d'après nous est trop (entraine un temps de calcul trop long selon nous), les algorithmes prennent du temps, et on voit mal réaliser cette expérience sur plus de **3 instruments** sans y passer plusieurs jours de calcul. Nous avons donc décidé de diviser la taille de ce tableau par 3. La nouvelle dimension est **43*95** ce qui nous donne **4085 valeurs,** soit une quantité de valeurs divisée par **9**, et donc une réduction de presque **90% (88.9%)** de la quantité de données.

Comment fonctionne la réduction

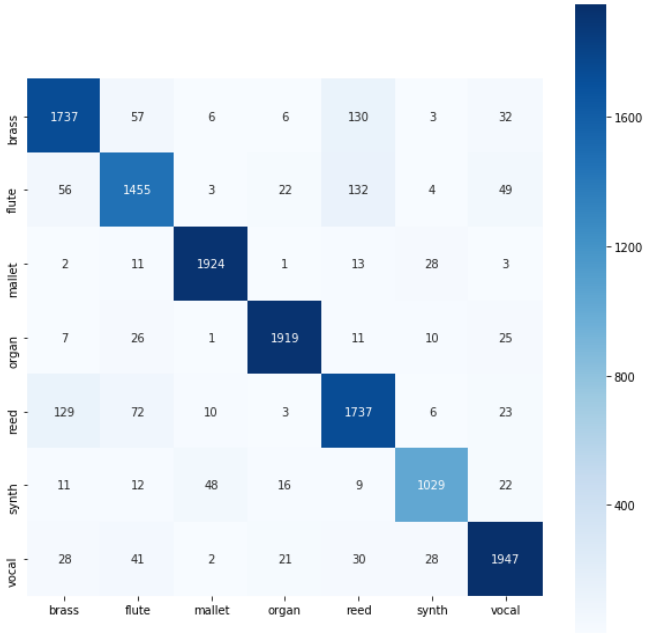
1. Decision Tree



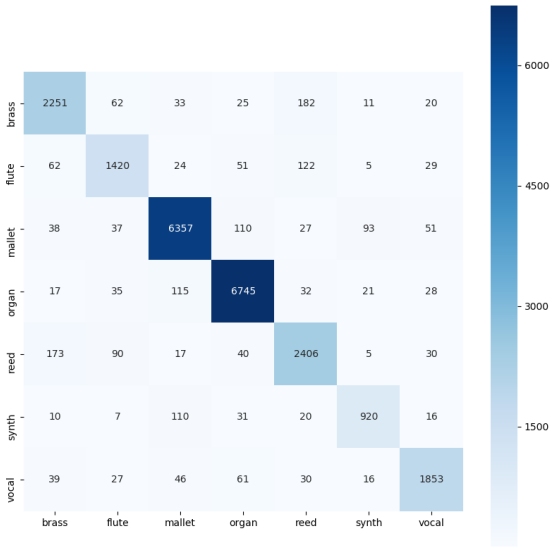
Avec trois instruments et 4.000 samples par instruments, nous obtenons sur le set de test, une précision de **98.7%.**



Avec cinq instruments et un maximum de 10.000 samples par instruments, nous obtenons sur le set de tests, un précision de **94.9%.**



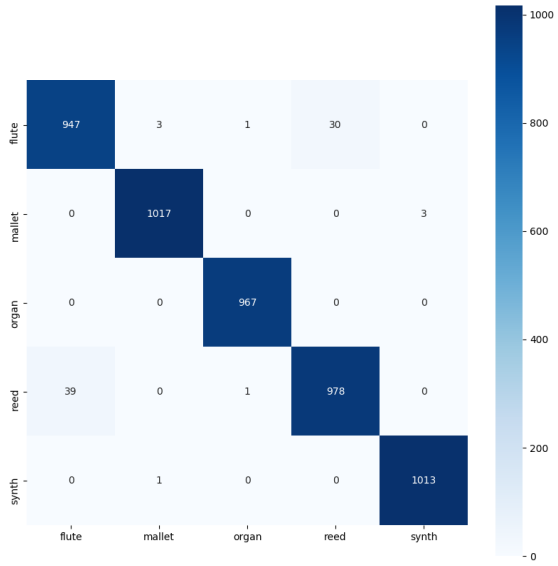
Avec sept instruments, ‘brass’, ‘flute’, ‘mallet’, ‘organ’, ‘reed’, ‘synth’, ‘vocal’, entraînés sur 10000 échantillons pour chaque instrument, nous avons une précision sur le set de test de **91.09%.**



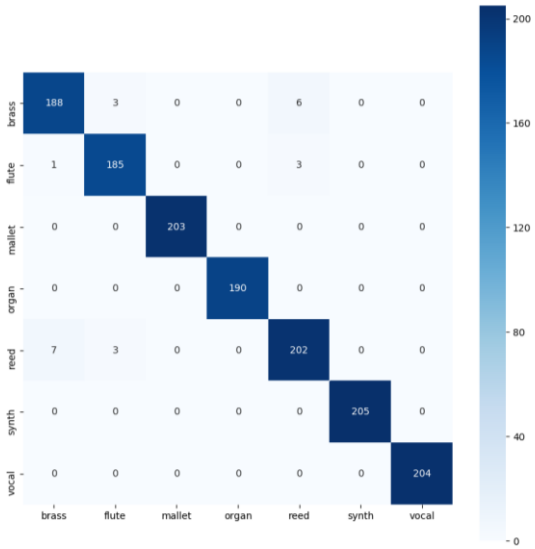
Avec sept instruments, ‘brass’, ‘flute’, ‘mallet’, ‘organ’, ‘reed’, ‘synth’, ‘vocal’, entraîner sur l’entièreté des sons disponibles pour chaques instruments, nous avons une précision sur le set de test de **91.66%.**

Avec tous les instruments et tous les échantillons disponibles pour chaque instrument, on obtient une précision de : **82.24%**

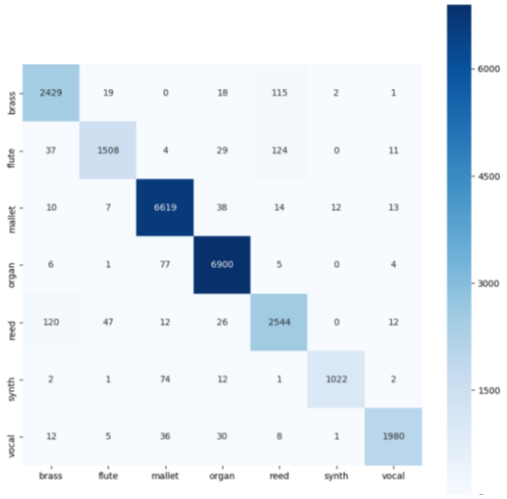
2. Random Forest



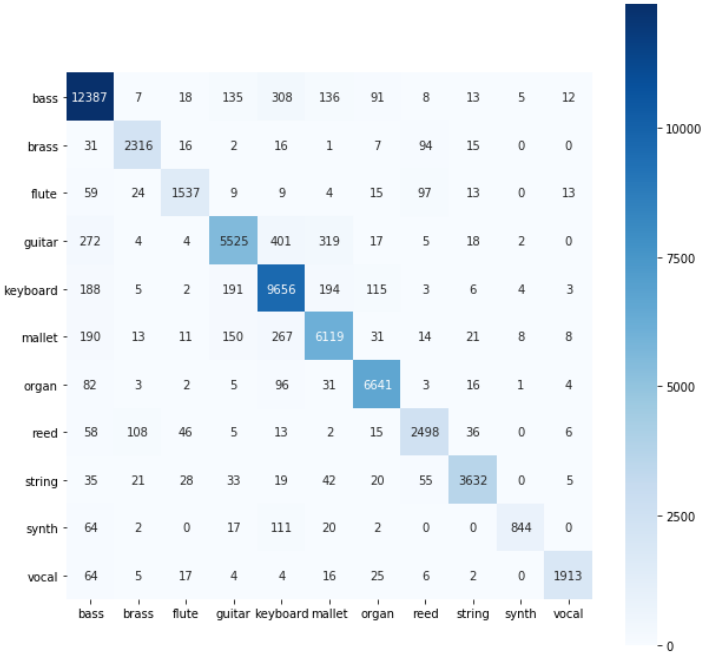
Avec cinq instruments, et 5.000 samples par instruments, en utilisant le même algorithme qu'avant, mais cette fois ci en utilisant une RandomForest, nous obtenons une précision de **98.44%**



Avec sept instruments et 10.000 samples, nous obtenons **98.36%** sur le set de test.



Avec sept instruments et tous les sons que l'on dispose dessus, à l'aide de la RandomForest, nous obtenons sur l'échantillon de test, **96.04%** de précision.



Avec tous les instruments intégral (TRUE FINAL BOSS PHASE_2) (lilian)

Précision: **91.75%**