Faculté des sciences Rabat

Date : Avril 2025

Détection d'Objets en Temps Réel avec Flutter et TensorFlow Lite

MODULE : Machine learning/Deep learning

Professeur : Abdelhak MAHMOUDI BENAMER

L'IA, c'est comme une caméra magique : elle voit ce qu'on ne voit pas !

MASTER IT TAM

Jamal SAMSAME et Ayoub SAGHRO

Table des matières

0.1 Introduction

0.1.1 Pourquoi ce projet ?

On voulait créer une application mobile qui utilise l'intelligence artificielle pour repérer des objets en temps réel, un peu comme si votre téléphone devenait un détective super malin ! On a choisi Flutter parce que ça permet de coder une seule fois pour iOS et Android, ce qui nous a fait gagner un temps fou. Avec TensorFlow Lite, tout se passe directement sur le téléphone, donc pas besoin d'envoyer des données sur un serveur, ce qui protège la vie privée et rend l'appli rapide comme l'éclair.

0.1.2 Nos objectifs

— Faire une détection d'objets en direct, sans ralentissements (>30 images par seconde).

— Ajouter des fonctions pour analyser des images et du texte.

— Construire une appli bien organisée, facile à améliorer plus tard.

0.1.3 Les outils qu'on a utilisés

On a misé sur :

— **Flutter/Dart** : pour développer l'appli.

— **TensorFlow Lite** : pour faire tourner l'IA directement sur le téléphone.

— **Clean Architecture** : pour que le code soit clair et bien structuré.

— **Bloc et GetIt** : pour gérer les interactions et les dépendances.

0.2 Comment l'appli est construite

0.2.1 Un plan clair

On a organisé l'appli en trois parties, un peu comme des couches dans un gâteau :

— **Présentation** : tout ce que l'utilisateur voit, comme les écrans et les boutons.

— **Domaine** : la logique, comme les règles pour détecter un objet.

— **Données** : où on stocke les modèles d'IA et les infos.

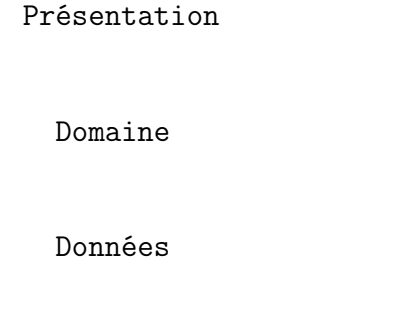

FIGURE 1 – Structure en couches de l'appli

0.2.2 L'interface utilisateur

L'écran principal montre ce que voit la caméra, avec des cadres autour des objets détectés, comme "chat 85%". On a utilisé le pattern Bloc pour gérer les clics et les changements d'état, par exemple pour passer du mode détection au mode analyse.

```
class RealTimeObjectDetectionPage extends StatefulWidget {
  @override
  _RealTimeObjectDetectionPageState createState() =>
   _RealTimeObjectDetectionPageState();
}
class _RealTimeObjectDetectionPageState extends State<
   RealTimeObjectDetectionPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Stack(children: [CameraView(), BoundingBoxes(),
   DraggableSheet()]),
    );
  }
}
```

Listing 1 – Écran principal

0.2.3 La logique métier

On a mis toute la logique dans des "Use Cases". Par exemple, un Use Case dit : "prends cette image et dis-moi ce qu'il y a dessus". Ça rend le code super clair et facile à réutiliser.

0.3 L'intelligence artificielle en action

0.3.1 Détection d'objets

L'appli utilise un modèle TensorFlow Lite pour repérer les objets en direct. Imaginez que vous pointez votre caméra sur un chien : l'appli le détecte et affiche un cadre autour avec "chien 90%". Voici les étapes :

1. La caméra filme.
2. On transforme l'image pour que le modèle la comprenne.
3. Le modèle analyse et trouve les objets.
4. On affiche les cadres avec les noms des objets.

```
class Classifier {
  late Interpreter _interpreter;
  static const String modelFile = "detect.tflite";

  Map<String, dynamic>? predict(image_lib.Image image) {
    TensorImage input = TensorImage.fromImage(image);
    input = preprocess(input);
    _interpreter.run(input, outputs);
```

```
    return {"objects": recognitions};
  }
}
```

Listing 2 – Code de détection

0.3.2 Autres fonctionnalités

On a aussi :

- **Classification** : pour dire si une image montre un chat ou une voiture.
- **Analyse de texte** : pour savoir si un texte est positif ou négatif, comme dans une critique.

0.4 Comment on a rendu l'appli rapide

0.4.1 Faire vite sans bloquer

Pour éviter que l'appli rame, on a mis l'IA dans une tâche à part, appelée "isolate". Ça permet à l'écran de rester fluide même quand l'IA travaille dur.

0.4.2 Astuces techniques

- Réduit la taille des modèles pour qu'ils prennent moins de place.
- Utilisé une résolution plus basse pour la caméra.
- Gardé des données en mémoire pour aller plus vite.

0.5 L'interface utilisateur

L'appli est intuitive : vous l'ouvrez, et la caméra montre ce qu'elle voit avec des cadres autour des objets. Par exemple, pointez sur une table, et vous verrez "table 90%". Un panneau coulissant permet de changer de mode, comme passer à l'analyse de texte.

0.6 Gestion des pépins

On a ajouté des sécurités pour éviter les bugs. Par exemple, on vérifie si les entrées utilisateur sont correctes (comme un email valide). Si l'IA a un souci, on affiche un message clair.

```
void loadModel() async {
  try {
    _interpreter = await Interpreter.fromAsset("detect.tflite");
  } catch (e) {
    print("Oups, probl me avec le mod le : $e");
  }
}
```

Listing 3 – Gérer une erreur

0.7 Résultats

0.7.1 Performances

On a testé l'appli sur plusieurs téléphones, et voilà ce qu'on a obtenu :

Métrique	Détection	Classification	Sentiment
Temps (ms)	85-120	45-60	25-35
Précision	85-90%	92-95%	88-91%
Taille (MB)	5.1	4.3	0.6
Images/s	28-35	45-60	60+

TABLE 1 – Nos résultats

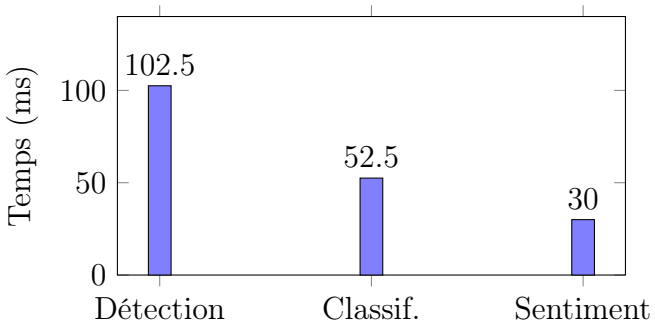

FIGURE 2 – Combien de temps prend chaque tâche

0.7.2 Pourquoi c'est génial

— Ça va super vite, même sur un téléphone pas trop puissant !

— Les données restent sur votre appareil, donc c'est sécurisé.

— Le code est bien organisé, prêt pour de futures améliorations.

0.8 Conclusion

On est vraiment contents de cette appli ! Elle détecte des objets en temps réel, analyse des images et des textes, tout en restant fluide et facile à utiliser. Ce projet nous a appris à jongler avec Flutter et l'IA, et on a hâte de voir jusqu'où on peut aller.

0.9 Références

— TensorFlow Lite : https://www.tensorflow.org/lite

— Flutter Documentation : https://docs.flutter.dev

— Clean Architecture - Robert C. Martin : https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html

— flutter_bloc package : https://pub.dev/packages/flutter_bloc