Rapport SAE 501

Semaine 4

MALORON Arthur,
DOTTO Matis,
MANICK Luc,
BELMADANI Abdourrahmane

Sujets abordées

Intégration du modèle d'IA

Conversion du modèle

Nous avons réalisé au cours de cette semaine que l'outil open source que nous avions utilisé pour convertir notre modèle .weights en .pb (pour ensuite pouvoir le convertir en .tflite) nous donnait un modèle dont les tensors d'entrées étaient :

```
Nom du tensor d'entrée : input_1:0
Dimensions d'entrée attendues : (None, 416, 416, 3)
```

suivi de 97 fois

```
Nom du tensor d'entrée : unknown:0
Dimensions d'entrée attendues : <unknown>
```

Ce résultat n'est évidemment pas normal, les modèles yolov4-tiny étant censés ne comporter qu'un seul tensor d'entrée.
Le premier tensor est correct, mais le fait que 97 autres tensors vides aient tout de même été configurés est un signe de potentielle corruption de données ou de problème de configuration.

Ainsi, nous ne sommes pas parvenus à trouver le problème dans notre processus de conversion afin d'en faire un outil viable.

Reconnaissance d'objets avec Yolov2-tiny

Suite à ces difficultées à convertir notre modèle yolov4-tiny, nous nous sommes tournés vers notre modèle yolov2-tiny.

Le modèle nous renvoie un tableau à 4 dimensions contenant 13*13*425 logits. Cependant, n'ayant pas la configuration du modèle, nous avons dû étudier cette structure.

Nous nous attendions originellement à recevoir des données sous la forme de :

- 4 valeurs pour les coordonnées de la boundingbox (zone dans laquelle l'objet détecté devrait se trouver)
- 1 valeur contenant la confidence (la probabilité qu'il y ait bien un objet dans la boundingbox)
- 80 valeurs représentant la probabilité, pour chacune de nos 80 classes, que l'objet détecté appartienne à cette classe.

Cela nous mène donc à 85 données pour un objet détecté.

Or, notre modèle nous renvoie des floatArray de 425 logits, ce qui signifie qu'il y aurait en théorie 5 objets par batch.

Cependant, en partant de ce principe, nous avons pensé que les données suivraient toujours cette structure, c'est-à-dire 4 valeurs pour la boundingbox, 1 valeur pour la confidence, et 80 pour les classes. Cependant, même en changeant l'ordre et en essayant de trouver une logique à cette structure, nous ne sommes pas parvenus à comprendre comment ces données sont organisées.
En effet, nous avons toujours soit des données négatives dans la boundingbox, soit une confidence négative, soit une confidence supérieure à 1.

Nous sommes encore en train de rechercher une logique dans les données renvoyées, mais nous n'avons pas pu trouver de documentation à ce sujet étant donné que cela dépend grandement de la configuration du modèle.

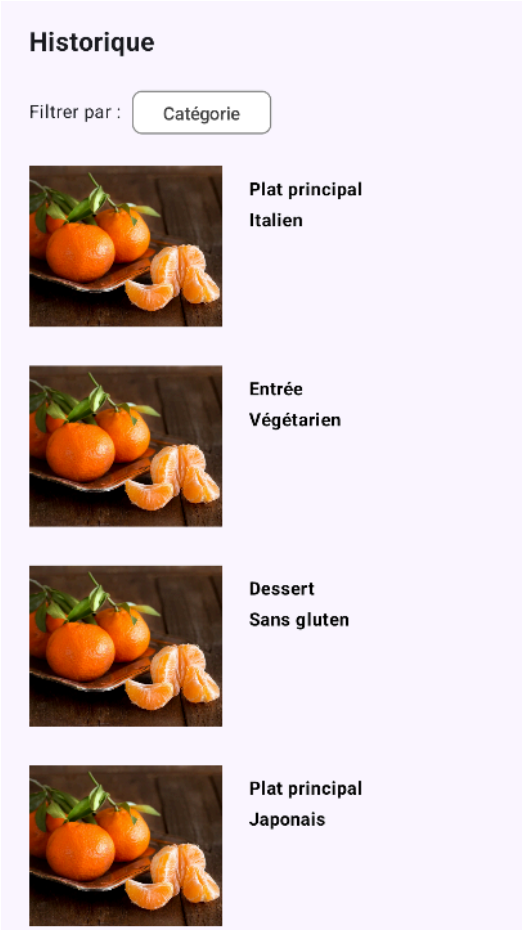
Choix et création de la base de données

Nous avons choisi d'utiliser une base de données SQL pour notre projet. Cette base de données est conçue pour stocker les historiques des reconnaissances d'objets. Elle dispose de trois tables :

image(image_id : int, url : varchar(x))
category(category_id : int, label : varchar(x))
image_category(#image_id, #category_id)

Chaque reconnaissance est effectuée à partir d'une image. Lorsqu'une reconnaissance est réalisée, le chemin de l'image correspondante est ajouté dans la base de données. Les images sont stockées sur un serveur, et chaque image peut être associée à une ou plusieurs catégories, qui sont également enregistrées dans la base.

Avec Android Studio, il n'est pas possible de communiquer directement avec une base de données distante. Il est donc nécessaire de passer par une API intermédiaire. Nous avons déjà commencé à implémenter sur notre application un système permettant de communiquer avec une API et d'afficher les données sur la page des historiques. Pour ce faire, nous utilisons la bibliothèque Retrofit, qui simplifie cette tâche.



Cependant, plusieurs éléments restent à implémenter. Pour tester cette nouvelle fonctionnalité, nous utilisons actuellement un service qui génère une fausse API. Bien que l'application parvienne à interagir correctement avec cette API, il faut encore gérer l'affichage des images.

La prochaine étape sera de développer notre propre API, hébergée sur un serveur distant, où toutes les images seront stockées. Nous pourrons ensuite nous concentrer sur le chargement et l'affichage des images sur la page, en utilisant une bibliothèque Android.

Entrainement du modèle

Nous avons utilisé Python et TensorFlow pour créer un modèle .tflite destiné à la détection d'objets dans notre application. Nous avons deux options :

- Entraîner notre propre modèle
- Utiliser un modèle préentraîné à adapter selon nos besoins spécifiques. Cependant, nous n'avons pas encore trouvé de modèle préentraîné correspondant à notre application.

Objectifs hebdomadaires

- Faire fonctionner en priorité la reconnaissance d'objets avec le modèle yolov2-tiny.
- Création du dataset.
- Trouver un modèle préentrainer et l'améliorer.

Lien du Trello

https://trello.com/invite/b/670e75ea832d0d8d7b7625cf/ATTI30db0fce264f5fcced5a98e25047eb67DF2B89AE/s501