Rapport Projet TDLog

Oscar CLIVIO, Andrei KARTASHOV, Lucas BRIFAULT

13 février 2017

Introduction

Check it Out est une startup parisienne qui s'adapte aux demandes actuelles de consommateurs. Elle a été proposée dans le cadre du cours TDLOG et notre équipe a été en lien direct avec l'équipe de fondateurs : Carole (EDHEC 2014), Sofiane (autodidacte) et Quentin (Ponts 2009). Cette start-up a pour but de développer une application permettant à ses utilisateurs de poster des commentaires vidéo sur des endroits (restaurants, bars ou autres) visités, afin de fournir un témoignage plus vivant, concret, voire même plus fiable, si la vidéo est prise sur place. L'objectif de ce projet est la création d'un prototype d'application mobile qui peut améliorer le son de fichiers vidéo en appliquant les différents filtres. Grâce à cette application, l'utilisateur pourrait par exemple réduire le bruit du vent et des moteurs de véhicules ou il peut augmenter spécifiquement le volume de sa voix.

Recherche des Filtres Audio

Choix du langage de programmation

Dans le but de développer et tester différents filtres audio le plus rapidement possible, il nous a fallu choisir un langage de programmation à la fois facile à manipuler et performant d'un point de vue "exigences scientifiques". Nous nous sommes alors naturellement orientés vers Python, qui en plus avait le mérite d'être bien connu de nous trois. Ce langage script dispose de surcroît de bibliothèques très pratiques pour la lecture et l'enregistrement de fichiers son, le traitement rapide du signal et l'analyse de la qualité des filtres créés (scipy, pylab, matplotlib...).

Les bases : le son

Avant de se lancer bille en tête dans la programmation d'une batterie de filtres, et comme nous l'a fortement suggéré notre encadrant, il s'agissait de bien comprendre, sinon de bien se rappeler ce qu'est le son et comment on peut l'interpréter.

Le son est une fluctuation de pression (le plus souvent dans l'air, mais ce peut être un autre milieu) selon certaines fréquences, fréquences qui vont, ou non, être captées par l'oreille humaine (domaine audible : $20Hz$ - $20000Hz$). Le son peut être interprété de plusieurs façons, on a notamment :
-L'interprétation temporelle (courbe de la variation de pression en fonction du temps)
-L'interprétation fréquentielle (en considérant que le signal est une somme de sinusoïdes à différentes fréquences, phases et amplitudes)

D'un point de vue plutôt "instrumental", on peut caractériser un son (ex : la voix) selon différents critères :
-La hauteur (la fondamentale)
-Le timbre (le spectre)
-L'enveloppe, qui est en quelque sorte la variation en amplitude des fluctuations, et qui correspond à la forme globale que prend la courbe temporelle du signal.

On peut également considérer un signal discrétisé (échantillonné, comme c'est toujours le cas numériquement) comme la réalisation d'un nombre fini de variables aléa-

toires. Par exemple on peut voir le signal total comme la somme de vecteurs aléatoires correspondant à chaque source sonore. Cela est assez puissant, puisqu'il nous donne accès à des outils probabilistes comme l'indépendance, qui peut aider dans certains cas à séparer certaines sources...

Les bases : l'analyse du signal

Nous présentons ici des outils et algorithmes très classiques en traitement du signal, qui nous ont guidé dans le choix des filtre (faisabilité...).

Bien sûr on commence avec la transformée de Fourier et la transformée de Fourier inverse, qui permettent justement (cf section ci-dessus) de passer de la représentation temporelle à la représentation fréquentielle du signal et réciproquement. Evidemment, on utilise i ?i les versions discrètes. Dans un soucis de rapidité, essentielle pour l'exécution des tests, et surtout pour la performance finale des filtres, nous n'avons pas utilisé la transformée simple, mais la "Fast Fourier Transform" (FFT) qui est un algorithme qui s'inspire du principe "divide and conquer", en calculant récursivement les transformées de sous-parties du signal discret (en séparant les indices pairs et les indices impairs du tableau contenant les points du signal). Cet algorithme est bien plus efficace, même si il nécessite en général qu'on lui fournisse en entrée un tableau dont la taille est une puissance de 2. En ce qui concerne python, cet algorithme est déjà inclus dans la bibliothèque scipy, même si il est facilement programmable.

Le second outil indispensable est la "Short Term Fourier Transform" (STFT), qui part du constat suivant : la transformée de Fourier ne permet qu'une vision globale de la présence de chaque fréquence sur l'ensemble du signal. Mais un signal n'est pas homogène et évolue dans le temps, et certaines fréquences sont plus ou moins présentes selon l'instant considéré. La STFT va alors "découper" le signal selon des fenêtres à différents pas de temps, et effectuer une transformée de Fourier (bien sûr on prendra aussi la FFT) sur chacun des morceaux obtenus. On dispose alors de la répartition des fréquences selon les pas de temps choisis. Il existe différents types de fenêtres qui "pondèrent" le signal d'une manière qui se veut optimale. Nous avons quant à nous choisi d'utiliser le modèle de la fenêtre de Hann, dont voici la formule :

$$w[i] \ = \ \frac{1}{2}\left(1 \ - \ cos(\frac{2i\pi}{N})\right)$$

où N est la largeur choisie de la fenêtre, et w est le tableau qui va pondérer la section du signal considérée (section qui est donc de largeur N, et qui démarre à un certain instant $t \ = \ k \times h$ où h est le pas de temps choisi).

Les filtres

Les filtres que l'on cherche à élaborer doivent être pensés selon le contexte d'utilisation de l'application (restaurant, bar, extérieur...). Nous avons donc dû lister (en en

discutant notamment avec notre encadrant) un certain nombre de nuisances sonores que l'utilisateur trouvera bon d'éliminer ou de diminuer. Parmis ces nuisances on trouve : Le vent, les bruits de moteur, les bruits de couverts, les bruits parasites (crachement...), le brouhaha (ex : discussions voisines)... Au contraire on peut chercher à amplifier ou garder certaines parties du signal (musique, explication vocale de l'utilisateur...).

Nous avons dû éliminer rapidement les traitements qui paraissaient trop ambitieux dans le temps qui nous était imparti. Par exemple, supprimer le brouhaha ambiant dans un restaurant en gardant la voix de l'utilisateur peut constituer a priori un véritable tour de force (on veut traiter différemment des sources qui ont les mêmes plages de fréquence, des amplitudes pas nécessairement très éloignées...). On peut cependant (ce qui a d'ailleurs été une de nos idées de filtres durant une période) procéder à une sorte de reconnaissance vocale pour rehausser la voix de l'utilisateur (dont les caractéristiques sont connues a priori) par rapport au reste du signal.

Nous avons finalement décidé de nous concentrer sur :

- Un filtre "Extérieur", à la fois anti-moteur et anti-vent, qui serait lui-même constitué :
  - D'un passe-haut de fréquence de coupure $f_c \approx 400Hz$ pour cibler les fréquences caractéristiques des moteurs de voiture et du vent "doux".
  - D'un filtre "anti-aberration", qui supprime localement les fluctuations trop importantes par rapport au reste du signal. Ce filtre est avant tout destiné à réduire l'impact d'un vent assez "fort", qui viendrait taper sur le micro.

- Un Filtre "Musique" destiné comme son nom l'indique à rehausser la musique par rapport au bruit ambiant. Une première idée était d'appliquer un passe-bas sur le signal de manière à conserver les basses de la musique en diminuant les fréquences correspondant aux bruits de conversation, etc...

- Un Filtre "Reconnaissance Vocale"' nous a paru pertinent en début de projet, en récupérant le spectre vocal de l'utilisateur sur un enregistrement à l'abris des nuisances sonores, pour ensuite comparer ce spectre à ceux des enregistrements futurs de l'utilisateur dans des milieux plus complexes.

- En parallèle il nous paraissait intéressant de nous intéresser à des méthodes de filtrage du bruit (type crachement) pour deux raisons : D'abord il peut être naturellement présent dans certains enregistrements (qualité du micro, environnement...), et de plus, le fait d'appliquer certains filtres (comme le filtre "Extérieur") peut parfois détériorer la qualité du signal, en ajoutant justement ce type de bruit. Nous avons alors eu l'occasion de tester plusieurs types de filtres comme le filtre médian, le filtre "moyen"...
  Certains filtres anti-bruit déjà élaborés sont très efficaces, mais ils nécessitent en général de sélectionner une zone du signal où l'on entend que le bruit seul, pour

pouvoir en déterminer la loi , ou densité de probabilité (si on revient dans la représentation du signal en variables aléatoires) ce qui permet par la suite de quasiment le supprimer sur tout le signal. Mais cela impliquerait une trop forte implication de l'utilisateur dans le processus de filtrage, qui se veut pourtant très simple et intuitif, pour une meilleure expérience.

Programmation d'un prototype d'application

L'objectif principal de la startup pour laquelle nous avons travaillé est la création d'une application mobile qui permettrait à un consommateur d'insérer des commentaires vidéo. A la fin du projet, nous nous sommes interrogés sur la manière de créer une application sur smartphone. Nous nous sommes concentrés sur Android, un système dont les téléphones de toute l'équipe sont équipés.

Python ou Java ?

Nous nous sommes demandés s'il était possible de pouvoir disposer de la puissance des librairies de traitement du son de Python et donc de pouvoir créer des applications Android avec Python. Il existe une bibliothèque permettant le développement de telles applications : il s'agit de Kivy. La compilation du projet en un fichier APK se réalise avec un outil nommé buildozer. Or nous nous sommes rendus compte qu'une de ses composantes principales, l'outil python-for-android, ne fonctionne pas sur Windows. Une solution est d'installer une image disque Linux sur VirtualBox fournie sur le site de Kivy et spécialement conçue pour réaliser la compilation sur Android. Or nous n'avons pu réussir à faire fonctionner cette image disque. Nous nous sommes alors décidés de nous rabattre sur Android Studio, qui utilise Java.

Plan de l'application

Nous avons alors voulu coder un prototype demandant à l'utilisateur de choisir entre l'importation d'un fichier audio ou vidéo ou l'enregistrement de celui-ci puis lui proposant de choisir entre les filtres "Extérieur" et "Musique" que nous avions développés en Python et que nous aurions alors adaptés à Android Studio. L'utilisateur pourrait alors voir le résultat, appliquer un nouveau filtre s'il le désire pour changer et sauvegarder le résultat.

Pour cela, nous avons pensé à un schéma reposant sur quatres activités Android : une pour le menu principal pouvant aboutir à une pour le choix d'un fichier et une autre pour son enregistrement, ces deux-là aboutissant systématiquement à une dernière pour le choix du filtre, la prévisualisation et l'enregistrement du résultat. Pour rappel, une activité est la composante principale d'une application Android, équivalent à une fenêtre pour une application sur PC.

Nous avons estimé que ce schéma serait une bonne base de départ pour l'application. L'activité du menu principal pourrait facilement être étoffée, par exemple avec

un service de connexion sur un compte en ligne et cela éventuellement dans un autre activité, celle du choix d'un fichier remplirait sa mission avec recherche dans les données multimédia du téléphone puis affichage d'une vue sous forme de liste (ListView), l'enregistrement aurait sa propre activité pour visualiser la vidéo tout juste enregistrée ou éventuellement implémenter son propre dispositif de filmage sans passer par l'application Caméra d'Android, et enfin l'activité finale pourrait être enrichie par de nouveaux filtres et accueillir plus d'options de partage, voire de publication sur un site Internet. Un autre avantage de ce découpage en activités est que l'utilisateur peut revenir en arrière à chaque étape simplement avec la touche de retour en arrière : par exemple pour enregistrer une nouvelle vidéo s'il en est aux filtres et s'il avait déjà enregistré une vidéo au préalable.

Malheureusement, le temps disponible et l'ampleur des notions d'Android Studio à maîtriser pour cela ne nous a pas permis, à l'heure où nous écrivons ces lignes, de pleinement développer le prototype selon ce plan. En nous concentrant sur les fichiers vidéo, nous avons pour le moment programmé :

- une interface graphique pour le menu principal. Le texte et les boutons y sont blancs sur fond noir. Nous avons jugé cela plus esthétique que l'interface par défaut d'Android.
- un dispositif d'enregistrement vidéo simple, faisant directement appel à l'application Caméra d'Android. Nous n'avons pas encore réussi à implémenter l'exploration des fichiers vidéo. Pour cependant donner des indications sur comment nous pourrions procéder, l'idée générale est d'obtenir un curseur portant sur une requête des fichiers vidéo du téléphone et d'en déduire une vue de la liste des fichiers vidéo du téléphone sous forme de ListView.
- l'interface graphique de la première étape de la partie filtrage, le choix entre les deux filtres et le bouton permettant d'appliquer ce choix. Cependant, nous n'avons pas encore pu implémenter les filtres réécrits en Java dans cette interface. Notons qu'il sera alors nécessaire d'extraire la piste audio de la vidéo : cela semble pouvoir se faire en portant l'application FFMPEG sur Android Studio à l'aide du Android NDK.

Difficultés générales

Pendant la confection des filtres

L'un des premiers obstacles qui est apparu rapidement est que si l'on se lance "à l'aveugle" dans le projet, on peut très vite se retrouver dans l'une des situations suivantes :

- On essaie de dénicher un maximum de pistes de travail pour maximiser les chances d'en avoir quelques unes qui aboutissent à de bons résultats, mais on prend le risque de s'éparpiller.
- On se concentre sur un nombre réduit de pistes (en fonction du nombre de per-

sonnes dans l'équipe) pour pouvoir bien les approfondir, mais si peu d'entre elles aboutissent, on se retrouve avec pas grand chose en terme de résultats.

Il était donc essentiel de réduire le plus possible cette part d'aveugle en partant d'un socle solide : littérature sur le sujet, culture personnelle. Typiquement l'application de filtres passe-haut, passe-bas, passe-bande se trouve préférentiellement parmis les premières étapes, pour commencer à avancer en terrain connu.

A cela s'est ajouté la difficulté à juger de la qualité d'un filtre. On peut en effet écouter les résultats pour certains enregistrements, mais comment être sûr que le filtre fonctionnera pour d'autres, pris dans un environnement différent ou avec des sources sonores supplémentaires ? On peut bien sûr comparer les spectres et courbes temporelles des signaux avant et après traitement, ce qui permet de percevoir un peu plus concrètement comment le filtre agit sur le signal, mais cela reste très peu général.

Enfin, on peut citer le réglage des paramètres. Chacun des filtres et presque chacune des fonctions qu'il utilise a été conçu(e) avec un certain nombre de paramètres (en arguments d'entrée en général) pour pouvoir maîtriser en partie son comportement (puissance de réduction des fréquences indésirables, taille des fenêtres de la STFT, fréquences de coupures, etc...). L'ajustement de ces paramètres s'est révélé aussi fastidieux que déterminent pour le fonctionnement des filtres. Et il est compliqué d'automatiser l'optimisation de ces paramètres étant donné que l'on ne dispose justement pas d'une fonction pouvant quantifier la qualité d'un filtre.

Au passage à l'application

Parmis les principales difficultés intervenues dans la phase de développement du prototype d'application on retrouve le fait que l'on ne dispose évidemment plus du tout des mêmes bibliothèques, pourtant bien pratiques sur python pour implémenter et tester rapidement nos fonctions/filtres (d'où le choix de réécrire une partie des outils de ces librairies "à la main").

Nous avons également dû nous intéresser à la gestion des fichiers audio et vidéo, de laquelle nous n'avions absolument pas à nous occuper auparavant. En fait un grand nombre de tâches qui n'étaient pas automatisées lorsque nous étions à un niveau que l'on peut qualifier de "programmation scientifique" doivent maintenant le devenir le plus possible dans le soucis de simplifier la vie de l'utilisateur (et dans notre cas, aboutir à un prototype qui permette une bonne expérience).

De plus, Java est... lent. Il ne permet pas la même rapidité de calcul que les bibliothèques de python telles scipy (dont une partie est potentiellement codée en C++, ce qui peut expliquer son efficacité).

Enfin, le développement même de l'application sur Android se révèle être un art plus complexe que nous ne l'avions imaginé, que ce soit dans la communication entre différentes activités ou l'utilisation de fonctionnalités d'Android a priori évidentes comme l'exploration de fichiers ou la caméra. En commençant l'apprentissage la programmation sur Android à la fin du projet, nous n'avons pas eu le temps d'acquérir une parfaite maîtrise des différents outils d'Android Studio, et la diversité des fonctionnalités du prototype fait appel à nombre de ces outils.

Conclusion et Perspectives

En conclusion, nous avons travaillé sur un projet actuel qui pourrait apporter plus de consistance et de sincérité aux commentaires de consommateurs. Dans ce cadre, les utilisateurs peuvent choisir le filtre approprié qui va améliorer le son d'un fichier vidéo. Notre application propose deux types de filtres : le filtre "Extérieur" qui est destiné à réduire l'impact d'un vent et le filtre "Musique" qui est attribué de l'élévation la musique par rapport au bruit ambiant. Ces filtres sont réalisés à l'aide de l'utilisation de FFT et STFT.

En ce qui concerne les perspectives de notre projet on peut remarquer trois points importants. Premièrement, l'application marche seulement pour Android tandis que iOS est clairement un système à ne pas négliger. Deuxièmement, le nombre de filtres qui sont réalisés et implémentés ne permet pas encore de traiter tous les types de vidéo. Par exemple, il nous manque encore un filtre "Reconnaissance Vocale"' qui pourrait récupérer rehausser la voix de l'utilisateur. Enfin, quelques filtres nécessitent de souvent changer des paramètres dont ils dépendent. L'idéal serait bien de développer un filtre indépendant de l'environnement et de la voix.