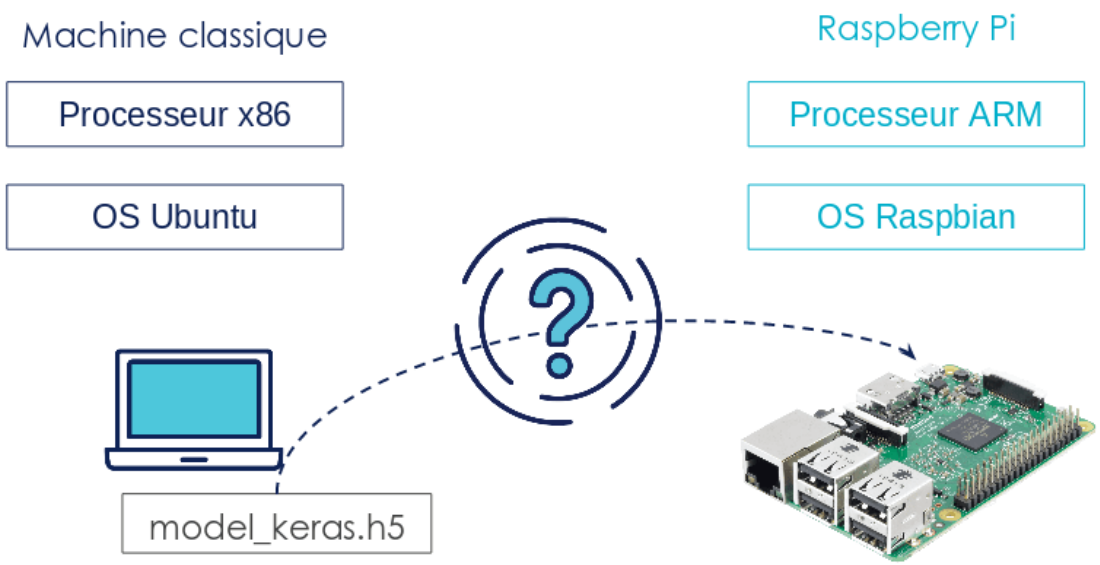
RAPPORT PROJET

EMBEDDED MACHINE LEARNING



Auteurs :

Mirado RAJAOMAROSATA (FISE23)
Nicolas DEFOUR (FISE23)

Le lien de notre git : `https://github.com/29Nicolas/ml_embedded.git`

31 janvier 2023

Introduction

Le machine learning est un domaine en constante évolution qui permet de rendre les systèmes embarqués plus intelligents et plus autonomes. Il permet de traiter des données en temps réel pour effectuer des tâches telles que la reconnaissance vocale, la reconnaissance d'images et la planification de parcours. En utilisant des algorithmes de traitement de données avancés, les systèmes embarqués peuvent prendre des décisions plus efficaces et améliorer leur performance.

Un des principaux défis liés à l'utilisation du machine learning sur des systèmes embarqués est le coût élevé de l'apprentissage des modèles d'IA. Les systèmes embarqués ont généralement des capacités de calcul limitées et des ressources mémoire restreintes, ce qui rend difficile l'apprentissage de modèles d'IA qui nécessite beaucoup de calcul et de contenir en mémoire l'ensemble des données.

Une solution couramment utilisée pour contourner les défis liés au coût de l'apprentissage des modèles d'IA sur des systèmes embarqués est de les apprendre sur un ordinateur de bureau ou un serveur avec des ressources de calcul plus importantes, puis de transférer le modèle appris sur la plateforme d'inférence, c'est-à-dire sur le système embarqué. Ce processus est connu sous le nom de "apprentissage en dehors de la boucle" ou "transfer learning". Cela permet d'optimiser l'utilisation des ressources en effectuant l'apprentissage sur un système de puissance de calcul plus élevée, tout en conservant les avantages du machine learning sur le système embarqué.

Notre projet vise à développer un système de reconnaissance de genre musical pour des systèmes embarqués. L'objectif principal est de créer un modèle de machine learning qui puisse prédire le genre musical d'un fichier audio, en fonctionnant efficacement sur des systèmes embarqués avec des ressources limitées. Pour atteindre cet objectif, nous allons utiliser le transfer learning pour entraîner le modèle sur nos pc avec des ressources de calcul plus importantes, avant de le transférer sur un système embarqué pour l'inférence.

Pour ce projet, nous allons coder en Python pour l'apprentissage des modèles car c'est un langage couramment utilisé en machine learning en raison de la disponibilité de bibliothèques dédiées telles que TensorFlow et scikit-learn, qui facilitent la création et l'entraînement de modèles. Cependant, pour l'inférence sur des systèmes embarqués, le C++ est souvent préféré en raison de sa performance plus élevée et de sa capacité à gérer des calculs en temps réel.

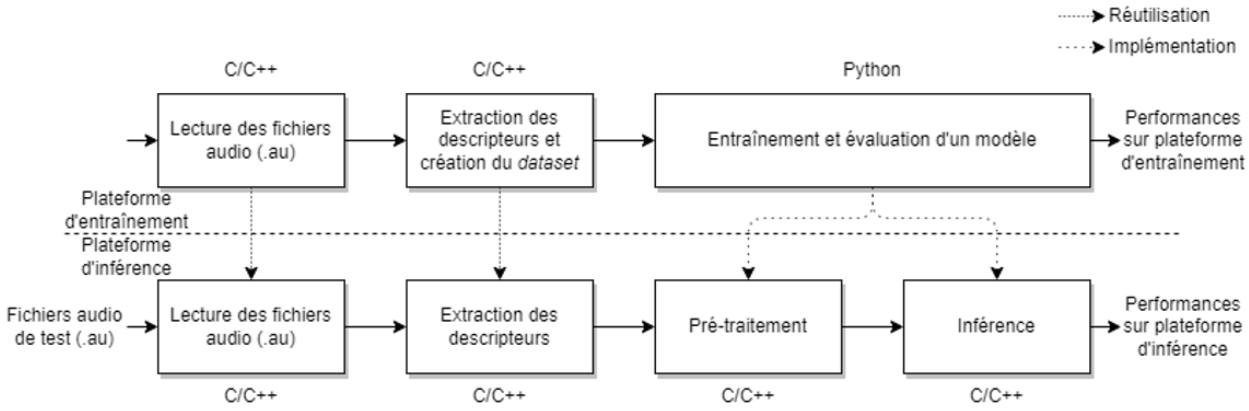


Figure 1 – Processus de développement

1 Plateforme d'apprentissage

L'apprentissage du modèle de reconnaissance de genre musical est conçu des étapes suivantes :
- Lire chaque fichier audio de l'ensemble des données
- Extraire les descripteurs en utilisant des techniques d'analyse de signal (en C++ pour pouvoir l'uiliser sur la plateforme d'inférence)
- Entraîner les modèles capables de prédire le genre musical d'un fichier audio à partir des descripteurs extraits
- Evaluer les modèles sur un ensemble de test
- Enregistrer l'ensemble de test pour évaluer les performances des deux plateformes sur le même ensemble
- Enregistrer les modèles entraînés pour pouvoir le transférer sur la plateforme d'inférence

1.1 Lecture et extraction des descripteurs de l'ensemble de données

L'ensemble des données est contenu dans le dossier 'archive' qui nous est fourni. Dans ce dossier, les fichiers audio sont classé dans des repertoires en fonction de leur genre musical. Le code doit trouver l'ensemble des données et enregistrer le chemin de chaque fichier audio dans une liste.
Les données comportent 10 genres musicaux différents et pour chacun, 100 fichiers audio. Le nom des fichies audio contient le nom du genre musical suivi d'un numéro c'est-à-dire le nom de chaque fichier est de la forme "<nom_genre>.XXXXX.au".

L'extension '.au' nous informe sur la structure des informations du fichier (cf Figure 2).

Mot 32 bits	Champ	Contenu pour GTZAN
0	Nombre magique	0x2e736e64 (779316836)
1	Décalage des données	-
2	Taille des données	~ 1 323 000 octets
3	Encodage	3 (16 bits linear PCM)
4	Fréquence d'échantillonnage	22050 Hz
5	Nombre de canaux	1 (monocanal)

FIGURE 2 – Entête d'un fichier .au (GTZAN)

La lecture du fichier doit débuter par la lecture du nombre magique. On peut vérifier l'intégrité du fichier car le nombre magique doit être : $0x2e736e64$. La lecture se poursuit avec des paramètres utiles notamment le décalage à réaliser pour pointer vers les données. Après avoir décaler le curseur sur le début des données, on peut lire les données audios. On lit 'L' bits de donnée ou si le fichier audio est trop court, on s'arrête à la fin du fichier audio et on remplit par des 0 pour obtenir des données de même taille.

On peut maintenant passer à l'extraction des descripteurs du fichier audio lu. Les descripteurs audio sont la moyenne temporelle du spectrogramme μ et l'écart-type temporel du spectrogramme σ de la transformée de fourrier rapide (fft).

Les données retenues du fichier audio ont une longueur L. On les découpe en K éléments de longueur N. On va lire chaque élement de longueur N le un après les autres. Pour chacun de ces éléments, nous allons calculer la transformée de fourrier rapide. On calculera μ et σ de manière itératif sur chaque élément de longueur N sans retenir les fft de chaque élément pour ne pas sur-charger la

mémoire.

Comme on utilise une méthode itératif, on commence par l'initialisation c'est-à-dire les deux premiers éléments du fichier audio :

$$\forall i < 2, \mu = |fft(i)|$$

$$\forall i < 2, \sigma = 0 \text{ et } s = |fft(i)|$$

Ensuite, on calcule μ à chaque itération i :

$$\mu = \frac{i-1}{i} \times \mu + \frac{1}{i} \times |(fft(i))|$$

et σ à l'itération i, $\forall i \geq 2$:

$$s = s + |fft(i)|$$

$$\sigma = \frac{1}{N} \times \left(\sigma + \frac{\left(i \times |fft(i)| - s \right)^2}{i \times (i-1)} \right)$$

Ainsi, on obtient la moyenne temporelle du spectrogramme et l'écart-type du fichier audio en limitant le besoin de mémoire. Il nous suffit d'enregistrer μ et σ dans un fichier csv tel que sur la ligne on retrouve μ, σ, label avec le label obtenu dans le nom du fichier audio.

1.2 Entrainement des modèles

On commence par lire en python le fichier dataset.csv contenant l'ensemble des descripteurs de chaque individus. On sépare cet ensemble d'individu en deux ensembles :

- un ensemble d'entrainement correspondant au 2/3 de l'ensemble de donnée, qui permettra d'entrainer le modèle
- un ensemble de test contenant le 1/3 des données restantes pour évaluer les performances du modèle

1.2.1 Modèle SVM, Decision Tree et Random Forest

La création et l'entrainement des modèles SVM, decision tree et random forest est similaire. On commence par d'utiliser une pipeline de sklearn car c'est un outil pratique pour automatiser les étapes de prétraitement des données, de sélection de modèle et d'évaluation de performance. Elle permet de regrouper toutes les étapes de l'analyse en un seul objet facile à utiliser. En utilisant une pipeline, on peut facilement intégrer des étapes de normalisation des données (preprocessing.StandardScaler()) et créer le modèle du processus d'entraînement (linearSVC(), DecisionTreeClassifier() ou RandomForestClassifier(n_estimators=100)). De plus, la pipeline de sklearn permet également de facilement évaluer les performances du modèle en utilisant des techniques de validation croisée, ce qui est crucial pour évaluer la qualité du modèle et s'assurer qu'il est capable de généraliser ses prédictions à des données non vues lors de l'entraînement.

Pour tout les modèles créés ici, nous avons créé un pipeline contenant la normalisation des descripteurs. Cette normalisation permet de réduire la complexité du modèle ainsi que de donner la même importance à tout les descripteurs. Pour récupérer la normalisation de chaque modèle, on enregistre, dans des fichiers csv, les vecteurs 'mean_' et 'scale_' de l'objet du modèle pipeline['standardscaler]. On pourra ensuite normaliser les descripteurs de la manières suivante : $descripteur\_normalise = (descripteurs - mean\_)/scale\_$.

1.2.1.1 Enregistrement du modèle SVM :

Pour enregistrer le modèle SVM, il suffit de récupérer les matrices coef_ et intercept_ qui permet de prédire la classe à partir des descripteurs par $proba\_classe = coef\_ \times descripteurs - intercept\_$. On enregistre donc dans un fichier csv (à la suite de la normalisation) les matrices 'coef_' et 'intercept_'.

1.2.1.2 Enregistrement du modèle Decision Tree :

Le modèle Decision Tree est un arbre. Nous avons donc créé une fonction récursive pour descendre dans l'arbre en partant du node 0 jusqu'au feuille qui permettent de prédire une classe. Ainsi, à chaque branche de l'arbre, on visite ensuite les deux noeuds voisins en appelant la fonction de manière récursif. On va donc de noeud en noeud jusqu'à que le noeud n'ait pas d'enfant (condition d'arrêt). On enregistre ensuite dans un fichier cpp l'algorithme en c++ avec les conditions pour arriver à cette feuille. L'algorithme s'écrit donc :

```
lecture node(i)
    si node enfant droite = noeud enfant gauche
        return: "return " + str(np.argmax(value[i])) + ";"
    sinon
        txt1 = fonction node(children_left[i])
        txt2 = fonction node(children_left[i])
        return: "if("+ condition sur mu ou sigma + "){" + txt1 + "}" + "else{" +
   txt2 + "}"
```

1.2.1.3 Enregistrement du modèle Random Forest :

Une forêt est composée de plusieurs arbres, ici 100. On doit donc appliquer la même fonction récursive que pour Decision Tree sur les 100 arbres. Il faut ensuite prendre la classe qui sera en majorité prédite par les 100 arbres. On a donc réaliser un code python générant les 100 fonctions récursives. Et on a ajouté une fonction permettant de faire les prédictions sur les 100 arbres pour un individu dont on passe ses descripteurs en argument et cette fonction récupérer la classe qui est majoritairement prédite. Pour prédire, il suffit donc d'appeler la fonction randomForest en lui donnant en argument les descripteurs d'un individu. Le modèle est donc enregistrer par la génération d'un code cpp.

1.2.2 Réseau de neurone

Pour le réseau de neurone, on commence encore par la lecure des descripteurs et par split en 2 ensembles. Ensuite, on doit codé les labels en indice c'est-à-dire qu'on identifie la classe par un entier de 0 à 9. On poursuit en normalisant les deux ensembles pour réduire la complexité du réseau et donner la même importance à tout les descripteurs.

On utilise la librairie tensorflow pour créer un réseau de neurone convolutif. On fait le choix de choisir un réseau de neurone simple composé d'aucune couche cachée. On ajoute une première couche dense (fully connected) en entrée du réseau à laquelle on doit alors spécifier la taille des descripteurs d'entrée du réseau ici 1024. Cette couche comprend 256 neurones et une activation relu. L'activation relu permet au modèle d'apprendre des fonctionnalités plus complexes et de s'adapter à des données non-linéaires. La couche de sortie est une couche dense de 10 neurones et l'activation softmax. On met 10 neurones sur cette couche pour avoir autant de neurone que de classe à prédire. L'activation softmax pour transformée les sorties des neurones en probabilités d'appartenir à la classe représentée par le neurone.

Le réseau est créé. On passe donc à la phase d'apprentissage. On choisit de compiler le réseau en prenant la fonction de coût SparseCategoricalCrossentropy car les labels sont des entiers et qu'on utilise une couche de sortie dense softmax pour une prédiction multi-classe. On a choisi l'optimizer ADAM qui utilise des méthodes d'estimation de moments pour mettre à jour les poids du modèle en utilisant les informations de gradient et de la moyenne mobile des gradients passés. Enfin, la metric utilisée est 'accuracy' pour évaluer le modèle. On peut donc entrainer notre modèle en modifiant le nombre d'epoch pour améliorer le résultat. On a obtenu des résultats satisfaisant (60%) avec 40 epochs.

Le réseau de neurone doit être enregistré dans un format tensorflow lite pour pouvoir être transféré sur la plateforme d'inférence. Il suffit de convert le modèle puis de l'enregistrer dans un fichier d'extension .tflite. Les paramèters par défaut ne recherche pas d'optimisation particulière. On a donc un fichier de 1Mo.

On peut chercher à optimiser l'enregistrement. Dans notre cas, on va optimiser l'enregistrement pour que le modèle soit plus léger. Pour cela, il suffit d'ajouter une ligne avant de finir la convertion. 'converter.optimizations = [tf.lite.Optimize.DEFAULT]' La convertion permet de réduire la taille du fichier à 250ko en ne dégradant pas trop les performances de prédiction. En effet, cette option permet de simplement réduire le nombre de bits sur lequel sont codé les nombres (32bits en normal contre 8bits par l'optimisation). Le modèle optimisé n'est pas compatible avec l'image de la raspberry Pi fourni. On ne peut donc pas vérifier les performances du modèle optimisé.

1.3 Evaluation des modèles sur la plateforme d'apprentissage

Pour chaque modèle, nous avons évalué le modèle sur un ensemble de test. Les commentaires sur les performances des modèles seront faite en même temps que la comparaison des performances entre les deux plateformes disponibles dans la section Performance et complexité des modèles.

1.4 Enregistrement de l'ensemble de test

Pour évaluer les performances sur la plateforme d'inférence et les comparer avec celle de la plateforme d'entraînement, nous devons enregistrer l'ensemble de test. Pour lire facilement l'ensemble en C++, nous avons fait le choix d'écrire les variables X_test, tableau contenant les descripteurs des individus de l'ensemble de test, y_test, les labels des individus codé en indice, et les tailles des tableaux dans un fichier d'entension .h en prenant le soin d'écrire les variables interprétables par le compilateur c++. On pense aussi à écrire dans le fichier les performances de la plateforme d'entraînement pour pouvoir comparer les deux plateformes.

2 Plateforme d'inférence

Nous allons débuter par coder les différentes fonctions qui seront utile pour l'utilisation des modèles sur la plateforme d'inférence.

2.1 Lecture des fichiers audio et extraction des descripteurs

Les fonctions pour lire et extraire les descripteurs ont été réalisé pour la plateforme d'entrainement en c++. On les réutilise donc directement sur la plateforme d'inférence.

2.2 Modèles

2.2.1 SVM

On commence par charger les informations contenues dans le fichier csv enregistré par la plateforme d'entraînement. Le fichier débute par la normalisation (mean_ et scale_) puis par les matrices coef_ et intercept_. Pour prédire à partir du modèle SVM, on doit réaliser des calculs matricielles. On préfère alors utiliser des vecteurs et des matrices d'Eigen pour réaliser des calculs matricielles en c++ de manière optimisée.

Ainsi, après avoir extrait les descripteurs du fichier audio souhaité, on normalise les descripteurs c'est-à-dire : $descripteur = (descripteur - mean\_)/scale\_$ Ensuite, la prédiction est : $proba = coef\_ \times descripteur - intercept\_$. Il suffit de prendre la probabilité la plus grande dans le vecteur proba obtenu pour prédire la classe de l'individu et donc le genre musical du fichier.

Pour évaluer les performances et les comparer avec la plateforme d'entrainement, on réalise la prédiction des individus de l'ensemble de test. On compte le nombre de prédiction juste et on divise par le nombre d'élément dans l'ensemble de test pour obtenir l'accuracy. On peut donc afficher l'accuracy des deux plateformes et comparer.

2.2.2 Decision Tree et Random Forest

La méthode de prédiction est la même pour Decision Tree et random Forest. On commence par lire la normalisation (mean_ et scale_), ici sous forme de tableau. La normalisation se fait ici sur chaque élément du tableau avec une boucle for contrairement au modèle SVM qui utilise des calculs matricielles.

Pour prédire, il suffit d'appeler les fonctions générées par la plateforme d'entrainement en python nommé respectivement decision_tree() et randomForest(). Ces fonctions retournent directement le nom de la classe et son indice. On a donc la prédiction. Pour évaluer le modèle, on réalise un boucle pour prédire les individus de l'ensemble de test. On compare l'indice obtenu par l'indice du label de la classe réelle. On compte le nombre de bonne prédiction et on divise par le nombre totale pour obtenir l'accuracy.

2.2.3 Neuronal Network

Pour le réseau de neurone, un code exemple nous était fourni dans l'image de la carte Raspberry. Il nous a simplement suffit de changer le nom des fichiers pour réaliser l'évaluation du modèle. Le code prédit, à partir du modèle tensorflow lite obtenu par la plateforme d'entrainement, la classe de chaque individu de l'ensemble de test et le compare à sa vraie classe pour obtenir l'accuracy. La prédiction seule n'est pas disponible sur ce modèle. Il suffirait d'appeler au départ les fonctions de lecture et d'extraction des descripteurs puis de faire une unique itération de la boucle de prédiction pour obtenir le genre musical du fichier souhaité.

2.3 Plateforme d'inférence

Une carte raspberry pi 4 va être utilisé comme ordinateur embarqué et être la plateforme d'inférence. La première étape a été de flasher l'image fournie sur la carte SD de la raspberry pi. Ensuite, nous avons autorisé le ssh pour pouvoir l'utiliser sans clavier. De plus, la carte raspberry a été programmé pour se connecter automatiquement sur un réseau d'un routeur avec une adresse IP statique. Ainsi, on utilise la carte Raspberry pi sans écran et sans clavier.

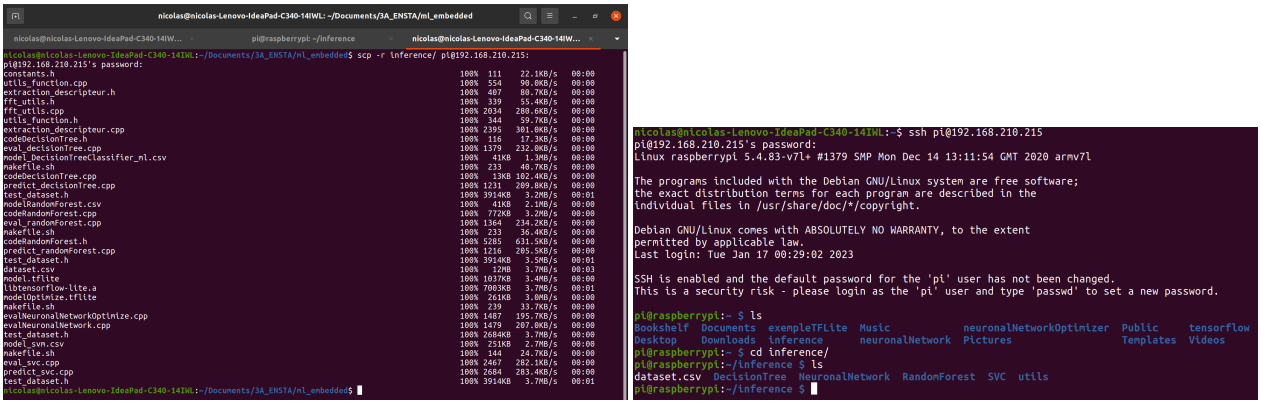
Pour envoyer le répertoire /inference/ contenant tous les codes réalisés pour la plateforme d'inférence, on se place le pc d'apprentissage sur le même réseau que la carte raspberry Pi et on utilise la commande suivante à exéctuer à la racine du répertoire github (Figure 3a) :

```
scp -r inference/ pi@<adresse_ip_RPi>
```

Lors de cette opération, il est demandé de rentrer le code de la Raspberry Pi qui est par défaut : "raspberry".

Après le téléversement des fichier, on peut maintenant se connecter sur la carte Raspberry en utilisant la commande (Figure 3b) :

```
ssh pi@<adresse_ip_RPi>
```



(a) Téléverser le répertoire d'inférence sur la RPi (b) Accès en ssh et vérification de la présence des codes

FIGURE 3 – Plateforme d'inférence à distance

Pour lancer les codes des différents modèle, il suffit de se déplacer dans le répertoire correspondant au modèle voulu puis de lancer makefile.sh qui compile les codes et va créer deux executables, l'un permettant d'évaluer le modèle à partir du dataset_test et l'autre permet de prédire (nécessite le nom du fichier audio placé dans le bon répertoire sans l'extension .au).

3 Performance et complexité des modèles

On peut évaluer les performances de tous les modèles utilisés et vérifier qu'elles sont indentiques sur les deux plateformes. On parlera aussi de la complexité temporelle et de mémoire de différentes méthodes.

3.1 Complexité de la lecture et de l'extraction des descripteurs

Pour extraire les descripteurs du fichier audio, on utilise une transformée de fourrier rapide. Cette méthode a une complexité temporelle de $O(n \times log(n))$ avec n le nombre d'élément d'entrée et une complexité en mémoire en O(n) car le calcul itératif des descripteurs permet d'uniquement conserver en mémoire un vecteur de taille n.

3.2 SVM

Les performances sur les deux plateformes sont les suivantes :

Plateforme d'entrainement :
- Accuracy : 0.4636
- Precision : 0.468
- Recall : 0.4636
- F1 score : 0.467

Plateforme d'inférence :

On constate que les performances de la plateforme d'inférence est proche de la performance de la plateforme d'apprentissage. On constate que l'évaluation dure 0.30s (sans lecture de fichier audio et extraction de descripteurs).

La prédiction du modèle SVM se fait en faisant multipliant une matrice de taille (10,1024) par un vecteur (1024) puis en soustrayant par un vecteur (1024). La complexité temporelle est en O(mn) où m est la dimension de la matrice et n est la dimension du vecteur soit une complexité en O(1010^7).

Le modèle demande de stocker les deux matrices coef_ (10,1024) et intercept_ (1,1024) ainsi que le vecteur de descripteur (1,1024) et le résultat en vecteur de (1,1024). En ne prenant pas en compte la mémoire supplémentaire nécessaire pour stocker les valeurs intermédaires. On doit donc stocker 10*1024+1024+1024 nombres en mémoire. Le produit matricielle a une complexité en mémoire de O(mn) comme la complexité temporelle. Il faut donc beaucoup de mémoire pour réaliser la prédiction.

3.3 Decision Tree

Les performances du modèle Decision Tree obtenue sur les deux plateformes sont les suivantes :

Plateforme d'entrainement :
- Accuracy : 0.4182
- Precision : 0.4187
- Recall : 0.4182
- F1 Score : 0.4159

Plateforme d'inférence :

```
Match: 138
Accuracy on training platform: 0.418182 %
Accuracy on inference platform: 0.418182 %
Prediction time: 0.00977s
pi@raspberrypi:~/inference/DecisionTree $
```

La prédiction se fait par une cascade de condition. Pour déterminer la complexité maximale, il faudrait déterminer le plus grand nombre de condition pour atteindre la feuille la plus loin. Il y a 174 'if" dans le fichier. On a donc une complexité temporelle faible par rapport à celle du modèle SVM. De manière générale, un arbre de décision a une complexité temporelle en O(log(n)) avec n le nombre d'élément dans l'ensemble de donnée soit ici 1024.

La complexité en mémoire d'un arbre de décision est en O(n) avec n=1024 le nombre d'entrée.

L'apprentissage en revanche demande beaucoup plus de mémoire et de temps pour construire l'arbre de décision. Mais ici, on ne s'intéresse pas à l'apprentissage car elle est faite en amont sur un pc performant sans soucis de temps et de mémoire.

3.4 Random Forest

Les performances obtenues sont :

Plateforme d'entrainement :

- Accuracy : 0.5061
- Precision : 0.5187
- Recall : 0.5061
- F1 Score : 0.4951

Plateforme d'inférence :

```
Match: 166
Accuracy on training platform: 0.506061 %
Accuracy on inference platform: 0.50303 %
Prediction time: 0.025478s
pi@raspberrypi:~/inference/RandomForest $
```

Les performances sont meilleurs que les modèles précédent.

La complexité temporelle est de O(n * m * log(n)) où n est le nombre d'éléments dans l'ensemble de données et m est le nombre d'arbres dans la forêt. Cela est dû au fait que chaque arbre est construit indépendamment des autres en utilisant une sous-partie aléatoire de l'ensemble de données, ce qui entraîne une complexité temporelle supplémentaire liée à la construction de chaque arbre. Dans notre projet, on a mit 100 arbres avec 1024 entrée. On a donc une complexité de $O(100 \times 1024 \times log(1024))$

La complexité en mémoire est de O(n * m) où n est le nombre d'éléments dans l'ensemble de données 1024 et m est le nombre d'arbres dans la forêt soit 100 ici. Cela est dû au fait qu'il faut stocker toutes les informations relatives à chaque arbre dans la forêt, y compris les informations de découpage, les informations de prédiction et les pointeurs vers les fils.

3.5 Neuronal Network

Les performances obtenues sont :

Classification Report

	precision	recall	f1-score	support
blues	0.64	0.53	0.58	30
classical	0.58	0.93	0.72	30
country	0.68	0.46	0.55	37
disco	0.64	0.60	0.62	42
hiphop	0.71	0.69	0.70	36
jazz	0.73	0.37	0.49	30
metal	0.65	0.81	0.72	32
pop	0.81	0.91	0.85	32
reggae	0.65	0.50	0.57	34
rock	0.27	0.41	0.32	27
accuracy			0.62	330
macro avg	0.64	0.62	0.61	330
weighted avg	0.64	0.62	0.62	330

(a) Plateforme d'entrainement

```
pi@raspberrypi:~/inference/NeuronalNetwork $ ./output
Match: 199
Accuracy on training platform: 0.621212 %
Accuracy on inference platform: 0.60303 %
Prediction time: 0.147734s
pi@raspberrypi:~/inference/NeuronalNetwork $
```

(b) Plateforme d'inférence

FIGURE 4 – Performance du réseau de neurone

Les performances sont bien meilleurs que tous les autres modèles.

La complexité temporelle est de O(n * m * k) où n est le nombre d'éléments dans l'ensemble de données d'entraînement (1000, 1024), m est le nombre de couches dans le réseau de neurones (264,970 neuronnes) et k est le nombre de neurones dans chaque couche (2 couches seulement).

La complexité en mémoire est la même que temporelle. C'est pourquoi nous avons essayé d'optimiser l'enregistrement du réseau sous tensorflow lite pour réduire la complexité en mémoire en

codant les nombres sur moins de bits. En effet, le modèle non optimisé fait 1Mo alors que celui optimisé fait plus que 250Ko.

4 Conclusion

Ce projet consiste à utiliser des techniques d'apprentissage pour développer un modèle de reconnaissance de genre musical qui pourra être utilisé sur une plateforme embarquée. Nous avons commencé par l'extraction des descripteurs des fichiers audio à l'aide de techniques d'analyse de signal en C++. Ensuite, nous avons utilisé des techniques de machine learning pour entraîner des modèles de classification multi-classe en utilisant les données d'entraînement. Les modèles ont été évaluer sur un ensemble de test et les résultats seront comparés entre les deux plateformes. Enfin, nous avons utilisé la plateforme d'inférence pour faire des prédictions en temps réel sur de nouvelles données. Ce projet permet de mettre en pratique les compétences de machine learning et les adapter pour reconnaitre le genre musical sur une plateforme embarquée. On a donc utiliser une plateforme d'apprentissage puis on a transféré les modèles sur la plateforme d'inférence, ici une carte Raspberry Pi.

Les performances obtenues sont les suivantes :

Modèle	Accuracy entrainement	Accuracy inférence	temps d'évaluation
SVM	46.36%	34.85%	0.30038s
Decision Tree	41.82%	41.82%	0.00977s
Random Forest	50.61%	50.30%	0.02548s
Neuronal Network	62.12%	60.30%	0.14773s