Rapport SAE 5.01

Rapport présenté par

MALORON Arthur,
DOTTO Matis,
MANICK Luc,
BELMADANI Abdourrahmane

Académie de Nancy-Metz
Université de Lorraine
Institut Universitaire Technologique de Metz
Département Informatique
Promotion 2022-2025

Table des matières

Introduction

De nos jours, l'intelligence artificielle évolue chaque jour, prenant de plus en plus d'espace dans nos vies. Que ce soit dans le domaine de la médecine, de la finance, ou de l'art, l'intelligence artificielle s'est déjà fait une place, pour le meilleur ou pour le pire.

Dans ce contexte, il est intéressant de comprendre le fonctionnement d'une intelligence artificielle, et c'est d'après nous la raison pour laquelle ce sujet nous a été donné.

Pour résumer le sujet, il nous a été demandé de créer une application de reconnaissance d'image intégrant un modèle d'intelligence artificielle.

L'application doit être capable de reconnaître les objets en temps réel tout en offrant la possibilité de prendre des photos et d'importer des images de la galerie de l'appareil pour les reconnaître.
Au niveau du modèle, nous devions prendre un modèle pré entraîner et le réentraîner (ou le fine-tuner) pour qu'il puisse être utilisé dans l'application.
Nous devions donc choisir un type d'objet que nous allions devoir faire détecter à notre modèle, et nous avons choisi une sélection de plats, afin de créer une sorte de shazam de l'assiette.
Certaines fonctionnalités supplémentaires peuvent être ajoutées, comme la possibilité de synchroniser ses images avec celles des autres utilisateurs.

Lexique

Tensor

Un tensor est, pour schématiser, un tableau multidimensionnel. Un tensor de forme (1, 3, 640, 640) peut donc être vu comme un tableau à 4 dimensions.

Dataset

Un dataset est un ensemble de données utilisées pour entraîner un modèle d'IA. Dans notre cas, il s'agit donc d'images accompagnées de labels pour définir quels objets sont détectés sur l'image ainsi que leurs coordonnées.

BoundingBox

Une boundingBox est un rectangle défini par 4 coordonnées (soit x1,x2,y1,y2 soit xCenter, yCenter, width, height) qui permet de mettre en évidence des zones d'intérêt, dans notre cas elles servent à mettre en évidence les objets détectés par le modèle.

Epoch

Un epoch est une métrique pour les modèles d'IA qui signifie que l'IA a effectué un parcours complet du dataset lors de son entraînement. Une IA avec 10 epochs a donc fait 10 parcours de son dataset lors de l'entraînement. Plus le nombre d'epochs est grand, plus le modèle se raffine et s'ajuste, devenant plus précis.

Technologie utilisés

Modèle d’intelligence artificielle

Yolo est un modèle d’intelligence artificielle conçu pour détecter des objets dans une image ou une vidéo Nous avons choisi YOLO V8 (You Only Look Once) comme modèle de reconnaissance d’objets.

Frameworks

LabelImg : C’est un outil open-source qui permet de créer les annotations des données.

Symfony : Ce framework a été utilisé pour structurer le backend et gérer les échanges de données entre l’application et le serveur. Il nous a permis de créer une API pour notre application.

TensorFlow : Nous avons utilisé TensorFlow, une bibliothèque open-source de machine learning largement reconnue, pour entraîner et optimiser notre modèle de reconnaissance d’objets.

Langages de programmation

Kotlin : Nous avons utilisé Kotlin pour développer l’application Android. C'est un langage de programmation orienté objet.

Environnement de développement mobile

Android Studio : C’est l’environnement de développement que nous avons utilisé pour créer l’application. Il nous a permis de coder, et tester notre application

Jetpack Compose: C’est un kit d'outils conçu pour simplifier le développement des interfaces utilisateur.

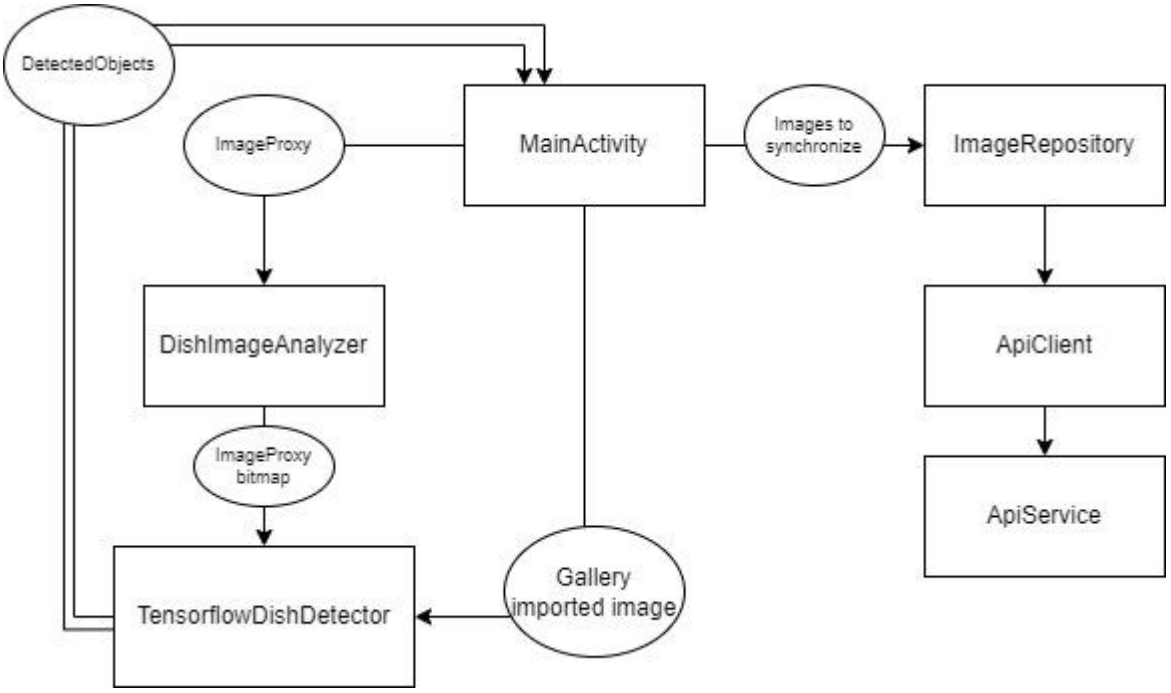
CameraX : Cet outil nous a permis d’intégrer les fonctionnalités de la caméra dans l’application, comme pour capturer des images.

Backend

Machine virtuelle (VM) : Nous avons configuré une machine virtuelle. Cela nous a permis d'héberger l’API. Ce qui nous permet d’avoir toutes les données en commun.

Architecture

Afin de comprendre l'architecture de notre application, référez vous au diagramme ci-dessous.



Comme on peut le voir, tout est centré autour du MainActivity, servant de page principale. Sur cette page, on retrouve toutes les fonctionnalités principales de l'application (à savoir détection en temps réel, prise de photo, synchronisation des images).

Nous allons donc rapidement nous pencher sur chacune de ces fonctionnalités.

Détection en temps réel

Le MainActivity utilise notre classe DishImageAnalyzer, une classe héritant de la classe ImageAnalyzer de la librairie CameraX.
Pour chaque frame, le DishImageAnalyzer récupère l'image capturée par l'appareil sous la forme d'un ImageProxy au format 640x480 (avec une rotation de 90°).
Cet ImageProxy est converti en bitmap et envoyé au tensorflowDishDetector pour que l'image soit analysée par le modèle.
Une fois l'image analysée par le modèle, le tensorflowDishDetector renvoie la liste des détections sous forme d'une liste de DetectedObject (classe contenant un name, une certainty, et une boundingBox), on peut alors dessiner les boundingBox sur l'image initiale à partir de ces détections.

Prise de Photo

La prise de photo avec incrustation d'overlay fonctionne en capturant une image à partir de la caméra et en y superposant des éléments graphiques représentant les objets détectés.

Lorsqu'on appuie sur le bouton de capture, la fonction capturePhotoWithOverlay est déclenchée. Elle crée un fichier d'image avec un nom unique et des métadonnées, puis capture la photo à l'aide du contrôleur de la caméra (LifecycleCameraController).

Une fois l'image capturée, elle est chargée sous forme de bitmap. La fonction createOverlayBitmap génère un deuxième bitmap avec des rectangles et des étiquettes représentant les objets détectés, en utilisant les dimensions de l'image capturée pour ajuster correctement l'échelle. Ces deux bitmaps (l'image capturée et l'overlay) sont ensuite combinés à l'aide de la fonction combineBitmap. L'image finale est sauvegardée à l'emplacement initial, écrasant la version sans overlay. Un message de succès s'affiche si tout se passe bien, sinon un message d'erreur est montré en cas de problème lors de la capture ou de la sauvegarde.

Analyse d’une photo provenant de la pellicule

La fonction chooseFromGalleryButton crée un bouton flottant circulaire qui, lorsqu'il est cliqué, appelle la fonction chooseImageInGallery. Ce bouton s'affiche en bas à droite de l'écran avec une icône d'ajout.

chooseImageInGallery ouvre l'application de galerie pour permettre à l'utilisateur de sélectionner une image. Une fois l'image choisie, l'application reçoit le résultat via onActivityResult.

Dans onActivityResult, si l'utilisateur a sélectionné une image avec succès, son URI est récupérée (imageUri). L'URI est ensuite convertie en Bitmap à l'aide de getBitmapFromUri.

L'image est analysée en utilisant un détecteur TensorFlow (TensorFlowDishDetector). Ce détecteur identifie les objets présents dans l'image et retourne une liste de détections avec leurs positions (box), noms (name) et niveaux de confiance (certainty).

Une copie du bitmap est créée et un Canvas est utilisé pour dessiner des cadres verts autour des objets détectés (drawRect). Le nom de chaque objet et sa précision sont également affichés en vert à côté du cadre (drawText).

L'image annotée est sauvegardée dans la galerie avec la fonction saveCombinedImageToGallery, qui crée un fichier dans le dossier "Pictures/S501" et y écrit l'image annotée. Si la sauvegarde réussit, un message de succès s'affiche.

Synchronisation

Les données d'une image sont stockées dans un objet de la classe Image, qui contient un identifiant (id), une URL et une liste de Category. La classe Category représente un objet détecté sur l'image et comprend un identifiant (id) ainsi qu'un label.

Lorsqu'une image doit être synchronisée, il est nécessaire de :

1. Créer un objet File (une classe Java native) à partir de l'URL de l'Image.
2. Créer un objet ImageCategory, une classe contenant l'identifiant de l'image (imageId) et une liste de Category.

Ensuite, le File et l'objet ImageCategory sont envoyés à l'ImageRepository, qui est chargé de gérer les requêtes vers l'API pour récupérer ou transmettre des données.

L'ImageRepository prépare une requête HTTP multipart, conçue pour transmettre à la fois des fichiers et des données. Pour effectuer les requêtes HTTP dans Android, nous utilisons la bibliothèque Retrofit.

La requête est transmise à l'ApiClient, une classe qui configure Retrofit pour établir une connexion avec l'API. Cette classe utilise l'interface ApiService, qui définit les différentes requêtes HTTP pouvant être effectuées sur l'API.

Enfin, la requête est traitée par l'API. L'image est correctement enregistrée et mise en ligne avec les autres images synchronisées par les utilisateurs.

API

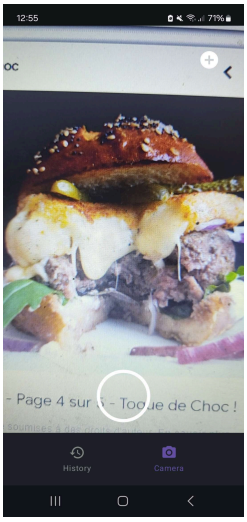
Pour héberger notre API, nous avons décidé d'utiliser une machine virtuelle, que nous exécutons à l'aide du logiciel VMware Workstation sur l'un de nos PC. Cette machine virtuelle utilise Debian, sur lequel nous avons installé un serveur Apache et une base de données MariaDB.

L'API est développée avec Symfony, une technologie adaptée à notre besoin. Elle contient un contrôleur avec des méthodes essentielles pour l'application. Ces méthodes permettent notamment de :

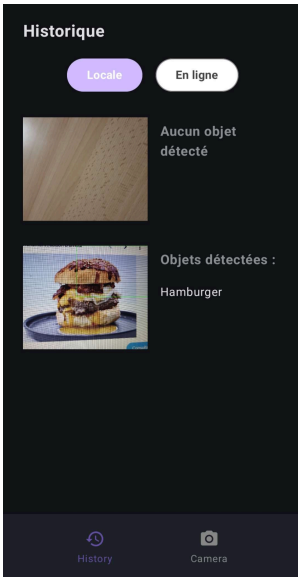
- récupérer toutes les images synchronisées en ligne
- enregistrer des images au format JPG dans un dossier de l'API
- ou supprimer une image

Présentation de l’application.

Pour l’application, nous avons opté pour un style ergonomique permettant une navigation facile pour n'importe quel utilisateur. Lorsque l’on arrive sur l’application, on atterrit sur la page de prise de photo avec un rond en bas pour prendre la photo, et la barre de navigation juste en dessous.



La barre de navigation est simple : deux boutons, un pour accéder à la page des historiques et un autre pour aller sur la page photo.
Lorsque l’on accède à la section Historique de l’application, deux onglets sont disponibles : Sur la page Locale, la liste des photos enregistrées localement s’affiche. Chaque photo est accompagnée de son descriptif. Si aucune détection n’a été effectuée pour une photo, le message « Aucun objet détecté. » est affiché.
Sur la page En ligne, les photos enregistrées sur le serveur sont listées de la même manière, avec leurs descriptions ou le message indiquant l’absence de détection.



Lorsque l’on clique sur une image, une page de détails s’affiche. Elle présente l’image en taille complète, la liste des objets détectés, ainsi qu’un bouton permettant de synchroniser ou de désynchroniser l’image en ligne.

Modèle d'IA

Le modèle d'IA que nous avons utilisé au cours de ce projet est un modèle YOLO (You Only Look Once) V8.

Yolo, dans son fonctionnement, divise l'image d'entrée en une grille de cellules qui vont permettre l'analyse de l'image.
La différence entre la taille de l'image et la taille de la grille de cellules est appelée le stride, un stride plus bas signifie plus de cellules pour la grille et donc une analyse plus précise, tandis qu'un stride plus haut signifie que le modèle peinera à détecter les plus petits objets. Dans notre cas, notre stride est de 32, ce qui est plutôt grand.

Lors de l'analyse de l'image, l'algorithme va analyser chaque cellule et lui assigner des coordonnées (pour entourer un objet potentiel) et un score de confiance pour chaque classe reconnue par le modèle.
Cela évite l'approche traditionnelle de commencer par analyser l'image pour définir des "régions d'intérêt" puis repasser sur ces régions d'intérêt pour trouver leur classe.

En effet, Yolo effectue toute son analyse en un seul parcours de l'image, ce qui est à l'origine de son nom (You Only Look Once), ce qui propose de gros gains de performances par rapport aux options traditionnelles.
Cependant, ce gain en vitesse n'est pas sans conséquences. En effet, si on compare Yolo avec un algorithme R-CNN (un autre algorithme de reconnaissance d'objet), on se rend compte que Yolo perd sur le champ de la précision.

Il n'existe pas d'outil parfait, et Yolo ne fait pas exception à la règle. R-CNN et d'autres alternatives restent jusqu'à maintenant plus efficaces au niveau de la précision, restent des outils de choix pour des applications n'ayant pas besoin d'être particulièrement réactives. Yolo reste cependant très adapté pour notre cas de figure, une application de reconnaissance devant reconnaître des images en temps réel, et donc être très réactive.

Optimisations

Afin d'améliorer l'apprentissage du modèle, nous avons mis en place quelques optimisations et avons fait quelques recherches à ce sujet que nous allons exposer ici.

Adaptation du learning rate

La première optimisation que nous avons mise en place est l'adaptation du learning rate.

Cette optimisation est simple, elle consiste à réduire le learning rate au fur et à mesure de l'apprentissage.
Pour comprendre l'impact de cette optimisation, utilisons cette analogie :
On peut voir l'apprentissage du modèle comme un curry dans lequel il faut mettre la bonne dose d'épices.
Si on met beaucoup d'épices à la fois, on finira vite, mais on risque d'avoir un curry trop épicé (ce qui n'existe pas d'après certains).
Si on ne met qu'une pincée d'épices à chaque fois, on trouvera le montant d'épices parfait, mais ça risque de prendre trop longtemps.

En commençant avec un gros learning rate et en le réduisant au fur et à mesure, on fait en sorte que le modèle apprenne rapidement mais un peu grossièrement au début, pour ensuite ajuster la précision sur la fin.

Cela permet non seulement de gagner en précision mais aussi en temps d'entraînement (gain de 20% de vitesse en moyenne). Cette optimisation nous a permis d'augmenter le nombre d'epochs, gagnant grandement en précision.

SGD et AdamW

Pour l'optimisation de l'entraînement, nous avons fait quelques recherches et avons pensé à utiliser deux d'entre eux.
Nous allons donc expliquer chacun d'eux et expliquer notre choix.

Commençons avec SGD.
SGD fonctionne en prenant des petits groupes de données du dataset à la fois pour chaque modification, ce qui rend la gestion de grands dataset beaucoup plus simple.
Les modifications étant plus fréquentes, SGD propose souvent une convergence plus rapide que la moyenne.

SGD est donc plus rapide lors de l'entraînement, mais il a aussi quelques défauts, en particulier son manque de précision ou sa tendance à overshoot.
Cet optimizer nécessite aussi une bonne connaissance des learning rate pour être correctement utilisé.

De son côté AdamW est un optimizer qui intègre trois principes fondamentaux : le momentum, le l'adaptation du learning rate et l'ajustement des poids.

Le momentum signifie que AdamW a connaissance de la direction générale dans laquelle va le modèle et prêtera moins d'attention aux données qui vont à l'encontre de cette direction. Si par exemple on lui donne une pomme et on lui dit que c'est une pizza, il le prend en compte, mais moins que les autres optimizers, car ce résultat ne va pas dans la direction de toutes les autres pizzas qu'il a vu jusqu'à maintenant.

L'adaptation du learning rate a été expliquée plus haut et a été intégrée à AdamW après qu'on ait commencé à l'utiliser.

L'ajustement des poids signifie qu'on essaie d'ajuster les poids au cours du parcours.
Le modèle va faire des prédictions et les comparer au résultat réel afin de définir quels poids (ou quelles images) ont le plus contribué aux erreurs, réduisant leur poids pour les futures détections. Cela permet de petit à petit permettre au modèle d'oublier les images qui l'auraient menés dans une mauvaise direction, augmentant la précision sur le long terme.

AdamW propose donc une meilleure précision et une bonne efficacité, malgré le temps perdu par l'ajustement des poids et le ralentissement parfois causé par le momentum.

Si on compare maintenant l'entraînement du modèle à descendre une montagne escarpée, SGD serait descendre à grands pas alors que la montagne et remplie de brouillard et qu'on ne voit qu'un petit espace autour de soi, tandis qu'AdamW serait comme descendre avec un compas qui nous garde dans la même direction, des chaussures qui réduisent les secousses du terrain, et un sac adaptatif qui permet de ne pas garder de poids inutile.

Finalement, nous avons décidé de choisir AdamW les raisons citées ci-dessus. Il semble plus adapté à nos besoins étant donné que notre dataset n'est pas assez grand pour justifier l'utilisation de SGD.

Difficultées rencontrées

Problèmes d'environnement

Afin de travailler avec des modèles d'IA complexes sans avoir à tout réinventer, nous avons dû utiliser de nombreux outils comme tensorflow ou pytorch.

Cependant, l'utilisation de ces outils n'a pas toujours été sans difficultés. En effet, nous avons eu du mal à faire fonctionner certains de ces outils, au point de devoir utiliser docker pour faire fonctionner nos outils.

Il ne serait pas très adapté de lister et expliquer nos problèmes d'environnement dans un rapport, mais nous avons pensé bon de le mentionner, étant donné que c'est un problème qui nous a suivis tout au long de ce projet.

Prise en main de Kotlin et des outils Android

Au cours de ce projet, nous sommes notés en partie sur le temps d'exécution de notre application. Nous avons donc pensé que réduire au maximum les couches d'interprétation serait bénéfique, ce qui implique de coder une application native.

Nous avons donc choisi de coder une application native Android en Kotlin, un langage originaire du Java qu'aucun de nous n'avait eu la chance d'utiliser jusque-là.
Le langage en lui-même n'était pas si compliqué à prendre en main, mais il nous a fallu un certain temps d'adaptation pour nous habituer aux différents outils impliqués dans l'application (Composer jetpack, CameraX, etc.).

Décoder les données de sortie de YOLO

L'une des premières difficultées que l'on a rencontré au cours de ce projet fut de décoder les données de sortie de notre modèle.
En effet, nous avons rapidement réussi à faire fonctionner le modèle YOLO sur android studio à l'aide de l'API Interpreter de tensorflow, mais la lecture des données était une toute autre histoire.

Le tensor d'output de notre modèle était (1, 14, 5376).
le premier nombre représente le nombre de batch, ou le nombre d'images qu'on a analysées, qui sera donc toujours de 1 dans notre cas.
Ensuite on trouve en seconde position le nombre de canaux, contenant les informations suivantes (4 coordonnées de bounding box, 1 donnée pour la confidence, et 10 données pour les probabilités de chaque classe, cela peut être plus si on a plus de classes).
Finalement, le dernier nombre est le nombre de prédictions pour chaque image. Une cellule de la grille a souvent plusieurs prédictions, c'est pourquoi le nombre est si grand.

Maintenant que l'on sait un peu plus comment fonctionne ces données de sortie, il est temps de les lire.

Dans ce code, modelOutputShape[2] est le nombre de détections et modelOutputShape[1] le nombre de cannaux.

```
private fun decodeYOLOOutput(array: FloatArray) : List<DetectedObject> {
    val detections = mutableListOf<DetectedObject>()

    for (i in 0 ≤ until < modelOutputShape!![2]) {
        var maxConf = precisionThreshold;
        var maxId = -1;
        var j = 4;
        var arrayId = i + modelOutputShape!![2] * j
        while (j < modelOutputShape!![1]){
            if (array[arrayId] > maxConf) {
                maxConf = array[arrayId]
                maxId = j - 4;
            }
            j+= 1;
            arrayId += modelOutputShape!![2];
        }
```

On parcourt chaque détection, créant pour chacune d'elle les variables maxConf et maxId qui vont nous permettre de savoir quelle est la classe avec la meilleure probabilité pour cette détection.

dans la formule d'arrayId, i représente l'id de la détection et (modelOutputShape[2] * j) représente l'écart nécessaire pour atteindre le prochain canal de la détection.
On parcourt ensuite toutes les probabilités des classes et on modifie la probabilité maxConf si on trouve une classe dépassant le palier minimum qu'on a placé.

Cette approche fonctionne car le tableau à 3 dimensions (1, 14, 5376) dans lequel on reçoit nos données peut être utilisé comme un tableau à 1 dimensions de taille 14 * 5376, c'est d'ailleurs comme cela qu'il est sauvegardé en mémoire.

```
if (maxConf > precisionThreshold) {
    val clsName = labels[maxId];
    val xCenter = array[i];
    val yCenter = array[i + modelOutputShape!![2]];

    val width = array[i + modelOutputShape!![2] * 2]
    val height = array[i + modelOutputShape!![2] * 3]

    val left = xCenter - (width/2F);
    val top = yCenter - (height/2F);
    val right = xCenter + (width/2F);
    val bottom = yCenter + (height/2F);
    if (left < 0F || left > 1F) continue;
    if (top < 0F || top > 1F) continue;
    if (right < 0F || right > 1F) continue;
    if (bottom < 0F || bottom > 1F) continue;

    detections.add(
        DetectedObject(
            name = clsName,
            certainty = maxConf,
            box = RectF(
                left,
                top,
                right,
                bottom
            )
```

Une fois qu'on a obtenu la classe ayant la meilleure probabilité pour cette détection, si sa probabilité (ou "confiance") est supérieure au palier qu'on a placé, on traite les données pour récupérer les coordonnées de la boundingBox ainsi que le nom de la classe avant d'ajouter la détection à la liste de toutes les détections initialisée au début de la fonction.

```
return applyNonMaxSuppression(detections).sortedByDescending { it.certainty }.take(maxResults);
```

Finalement, on applique l'algorithme de nonMaxSuppression aux détections et on retourne les meilleures détections triés par leur confiance.

IoU et NonMaxSuppression

Une fois que les détections sont formatées en une liste de DetectedObjects, on peut se rendre compte qu'on a des centaines de détection et qu'énormément d'entre elles se superposent.

C'est là qu'entre l'algorithme de NonMaxSuppression. En effet, il s'agit d'un algorithme qui va nous permettre de détecter et de supprimer les détections qui se superposent, afin qu'on ne détecte pas plusieurs fois le même objet au même endroit.

Pour cet algorithme, nous allons avoir besoin de l'IoU (Intersection Over Union), une métrique permettant de calculer la similarité entre deux boundingBox.
L'IoU peut être calculé avec la formule suivante :

$$Iou = \frac{Zone\ d'intersection}{Zone\ d'union}$$

```
private fun calculateIoU(box1: RectF, box2: RectF): Float {
    val intersectionLeft = maxOf(box1.left, box2.left)
    val intersectionTop = maxOf(box1.top, box2.top)
    val intersectionRight = minOf(box1.right, box2.right)
    val intersectionBottom = minOf(box1.bottom, box2.bottom)

    val intersectionWidth = maxOf( a: 0f,  b: intersectionRight - intersectionLeft)
    val intersectionHeight = maxOf( a: 0f,  b: intersectionBottom - intersectionTop)

    val intersectionArea = intersectionWidth * intersectionHeight
    val box1Area = (box1.right - box1.left) * (box1.bottom - box1.top)
    val box2Area = (box2.right - box2.left) * (box2.bottom - box2.top)

    val unionArea = box1Area + box2Area - intersectionArea
    return if (unionArea > 0) intersectionArea / unionArea else 0f
}
```

Au niveau de son implémentation, on commence par récupérer la zone d'intersection, qui est définie par la zone dans laquelle se trouve les deux boîtes, pour cela on prend donc les coordonnées les plus grandes dans les deux boîtes pour créer une grande boîte qui englobe les deux autres.
Une fois cela fait, on calcule la largeur et la hauteur de la zone d'intersection qu'on vient de créer. On a maintenant la zone d'intersection.

On calcule ensuite la zone de chaque boîte avec la formule :

$$Zone\ d'une\ boîte\ =\ (droite\ -\ gauche)\ \times\ (bas\ -\ haut)$$

Une fois la zone de chaque boîte calculée, on calcule la zone d'union en les additionnant et en retirant la zone d'intersection.

Finalement, on calcule l'IoU en divisant la zone d'intersection par la zone d'union comme vu précédemment.

Maintenant que l'on a vu comment fonctionnait l'IoU, on va pouvoir appliquer l'algorithme de nonMaxSuppression.

```
private fun applyNonMaxSuppression(detections: List<DetectedObject>) : MutableList<DetectedObject> {
    val sortedDetections = detections.sortedByDescending { it.certainty }.toMutableList()
    val filteredDetections = mutableListOf<DetectedObject>()

    while(sortedDetections.isNotEmpty()) {
        val first = sortedDetections.first();
        filteredDetections.add(first);
        sortedDetections.remove(first);

        val iterator = sortedDetections.iterator();
        while (iterator.hasNext()) {
            val nextDetection = iterator.next();
            val iou = calculateIoU(first.box, nextDetection.box);
            if (iou >= iouThreshold) {
                iterator.remove();
            }
        }
    }

    return filteredDetections;
}
```

Pour chaque détection, on va parcourir toutes les autres détections encore dans la liste et on va calculer leur IoU. Si il est en dessous du palier minimal iouThreshold, on supprime la détection de la liste. L'algorithme continue jusqu'à ce qu'il ne reste plus aucune détection non filtrée, supprimant toutes les boundingBox redondantes.

Dataset

Nous avons peiné à trouver un dataset pré fait adapté à nos besoins pour entraîner notre IA et avons donc décidé de prendre la base de données d'images Food 101, qui remplissait toutes nos demandes. Cependant, ces images ne sont pas labellisées, c'est-à-dire que les boundingBox n'ont pas été placées sur ces images, sans quoi le modèle n'a aucun moyen de savoir quelle image contient quels objets lors de l'entraînement.

Nous avons donc annoté manuellement les images afin d'y ajouter les bounding boxes nécessaires. Ce processus a consisté à identifier et à annoter précisément les objets présents dans chaque image. Cette étape est longue en termes de temps et d'efforts mais elle est essentielle pour permettre au modèle de reconnaître efficacement les objets.

Nous avons utilisé LabelImg pour faciliter la création des annotations. Grâce à cet outil, nous avons pu délimiter les zones correspondant aux catégories d'aliments choisies : hamburgers, salades Caesar, bœuf tartare, frites, glace, onion rings, pancakes, pizza, pâtes carbonara et steak.

Afin d'améliorer les performances du modèle, nous avons veillé à intégrer une grande variété d'images, en jouant sur les angles, l'éclairage, les arrière-plans. Cette diversité est essentielle pour permettre au modèle de mieux généraliser et de devenir plus performant face à des situations réelles. En effet, un modèle entraîné uniquement sur des images similaires aura des difficultés lorsqu'il sera confronté à des conditions légèrement différentes de celles rencontrées lors de l'entraînement.

- Les variations d'angles permettent au modèle de mieux identifier un objet sous différents points de vue:
- La diversité des éclairages aide le modèle à s'adapter aux changements de lumière ou de contraste.
- La diversité des arrière-plans et des contextes garantit que le modèle puisse détecter les objets même dans des environnements variés.

Cette phase de labellisation manuelle nous a permis de constituer un dataset adapté à notre cas d'utilisation, avec un total de 2982 images, chaque catégorie comptant au minimum 100 images, et certaines, comme les hamburgers, la salade Caesar, le bœuf tartare et la pizza en ayant jusqu'à 500.

ImageProxy CameraX

CameraX est la librairie android que nous avons utilisé pour récupérer le flux d'images de l'appareil, nous permettant d'ensuite envoyer ces images pour les analyser et les afficher avec les boundingBox.

Pour cela, nous avons créé une classe héritant d'ImageAnalysis.Analyzer, la classe DishImageAnalayzer.
Cette classe nous permet de récupérer et d'analyser les images chaque frame (dans notre cas nous avons mis en place une analyse toutes les 30 frames pour gagner en performances). Ce que l'on récupère dans notre imageAnalyzer est un ImageProxy, dans les dimensions 640x480.

Tout semblait parfait, mais notre modèle détectait rarement les bons objets et plaçait les boundingBox au mauvais endroit. C'était parce que l'ImageProxy qui nous était envoyé chaque frame avait subi une rotation à 90°, il fallait donc appliquer la rotation à l'ImageProxy pour retrouver une image avec la même rotation que l'image de base.

Lorsque nous utilisions un modèle SSD Mobilenet, nous avons aussi eu quelques problèmes à cause de cette rotation de l'ImageProxy. En effet, nous appliquions déjà une rotation à l'image avant de la passer au modèle, mais nous n'avions jamais pensé qu'il faudrait appliquer à nouveau la rotation aux résultats renvoyés par le modèle SSD Mobilenet. Il a aussi fallu inverser les coordonnées en y des données renvoyées par ce modèle, ce que l'on arrive pas vraiment à expliquer et qu'on a trouvé par hasard. C'est en partie à cause de toutes ces zones d'ombres qu'on a décidé de changer de modèle pour Yolo, gérant tout le traitement des données de notre côté avec l'API Interpreter pour avoir la main sur tout et pouvoir comprendre en détail comment fonctionne le modèle.

Ces problèmes peuvent sembler anodins, mais il nous a fallu du temps pour comprendre ce qui causait la mauvaise analyse du modèle. Nous n'étions pas sûrs non plus que le modèle était fiable étant donné que nous n'avions pas pu le tester en utilisant d'autres outils, ce qui nous a dispersé dans nos recherches.

Organisation

Cette partie vise à montrer notre organisation et la répartition du travail au cours de ce projet.

Labellisation des images :

- Luc
- Abdourrahmane
- Arthur

Implémentation et entraînement du modèle :

- Arthur

Prise de photo et importation depuis la galerie :

- Arthur
- Abdourrahmane

Historique, API et synchronisation :

- Matis

Conclusion

Pour revenir sur ce qui a été dit. Nous avons utilisé un modèle d'IA Yolo V8 ainsi que beaucoup d'autres outils Android pour réaliser une application de détection de plats fonctionnelle.

En plus des fonctionnalités actuelles, nous avons pensé aux potentielles évolutions futures que l'application pourrait avoir :

- L'application ne détecte actuellement qu'une petite sélection de plats (10 plats), mais avec un peu de temps l'application pourrait devenir un vrai shazam des plats, qui pourrait même être accompagné d'une description du plat lorsqu'il est détecté.
- L'application pourrait intégrer une fonctionnalité de tracking de calories en fonction des plats scannés, ce serait très approximatif mais ce serait intéressant de tenter d'implémenter cette fonctionnalité.
- Une dimension sociale pourrait être ajoutée en gardant une trace des plats que les utilisateurs synchronisent avec le serveur afin de pouvoir contacter celui qui a cuisiné le plat pour lui demander sa recette.

Au cours de la réalisation de cette application, nous avons dû apprendre le fonctionnement intrinsèque d'un modèle d'intelligence artificielle, comment les entraîner, et comment les optimiser au travers de cet entraînement pour obtenir les résultats attendus du projet.
Nous avons aussi découvert à quel point le monde de l'IA est vaste, découvrant des outils adaptés à des cas d'utilisation très précis.
Cela nous pousse à penser à ce que l'on pourrait réaliser en utilisant l'IA, et ces connaissances nous aideront assurément à réaliser ces potentiels projets.

Ce projet était très enrichissant pour nous, et pour cela, nous sommes reconnaissants envers nos professeurs, M.Jozefowiez et Mme.Boudjeloud-Assala, pour nous avoir donné ce sujet.