



École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique 5^e année 2020-2021

Cahier d'analyse

Canne connectée pour aveugles

Encadrants

Gilles VENTURINI
gilles.venturini@etu.univ-tours.fr

Auteurs

Djawad M'DALLAH MARI
djawad.mdallah-mari@etu.univ-tours.fr

Université François-Rabelais, Tours

DII5 2020-2021

Version du 15 janvier 2021

Table des matières

1	Cahier d'analyse	3
1.1	Introduction	3
1.2	Reconnaissance d'objet	3
1.2.1	Librairies	3
1.2.2	Modèles	3
1.2.3	Intégration dans une application Android	8
1.3	Informier l'utilisateur	8
1.3.1	Synthèse vocale	8
1.3.2	Vibration	9
1.4	Navigation	9
1.4.1	Guide d'utilisation	9
1.4.2	Accès aux réglages	9

Cahier d'analyse

1.1 Introduction

Ce cahier d'analyse s'inscrit dans le cadre du projet Canne connectée pour aveugles. Il vise à présenter les analyses faites pour répondre aux besoins exprimés dans le cahier de spécifications. Une lecture au préalable du cahier de spécifications est donc recommandée afin de comprendre le contexte et les enjeux du projet.

Nous verrons donc dans ce document une analyse sur l'application Android à développer. Nous verrons en particulier quelques méthodes de reconnaissances d'objet pour le mobile, les différentes méthodes qui permettront d'informer l'utilisateur et également comment garantir à l'utilisateur une interface adaptée à ses contraintes.

1.2 Reconnaissance d'objet

1.2.1 Librairies

L'un des besoins primaires pour la réalisation de ce projet est la reconnaissance d'objet. Pour cela, il est important de choisir une librairie qui permet d'implémenter un réseau de neurones. Sur Android il existe l'Android Neural Networks API (NNAPI) qui est un API en C permettant de réaliser des opérations de machine learning. Il a été conçu pour fournir une couche de fonctionnalités bas niveau qui sera ensuite utilisable par des couches plus hauts niveau comme les frameworks TensorFlow ou encore Caffe2. Toutefois, il est tout à fait possible de faire abstraction des frameworks et implémenter une solution utilisant directement l'API. L'utilisation la plus commune est avec TensorFlow et plus précisément TensorFlow Lite qui est une version allégée de TensorFlow. Cette version allégée limite certaines opérations de TensorFlow lui permettant donc d'être utilisée pour les appareils embarqués. Il existe également Keras, un autre outil intéressant permettant de faire du machine learning. Cependant, dès lors que l'on souhaite déployer sur Android, il faudra passer par TensorFlow Lite et convertir le modèle en format TensorFlow Lite¹. Dans notre cas, l'utilisation de TensorFlow Lite pourrait suffire pour répondre à notre besoin. De plus, celle-ci dispose d'un large panel de modèles préentraînés comme on va le voir dans les parties qui suivent.

1.2.2 Modèles

Un modèle est un fichier qui a été entraîné pour reconnaître certains types de motifs (pattern). Il est entraîné à partir d'un ensemble de données et utilise un algorithme qui lui permet "d'apprendre" à travers ces données.

Le choix du bon modèle est un facteur important pour répondre au besoin de reconnaissance d'objet. En effet, toute l'application, pour qu'elle soit utile aux utilisateurs finals, dépend

1. <https://keras.io> : partie "Deploy Anywhere."

de la capacité du modèle à détecter et identifier un objet. Afin de répondre à ce besoin, il faudrait faire un inventaire des modèles de reconnaissance d'objet disponible puis faire des comparaisons. Pour mesurer les performances de chaque modèle, des critères doivent être établis (Latence, Disponibilité, Rapidité, Coût, ...) ainsi que des conditions de fonctionnement bien définis (caractéristique du smartphone, version d'android, version, ...). Cela permettrait d'avoir un environnement d'exécution commun pour chaque modèle et donc des mesures cohérentes.

Modèles disponibles

Avec la librairie TensorFlow, nous disposons d'un grand panel de modèles préentraînés. La plupart de ces modèles ne sont pas directement compatible avec TensorFlow Lite. Ceux sur TF2 Detection zoo² peuvent être converti en utilisant TFLite Converter³. Les modèles déjà adaptés TensorFlow Lite sont disponible sur le hub officiel de TensorFlow⁴ mais peu de modèles de reconnaissance d'image y sont recensés notamment de reconnaissance d'objets (**Object Detection**)⁵. En effet, il existe que trois modèles officiels dans cette catégorie : SSD MobileNet, Mobile Object Localizer et East Text Detector. Parmi ces trois modèles, on peut déjà abandonner le East Text Detector puisqu'il s'agit ici de détecter du texte. En revanche, d'autres modèles de reconnaissance d'objet sont disponibles sur le github de TensorFlow⁶ et sont à priori déjà compatible mobile. Ces modèles là sont intéressants car ils n'ont pas tous été entraînés avec la même banque d'images, on pourra donc choisir celui qui répond le plus à notre besoin (voir partie Choix du modèle 1.2.2). Au niveau des modèles de **Classification**, il existe un peu plus de modèles compatibles TensorFlow Lite⁷. Les modèles de classification, malgré le fait qu'ils sont plus précis, vont moins répondre à notre besoin initial, mais méritent d'être évoqués puisque ceux-ci pourraient être mis en place dans une version futur du projet ; en complément du modèle de reconnaissance d'objet choisi par exemple.

Object Detection vs Classification

Les modèles de reconnaissances d'objet (Objet Detection) sont capables de localiser et identifier plusieurs objets sur une même image.

Ce type de modèle est entraîné avec des objets de différentes classes (vêtements, fruits, etc.). Avec TensorFlow, lorsqu'on met à l'entrée de ce type de modèle une image, on obtient en sortie une liste d'objets avec chacun sa localisation, sa classe et un degré de confiance (qui correspond à la fiabilité de l'objet identifié).

En revanche, les modèles de **classification** avec TensorFlow ne détecte qu'un seul élément sur une image. Ces modèles sont entraînés sur une seule classe générique (exemple : vêtements, aliments, plantes, etc) qui va ensuite être capable d'identifier l'élément de manière plus précise. Exemple :

— Classe vêtements : T-shirt, jean, ...

2. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md

3. <https://www.tensorflow.org/lite/convert>

4. <https://tfhub.dev>

5. <https://tfhub.dev/s?deployment-format=lite&module-type=image-object-detection>

6. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md

7. <https://tfhub.dev/s?deployment-format=lite&module-type=image-classification>



FIGURE 1.1 – Exemple de reconnaissance d'objet

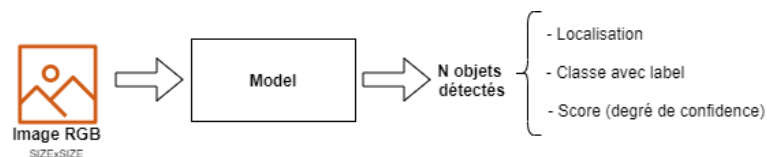


FIGURE 1.2 – Schema principe de fonctionnement d'un modèle

- Classe aliments : salade, pâtes, ...
- Classe fruits : pomme, banane, ...
- Classe insectes : sauterelle, abeille, papillon, ...
- ...

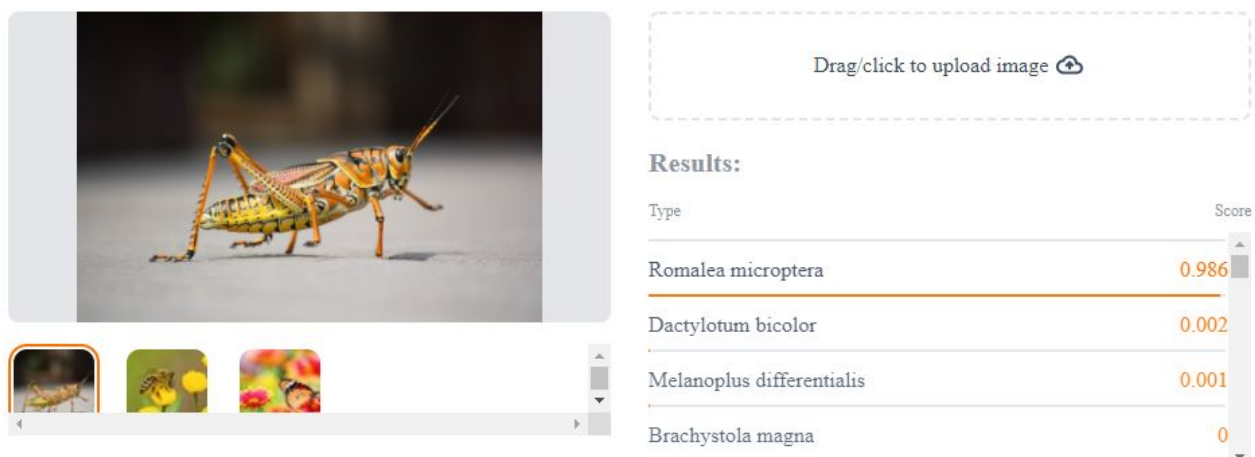


FIGURE 1.3 – Exemple de classification d'insectes

Choix du modèle

Notre objectif pour notre premier prototype est d'utiliser un modèle qui reconnaît un maximum d'objet dans différentes classes. Cela permettrait donc aux utilisateurs finaux de détecter un ensemble d'objets de différentes catégories plutôt que d'une seule catégorie spécifique. De ce point de vue, les modèles de classification tels qu'on les a vu précédemment ne correspondraient donc pas à ce besoin. Néanmoins, pour des versions ultérieures de l'application, il pourra être envisager, soit une amélioration du modèle utilisé pour que celui-ci soit plus précis, ou ajouter la possibilité de changer de modèle au sein de l'application. Il nous faut donc choisir parmi les deux modèles disponibles évoqués précédemment à savoir SSD MobileNet et Mobile Object Localizer. On pourra par la suite comparer le modèle choisi parmi ces deux là avec les autres modèles entraînés à l'aide d'autres banque d'images⁸.

Ces deux modèles permettent donc d'identifier des objets, parmi les objets connu du modèle, et d'en fournir leurs positions dans l'image. Tout deux utilisent l'architecture de réseau de neuronne MobileNet⁹. SSD MobileNet a été entraîné avec la banque d'image COCO dataset¹⁰. Cependant une autre version disponible uniquement sur PC a été entraîné avec Open Images V4¹¹. En revanche aucune information sur la banque d'image utilisée pour Mobile Object Localizer. Pour faire notre choix de modèle, il est intéressant de comparer ses deux modèles en faisant un benchmark. En effet TensorFlow dispose d'un outil qui permet de mesurer et faire des calculs de statistiques sur les modèles. L'outil permet de mesurer le temps d'initialisation du modèle, le temps de traitement (inference time), et bien d'autres données. Dans notre cas le critère intéressant à mesurer serait la latence du modèle. Les résultats des benchmarks varient beaucoup en fonction du smartphone utilisé. Ci-dessous les résultats de benchmarks qui ont été fait par TensorFlow sur différents appareils.

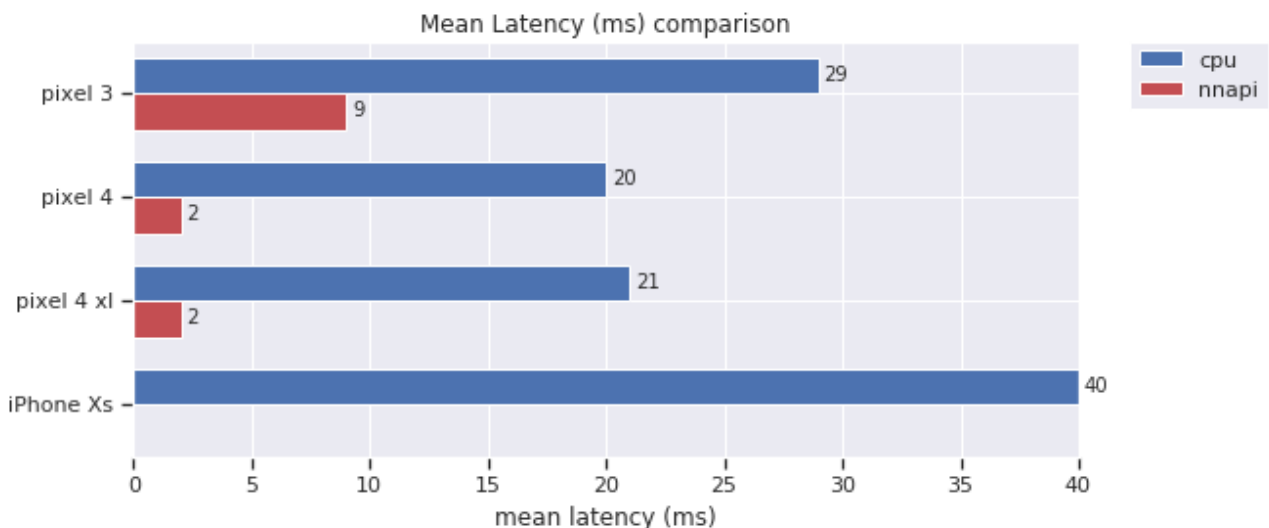


FIGURE 1.4 – Benchmark de Mobile Object Localizer sur plusieurs appareils

8. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md

9. <https://arxiv.org/abs/1801.04381>

10. <https://cocodataset.org/>

11. <https://storage.googleapis.com/openimages/web/index.html>

(Remarque : Ici aucune indication sur le nombre de threads utilisés)

Model Name	Model size	Device	GPU	CPU
COCO SSD MobileNet v1	27 Mb	Pixel 3 (Android 10)	22ms	46ms*
		Pixel 4 (Android 10)	20ms	29ms*
		iPhone XS (iOS 12.4.1)	7.6ms	11ms**

* 4 threads used.

** 2 threads used on iPhone for the best performance result.

FIGURE 1.5 – Benchmark de SSD MobileNet sur plusieurs appareils

Travaillant sur le Samsung Galaxy S10e dans le cadre de ce projet, il est intéressant de mesurer les performance de ces deux modèles sur ce smartphone également. Ci-dessous les résultats obtenus :

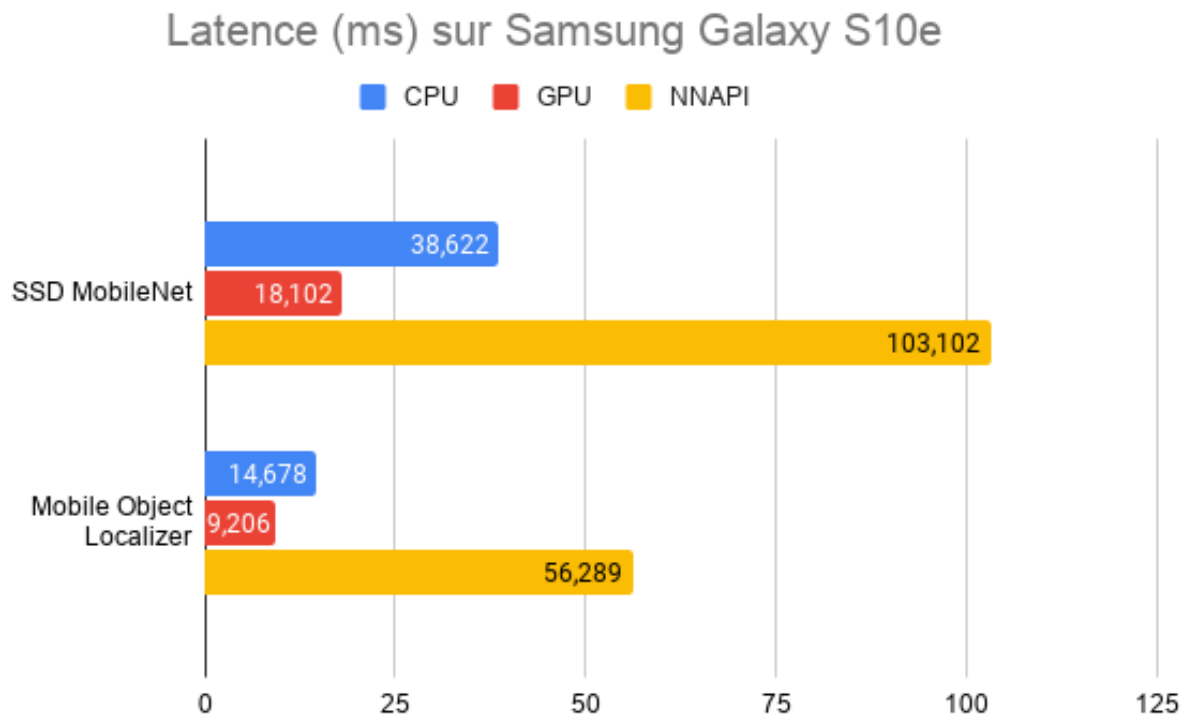


FIGURE 1.6 – Benchmark sur Samsung Galaxy S10e

Pour conclure sur ces mesures, on peut voir donc clairement que le modèle SSD MobileNet présente plus de latence que Mobile Object Localizer. Il serait donc plus judicieux de choisir ce modèle là. Cependant pour utiliser ces modèles, nous avons besoin d'un fichier important parmi les métadonnées du modèle qui est le fichier label. Ce fichier stocke les labels de toutes les classes d'objets connus par le modèle. En l'absence de ce fichier qui doit accompagner le modèle, il est

donc impossible d'attribuer un label à un objet détecté. Or, le fichier accompagnant le Mobile Object Localizer est vide et aucun moyen de le retrouver ou d'extraire ces données. On peut d'ailleurs voir directement le résultat de l'absence de données dans ce fichier sur le site ¹² :

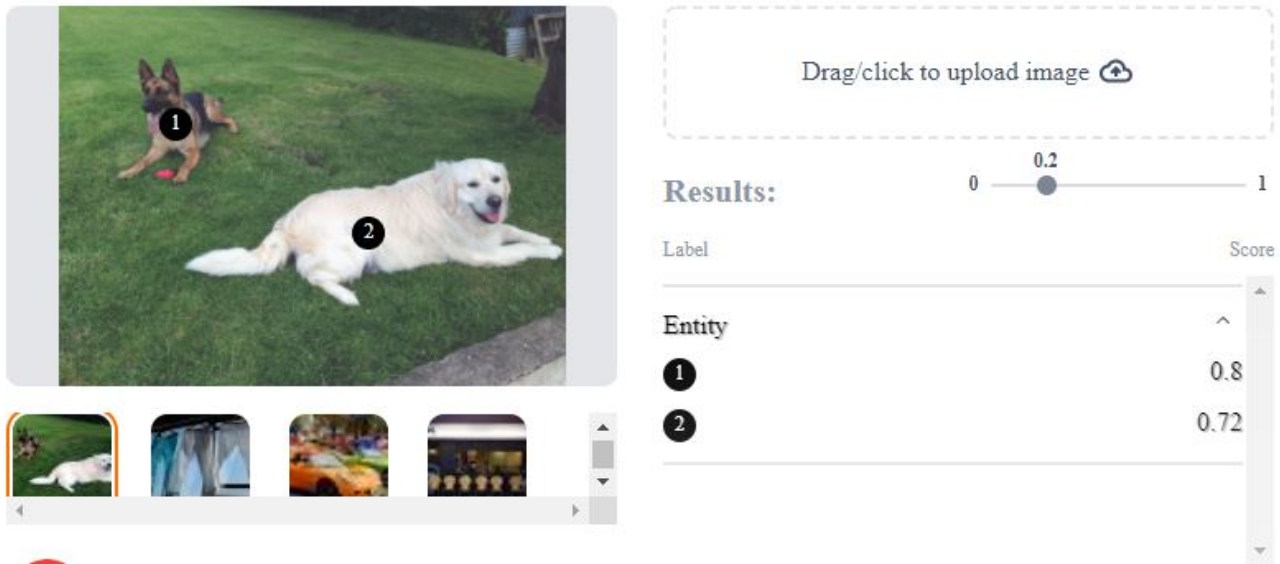


FIGURE 1.7 – Labelmap vide

Les objets sont détectés et localisés mais aucun label leur est assigné. Souhaitant informer l'utilisateur de l'objet détecté, nous retenons plutôt le modèle SSD MobileNet.

Mobile Object Localizer quoi ? localiser ? éliminé !

SSD MobileNet quoi, banque d'image, graph ?... Different mobileNet qui existe (changement minim) :

Zoo Modeles Comparaison

notre besoin (max d'objet, existant, fiable..) benchmark.. banque d'image classification Dans notre cas ident 1 objet à la X(object id)

1.2.3 Intégration dans une application Android

les méthodes d'intégration fonctionnement entrée sortie

1.3 Informer l'utilisateur

1.3.1 Synthèse vocale

fonctionnement paramétrage rendre le message compréhensible : viseur, vibreur encodage-trame (1mots, phrase, 1vib, 2vib, vib long.. ?) diag de classe

12. https://tfhub.dev/google/lite-model/object_detection/mobile_object_localizer_v1/1/default/1

1.3.2 Vibration**1.4 Navigation****1.4.1 Guide d'utilisation****1.4.2 Accès aux réglages**

Table des figures

1.1	Exemple de reconnaissance d'objet	5
1.2	Schema principe de fonctionnement d'un modèle	5
1.3	Exemple de classification d'insectes	5
1.4	Benchmark de Mobile Object Localizer sur plusieurs appareils	6
1.5	Benchmark de SSD MobileNet sur plusieurs appareils	7
1.6	Benchmark sur Samsung Galaxy S10e	7
1.7	Labelmap vide	8

Liste des tableaux

Canne connectée pour aveugles

Département Informatique

5^e année
2020-2021

Cahier d'analyse

Résumé : Cahier d'analyse canne connectée pour aveugles

Mots clefs :

Abstract:

Keywords: Encadrants
Gilles VENTURINI
gilles.venturini@etu.univ-tours.fr

Université François-Rabelais, Tours

Auteurs
Djawad M'DALLAH MARI
djawad.mdallah-mari@etu.univ-tours.fr

DII5 2020-2021