Projet final - Apprentissage profond

Maxime Wirth, Alexandre Radin

Avril 2024

1 Introduction

Le projet consiste à exploiter des pistes audio afin d'y extraire les instruments présents. Le code se décompose en deux parties : un fichier permettant de générer les datasets en format csv à partir de pistes wav, et un fichier permettant la création et l'entraînement du modèle.

2 Dataset utilisé

Nous avons utilisé le dataset IRMAS. Il a été construit pour la reconnaissance d'instruments dans des pistes audio. Il comprend donc des extraits musicaux avec des annotations du ou des instruments prédominant(s). Les instruments considérés sont : violoncelle, clarinette, flûte, guitare acoustique, guitare électrique, orgue, piano, saxophone, trompette, violon et voix. Le training set contient 6705 fichiers audio durant 3 secondes chacun. Le testing set contient 2874 fichiers audio. La taille des pistes varie entre 3 et 20 secondes.

3 Création du dataset

La première étape a été de construire les fichiers csv en ne gardant que les informations importantes (nom de la piste, instruments présents, et durée). Les données du dataset sont présentées comme un fichier .txt et .wav pour chaque piste (ou track). Le fichier .wav contient l'audio à proprement parler, et le .txt contient les abréviations des instruments présents dans la piste, un par ligne. Les pistes sont découpées de telle sorte que ces instruments sont les seuls présents sur toute la piste, et sont tous toujours présents.

3.1 Extraction des caractéristiques du testing set d'IRMAS

La partie test d'IRMAS est découpée en trois portions. Nous itérons sur tous les fichiers et lisons les fichiers texte associés qui contiennent la ou les classe(s) des instruments. Nous extrayons aussi la durée de chaque piste grâce à la librairie librosa. On stocke tout cela dans un fichier csv (nom, durée, et classes).

3.2 Extraction des caractéristiques du training set d'IRMAS

La partie train d'IRMAS est découpée en autant de dossiers qu'il y a de classes. Ici, il n'y a pas de fichier texte associé mais chaque fichier contient dans son nom la ou les classe(s) présente(s). On itère sur chaque fichier et on extrait de son nom la ou les classe(s). Pour ce faire, on repère les brackets ([]). On stocke dans un fichier csv le nom, la durée (qui est ici fixe), et les classes.

3.3 Utilisation des csv dans le workflow

À cause du grand volume de données géré par l'ordinateur (et pour des raisons de rapidité lors des tests), nous avons décidé de poser une étape supplémentaire avant les entraînements. Nous sauvegardons le dataframe dans des fichiers dans l'architecture de fichiers (qui ne sont pas présents dans le repo Github pour des raisons de place : plusieurs centaines de Mo par fichier). Après avoir sauvegardé les données dans des fichiers, nous n'avons plus qu'à les récupérer dans des variables, ce qui est très rapide, puis les utiliser. Nous avons utilisé pour cela la librairie pickle.

3.3.1 Création des spectrogrammes

À ce stade, nous avons deux fichiers csv contenant les attributs que nous avons sélectionné. Cependant, le dataset n'est pas encore construit : nous allons extraire le spectrogramme associé à chaque piste et effectuer quelques opérations supplémentaires (normalisation et découpe des pistes du testing set en portions de 3 secondes).

Pour extraire le spectrogramme, nous définissons une fonction. Nous utilisons la librairie librosa. Elle permet d'extraire le spectrogramme en spécifiant les paramètres (fréquence max, n_mels, ...). Avant de retourner le spectrogramme associé à chaque piste, nous le normalisons par bandes en appelant un StandardScaler (ici, nous découpons le spectrogramme en 4). Nous avons aussi testé d'utiliser un MinMaxScaler par bandes ainsi que sur tout le spectrogramme.

Les essais sans aucune normalisation aboutissaient à une explosion de la

3.3.2 Découpe des pistes de test en portions de 3 secondes

Pour correspondre à l'entrée du modèle, nous avons besoin de pistes audio durant 3 secondes. En effet, le spectrogramme extrait est de taille (128, 130). Pour cela, nous définissons une fonction qui extrait le spectre et le découpe en plusieurs portions de taille (128, 130) selon sa longueur initiale.

3.3.3 Création des labels

Nous choisissons d'utiliser le format one-hot pour nos labels. Nos pistes peuvent contenir un ou plusieurs instrument(s).

Nous faisons une partie du prétraitement à cette étape : pour les features, nous récupérons le spectrogramme de chaque piste dans les variables x_train et x_test. Pour les labels, nous récupérons pour chaque piste la liste des instruments, que nous transformons en vecteurs one-hot dans les variables y_train et y_test.

Ensuite pour chaque élément de toutes ces listes, nous appliquons une transformation en tenseur. Enfin, nous sauvegardons ces listes dans notre architecture de fichiers afin de les récupérer pour l'entrainement.

4 Création du modèle et entraînement

Nous avons utilisé un CNN pour extraire les caractéristiques de nos spectrogrammes et prédire l'instrument présent.

4.1 Architecture du modèle

Comme mentionné précédemment, le modèle prend en entrée des spectrogrammes de la forme (128, 130, 1). Il applique plusieurs couches de convolution et de max pooling afin d'extraire et de sélectionner/synthétiser les caractéristiques. Une fois le spectrogramme compressé, nous appelons une couche Flatten pour concaténer le résultat dans un vecteur 1D afin de l'introduire dans des couches densément connectées. Nous passons par une première couche de 100 neurones, une deuxième de 30 neurones, puis une dernière couche pour la prédiction avec une fonction d'activation sigmoïde. Dans toute l'architecture, nous introduisons des Dropout pour réduire la dépendance du modèle vis-à-vis de certains neurones spécifiques. Cela nous permet de faire face à de l'overfitting.

4.2 Entraînement

Afin de procéder à l'entraînement, nous récupérons d'abord les données depuis les fichiers sauvegardés. Durant toute notre période de tests nous avons fait de nombreux GridSearch afin de trouver les meilleurs paramètres pour notre modèle. Nous avons aussi appliqué une validation croisée sur 3 folds afin d'obtenir les moyennes et ainsi faire face à une bonne accuracy obtenue "par hasard" sur un shuffle spécifique.

Les paramètres que nous avons fait varier ont été la batch_size, le nombre d'epochs ainsi que la validation_split.

Nous avons utilisé pour la batch_size les valeurs 8, 16, 32, et 64.

Nous testions d'abord sur 50 epochs, puis pour les modèles qui paraissaient avoir les meilleures performances, nous avons testé 100 à 150 epochs.

Pour la validation_split, nous avons testé 0.1, 0.2, 0.3, 0.4 et 0.5.

5 Difficultés rencontrées

Au cours du projet, nous avons rencontré plusieurs difficultés au niveau de l'entraînement du modèle. La loss de ce dernier explosait et son accuracy ne dépassait jamais les 25%.

Pour résoudre le problème, nous avons joué avec la génération des spectrogrammes et leurs tailles, la normalisation (sur l'ensemble du spectre ou par bandes), la fonction d'activation, la fréquence maximale des spectres, ou l'architecture du modèle (voir les rapports de sprint). Malgré tous nos efforts, nous n'avons pas réussi à entraîner le modèle.

6 Conclusion

Ce projet est devenu, au fil de temps, un projet exploratoire. Nous avons énormément testé au niveau de l'architecture, prenant aussi exemple sur un modèle trouvé sur Github.

Après avoir en partie résolu le problème d'explosion de la loss, l'objectif a été de produire un modèle convenable. C'est à ce moment que nous avons largement modifié la facon de construire les données afin d'arriver à cette version finale.