Polytech Tours

# Application d'assistance pour les malvoyants

Projet IA – S9

**Lien Github :** https://github.com/nekosse/IAMalvoyant.git

Yoann Dupas & Léo Pinot
12/01/2021

# Table des matières

# 1 Introduction

Ce rapport présente nos travaux réalisés au semestre 9 pour le projet d'assistance aux malvoyants. Ce sujet a été proposé par Gilles Venturini. Le but est donc de développer une application smartphone, afin d'éviter tout appareils tiers, capable de reconnaitre son environnement. Il existe de nombreuses technologies de reconnaissance d'image, chacune ayant ses spécificités, ses avantages et ses inconvénients. Pour ce projet, nous utiliserons le « deep learning », une branche de l'intelligence artificielle qui fait beaucoup parler d'elle dans le domaine de la reconnaissance d'image.

Le projet consiste donc à réaliser, une collecte de donnée, plus précisément des images labélisées, pour entrainer un réseau de neurones. Ce réseau pourra classifier des images en fonction des scènes et des objets qu'il aura détecté. Une fois entraîné, il sera intégré à une application sur smartphone qui avertit l'utilisateur lorsque celui-ci détecte certaines scènes ou objets.

L'année dernière au semestre 10, un projet libre a été effectué par des étudiants de Polytech Tours sur le domaine de l'IA pour les malvoyants. Ils ont développé une application capable de reconnaitre nos objets du quotidien grâce à un réseau de neurone à convolution (VGG16) déployé sur smartphone. Nous avons utilisé leurs travaux comme base pour notre projet.

Dans ce rapport nous détaillerons, dans un premier temps, les besoins et les objectifs du projet, les ressources à notre disposition et la démarche que nous avons suivi pour la réalisation. Dans un deuxième temps, nous présenterons les éléments qui nous ont permis de réaliser l'application final, la collecte de donnée, le réseau de neurone et l'application smartphone. Enfin, nous présenterons les résultats que nous avons obtenus pour chacun de ces différents éléments.

# 2 Description du projet

Nous verrons dans cette partie les différentes problématiques aborder par le sujet, les objectifs que nous ne sommes fixées, les ressources à notre disposition et la démarche final de notre projet.

## 2.1 Besoin

Nous avons identifié un besoin de la part de notre encadrant. Il souhaite une solution facilement utilisable et qui ne nécessite pas de coûts supplémentaires afin d'avertir et d'indiquer aux personnes malvoyantes les dangers et les obstacles qu'ils peuvent traverser dans la vie du quotidien. Il nous a imposé l'utilisation du deep learning pour la reconnaissance d'image, le reste du projet est libre.

## 2.2 Objectif

L'objectif du projet consistera à réaliser une campagne de collecte de donnée afin d'entrainer un réseau de neurones et de l'implémenter sur un smartphone puis d'évaluer les forces et faiblesses de la méthode. Ce sujet a pour but d'être traité par un doctorant dans le cadre d'une thèse, l'objectif principal est donc de tester et d'explorer la démarche afin de pouvoir soulever certains axes et d'en tirer des conclusions. Cela permettra d'avoir un premier prototype entièrement réutilisable (possibilité de réentraîner et réimplémenter le réseau de neurones facilement) pour une reprise future.

## 2.3 Solution

La solution que nous avons décidé de développer est une application Android qui permet de faire de la détection d'objets ou de scènes (escalier, portes, ascenseur et prise).

## 2.4 Démarche

Comme énoncé dans l'objectif, pour pouvoir réaliser l'application nous avons mis en place la démarche suivante :

- **Collecte de données :** Cette étape consiste à étudier d'éventuelle base de données existante et dans le cas contraire constituer notre propre base de données nécessaire pour l'entraînement du réseau de neurones.

- **Réseau de neurone pour la classification :** Cette étape consiste à réaliser un réseau de neurone pour la classification d'image. Il comprend donc son développement, sa paramétrisation et son entrainement.

- **Application Android :** Cette étape consiste à créer le support de réseau de neurone et l'interface entre l'utilisateur et l'application.

## 2.5 Ressources

Nous avons à notre disposition une application Android réalisé par des anciens étudiants. Le projet étant similaire au notre, il nous a servie de base pour l'application. Nous avons ensuite procédé à des modifications pour l'adapter à notre besoin.

# 3 Solutions

Cette partie décrit les recherches et les solutions que nous avons mis en place afin de mener à bien notre projet. Nous verrons pour chaque étape de notre démarche les problématiques, les axes que nous avons choisis pour y répondre et la manière dont nous les avons implémentés.

## 3.1 Collecte d'images

Collecter des images à grande échelle est une problématique plus complexe qu'il n'en parait. Il faut savoir à qui on peut s'adresser, comment les convaincre de collaborer à notre projet, quel support utiliser pour le dépôt des images mais aussi définir notre besoin, notre demande. Nous verrons dans cette partie notre démarche pour notre collecte d'image.


*Figure 1 : Etapes pour la collecte de données*

### 3.1.1 Polytech une communauté idéale

Selon nous le meilleur moyen de trouver un groupe de personnes actif, volontaire, suffisamment grand et facile à contacter, a été de faire appel à l'ensemble des élèves de Polytech Tours (toutes spécialités confondues). On a un contact facile avec eux via les listes étudiantes de la boite mail de l'université. Ils sont actifs par mail et recevront très certainement notre message. Ils sont curieux d'aider à des projets pour une cause honorable. Ils ont ou auront aussi à effectuer des projets libres, ils peuvent sans doute comprendre plus facilement notre démarche.

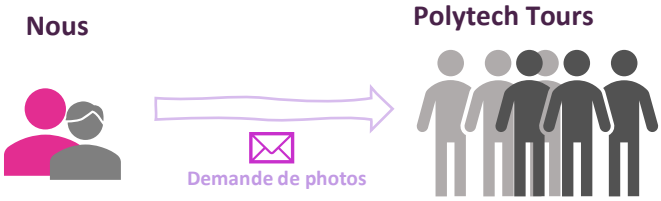



*Figure 2 : Polytech Tours, une communauté idéale*

### 3.1.2 Impossible de résister devant un Pitaya

Prendre une ou plusieurs photos et les envoyer sont des actions qui nécessite un peu de temps, et comme un travail n'est jamais gratuit, nous avons décidés d'inciter les gens à poster des photos en leurs proposant une récompense selon le principe : « plus vous postez, plus vous avez des chances de gagner la récompense ».

Notre choix a été d'offrir un repas livré à domicile chez la personne tirée au sort, le fait que le groupe de personne ciblé soit Polytech facilitera l'acheminement de la récompense vers la personne gagnante. Cette démarche présente deux avantages. On peut s'attendre qu'un maximum de personne répondent à notre sondage. Et les personnes motivées auront plus d'intérêt à poster un grand nombre de photos.


*Figure 3 : Equation illustrative de notre pensée*

### 3.1.3 Google Forms et ses possibilités

La plateforme de dépôt est sans doute la problématique la plus embêtante. Il faut pouvoir mettre en place simplement et sans coûts un serveur sur lesquels les gens pourront facilement déposer des photos prises directement depuis leurs smartphones.

Notre première idée a été de mettre en place un serveur web hébergé compatible smartphone et ergonomique sur lequel les gens pourront déposer leurs images via un déclenchement automatique du smartphone. C'est techniquement est faisable mais cette méthode nécessite un temps de développement plutôt conséquent par rapport au temps que nous avons à disposition. Elle demande aussi de trouver un hébergement pour y stocker les données.

Nous avons finalement opté pour Google Forms. La plateforme permet de déposer des images par les utilisateurs directement sur le cloud gérer par le service Google Drive. De plus, il est facile de générer un fichier CSV qui recense automatiquement les images et leur labels associés. Cela va permettre de traiter les données beaucoup plus facilement.

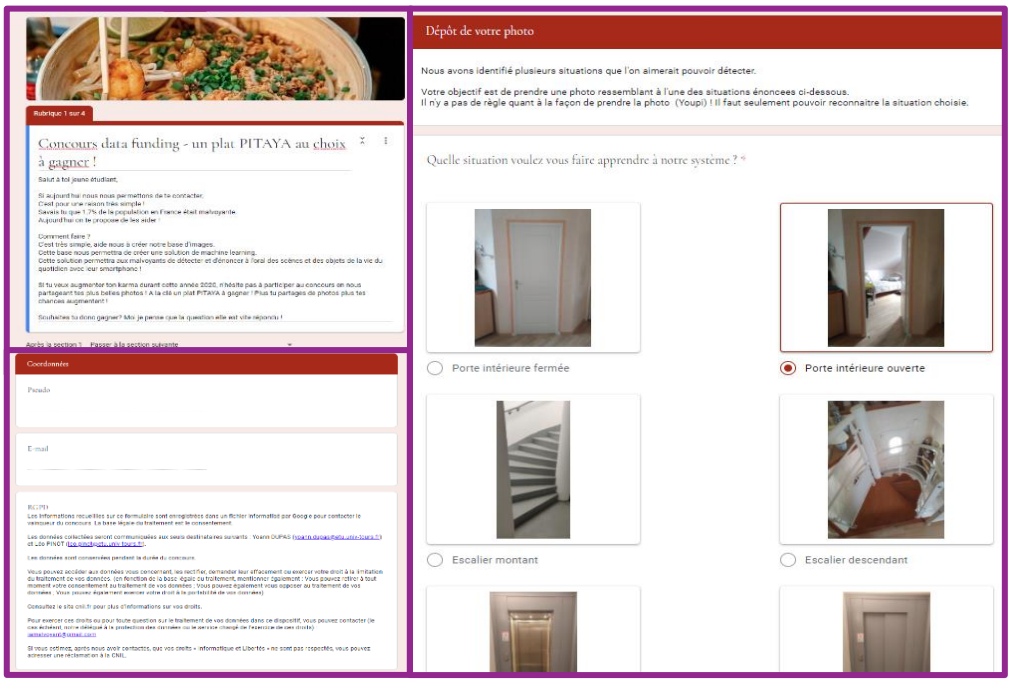



*Figure 4 : Extrait du sondage*

### 3.1.4 Script de traitement des données

En cas d'un grand nombre de données, télécharger les images une à une sur le drive de dépôt et les organiser correctement afin qu'elles soient prêtes à être utiliser pour l'apprentissage peut vite devenir

un travail fastidieux. C'est pourquoi nous avons mis en place un script python qui répond à ces problématiques.

Le script fonction de la manière suivante. L'utilisateur devra choisir un dossier de destination, valide l'opération et le script effectue les actions suivantes :

- Téléchargement des images sur le drive ainsi que leurs labels
- Création d'un dossier par label
- Attribution de noms uniques aux images
- Répartition des images dans leur dossiers respectifs
- Affiche la progression

Le code du script est disponible sur le git du projet (tools/ImportImage.py).

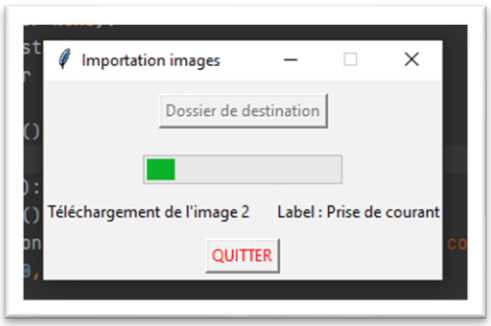



*Figure 5 : Interface du script*

Toute ces réflexions pour la récolte des données ont pour but d'amasser le maximum d'information afin de maximiser l'efficacité de l'apprentissage. Les résultats de cette expérience sont présentés en partie 4.1.

## 3.2 Modèle d'apprentissage profond

Notre base d'image étant complète, nous devons mettre en place notre réseau de neurones. Notre réseau de neurone doit être capable de classifier des images en détectant des objets ou des scènes. Nous verrons dans cette partie les différents choix qui nous ont permis d'arriver avec un modèle exploitable pour notre application.

### 3.2.1 Framework

Avant de commencer la recherche de modèle dans la littérature, nous avons étudié les compatibilités des différents Framework pour notre projet. Pour rappel, nous avons comme contrainte de devoir embarquer le modèle dans une application Android.

Notre choix s'est donc porté sur les Frameworks TensorFlow et TensorFlow Lite. En effet, TensorFlow Lite est un Framework dédié aux déploiements de model de réseau de neurone profond sur mobiles. L'avantage est qu'il nous permet de passer facilement d'un modèle développé sur machine vers un modèle compatible mobiles grâce aux outils de conversion. L'application pourra ainsi bénéficier d'un modèle prêt à l'emploi.

### 3.2.2 Modèle de classification d'image

Le Framework choisi, nous devons choisir l'architecture de notre réseau de neurone. Pour rappel, notre souhait est de développé un réseau de neurone dédié à la classification d'image multi labels. Nous avons aussi décidé pour simplifier la tâche, qu'une image est associée à un label.

Dans la littérature, il existe de nombreux modèles permettant de faire de la classification d'image. Le plus connu est le modèle VGG16. Il est capable de classifier des photos sur 1001 labels comme par exemple un porte avion, un chat, du fromage ou encore un panda roux. Notre piste c'est donc porté dans un premier temps sur ce modèle.

#### 3.2.2.1 VGG16

VGG16 est un réseau de neurone à convolution. En réalité, il ne s'occupe pas de la classification mais de réduire la dimensionnalité élevée d'une image en un vecteur de taille relativement faible. Ce vecteur résume les caractéristiques de l'image. On peut donc dire que VGG16 est un extracteur de caractéristique.

#### 3.2.2.2 Architecture d'un classifieur

Avec le vecteur, on peut ensuite utiliser un réseau de neurone classique (Perceptron) permettant de faire d'établir un score, une prédiction d'appartenance de l'image à un label. La Figure 6 illustre l'architecture d'un classifieur, ici multi labels.

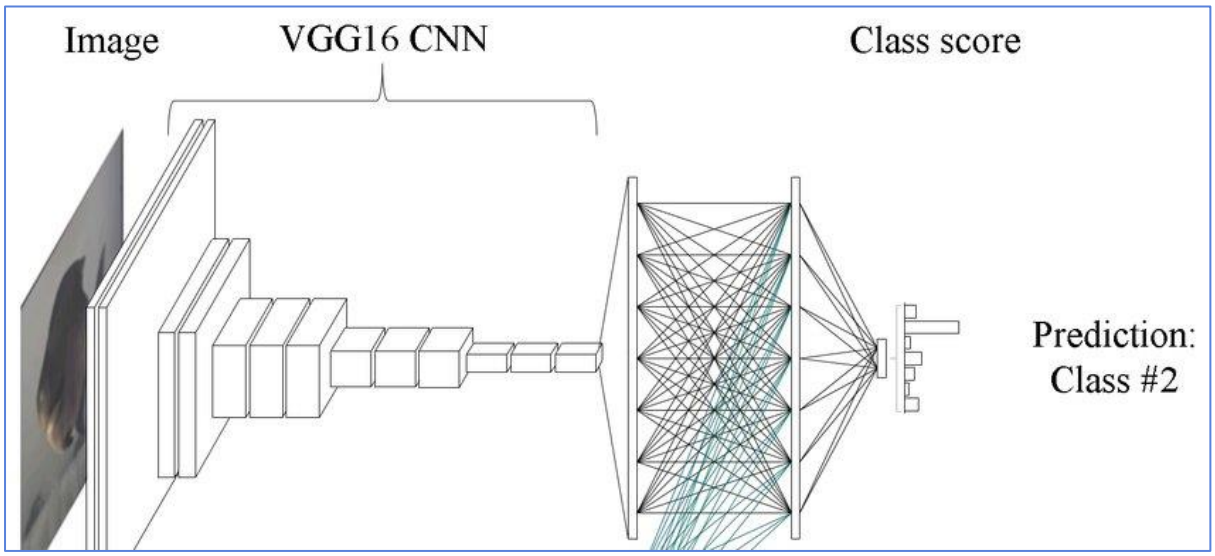


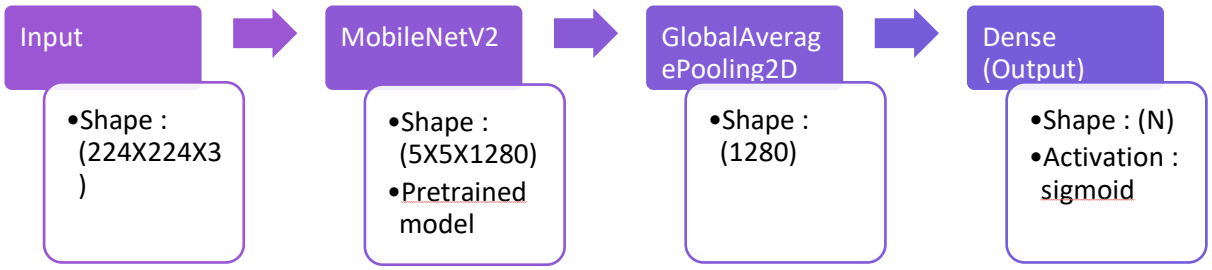

*Figure 6 : Architecture d'un classifieur avec VGG16 comme extracteur de caractéristique*

#### 3.2.2.3 MobileNetV2

Il existe aujourd'hui de nombreux modèles de réseau de neurone permettant de faire de l'extraction de caractéristique sur des images. Elles utilisent souvent des couches de neurone à convolution. Dans notre projet, nous avons décidé de ne pas utiliser VGG16 mais MobileNetV2. C'est un réseau de neurone qui à l'avantage d'être plus léger en nombre de paramètre que VGG6 et plus rapide. De plus, étant développé par le même auteur que TensorFlow, son utilisation sera amplement facilitée.

### 3.2.3 Architecture

Notre modèle final se composera un extracteur de caractéristiques et d'un classifieur. L'entrée de notre réseau est une image et la sortie un vecteur de taille N. N correspond aux nombres de labels que nous avons défini pour notre projet.



Nous avons donc une image en entrée de taille 224 par 224 en couleur RGB (c'est-à-dire 3 channels). L'image d'entrée étant souvent plus grande, un prétraitement des images est effectué pour que l'entrée soit transformée à la bonne taille. Ensuite, l'entrée est redirigée vers notre extracteur de caractéristique, MobileNetV2. On obtient en sortie un tenseur de dimension $(5 \times 5 \times 1280)$. Pour réduire encore ce tenseur, on applique ensuite une couche « GlobalAveragePooling2D ». Cette couche, comme son nom l'indique, extrait la moyenne sur certaines dimensions du tenseur. On obtient donc un vecteur de taille 1280. La dernière étape consiste à envoyer notre vecteur dans notre classifieur de taille N. Les neurones de notre classifieur possède la fonction d'activation « sigmoïde » pour transformer les sorties en valeur comprise entre 0 et 1.

### 3.2.4 Transfer Learning

Notre architecture est prête mais pour l'entraînement d'un réseau de neurone pour la classification d'image demande une quantité très importante d'image labelisées notre base d'apprentissage. Même si nous espérons une quantité d'image assez importante avec notre collecte d'image, la quantité sera toujours insuffisante. Pour données un ordre de grandeur, la base d'image COCO est composé de plus de 200 000 images labélisées. Il est donc impensable d'obtenir un modèle performant avec un entraînement de zéro.

Pour faire face au problème, nous allons utiliser la méthode dite de « transfer learning » (Figure 7). Cette méthode consiste à récupérer une partie d'un réseau de neurone déjà entraîné ($A$) pour une autre tâche similaire ($T_A$) à la nôtre avec une base générique ($D_A$) plus conséquente (par exemple la base de données COCO). Cette partie entraînée est ensuite injecté dans notre réseau ($A'$), puis compléter par d'autres parties spécifique ($B$) à notre tâche ($T_B$). On entraine ensuite notre nouveau modèle avec notre base spécifique ($D_B$) en prenant soin de « geler » les poids entrainés de notre partie récupérée.

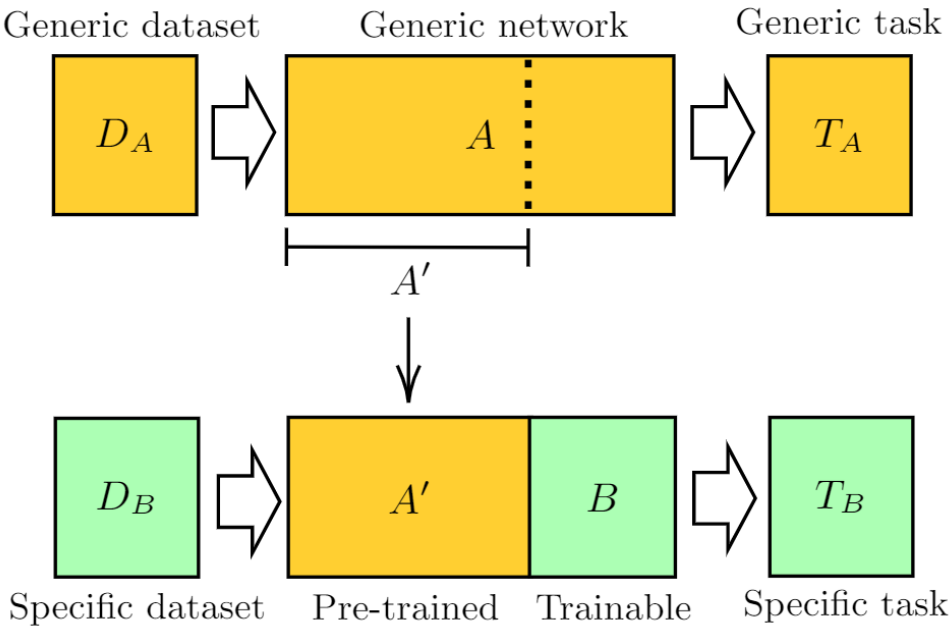



*Figure 7 : Illustration du transfer learning*

On peut ensuite ajuster notre modèle en entrainant sur une petite partie de notre base d'apprentissage (10 % environs). Cette étape est appelée le « fine tuning »

### 3.2.5 Paramètre d'apprentissage

Pour notre apprentissage, nous avons défini les paramètres suivants :

| | |
|---|---|
| **Loss Function** | Categorical Cross-Entropy |
| **Initial epochs** | 32 |
| **Fine tuning epochs** | 8 |
| **Optimizer** | Adam |
| **Image shape** | (224,224,3) |
| **Learning Rate – Transfer Learning** | 0.0001 |
| **Learning Rate – Fine Tuning** | 0.00001 |
| **Fine Tuning at layer** | 100 |
| **Batch Size** | 4 |
| **Metric** | Accuracy |

### 3.2.6 Convertissement et Quantification

#### *3.2.6.1 Convertissement*

Une fois notre modèle entraîné nous devons ensuite le convertir vers un format compatible avec le Framework TensorFlow Lite sur Android. Pour cela, il suffit de convertir le modèle en format «.tflite ». Le fichier conserve aussi les poids d'entraînement.

#### *3.2.6.2 Quantification*

TensorFlow propose aussi de quantifier le modèle afin de réduire la taille et améliorer la rapidité au détriment d'une faible perte de précision. Il existe plusieurs types de quantifications.

| Technique | Benefits | Hardware |
|---|---|---|
| **Dynamic range quantization** | 4x smaller, 2x-3x speedup | CPU |
| **Full integer quantization** | 4x smaller, 3x+ speedup | CPU, Edge, TPU, Microcontrollers |
| **Float16 quantization** | 2x smaller, GPU acceleration | CPU, GPU |

Le choix de la quantification se fait en suivant les indications données sur le site de TensorFlow. Nous avons opté pour la quantification dynamique et la quantification Float16. Cependant, nous n'avons pas trouvé de différence lors des essaies sur l'application. La quantification entière complète est quant à elle impossible car certaines opérations de notre modèle ne sont pas supportées.

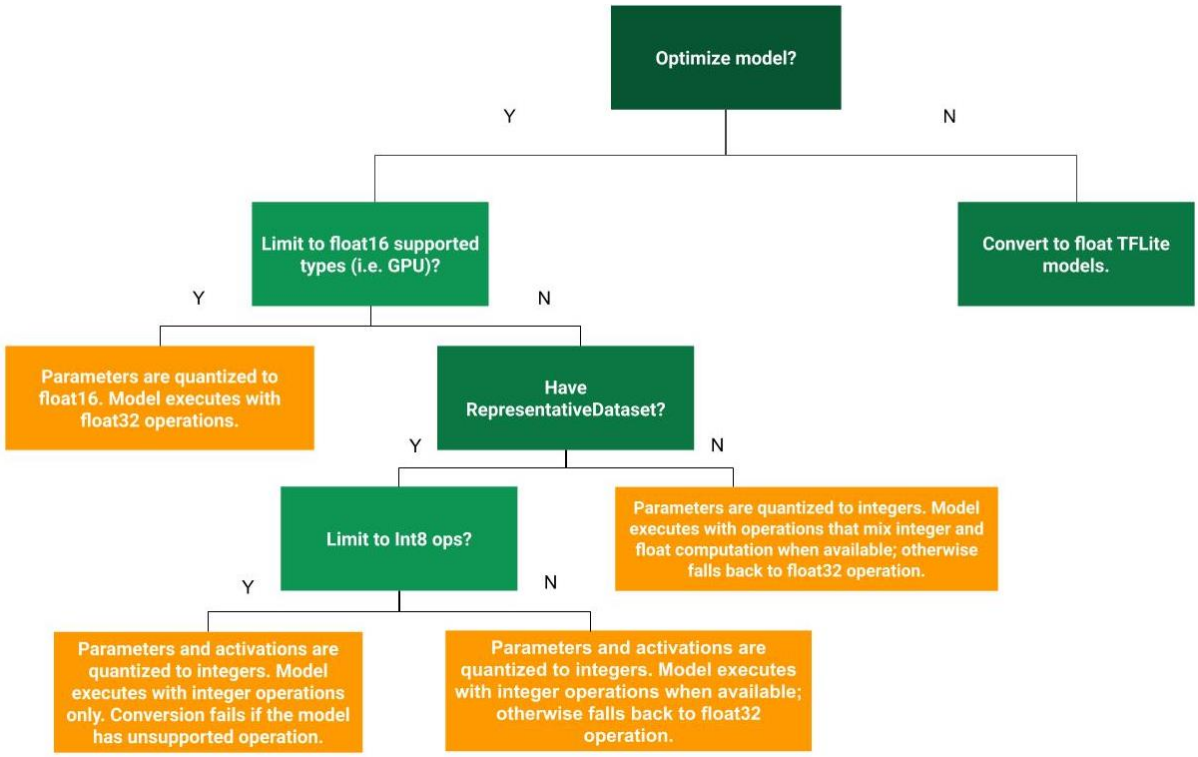



*Figure 8 : Arbre de décision pour le choix de la quantification*

## 3.3 Application Android

Dans cette partie, nous présentons les fondements de l'application Android. Il faut noter qu'elle est basée sur notre ressource, c'est-à-dire l'application Android de détection d'objet personnelle du quotidien, qui est elle-même basée sur les exemples du site de TensorFlow.

### 3.3.1 Librairie TensorFlow Lite

TensorFlow Lite est un portage du Framework TensorFlow sur mobile. Cette librairie permet de faire tourner des modèles de réseau de neurone profond. On peut s'en servir grâce à des instances de la classe nommée **Interpreter** qui permet d'utiliser nos modèles développés durant notre projet.

### 3.3.2 L'interface

Nous souhaitons avant tout tester l'efficacité du modèle. Nous avons donc opté pour l'interface la plus simple et ergonomique possible.

Notre vue possédera donc seulement l'image que filme la caméra. Par-dessus on trouve une zone de texte. Cette zone indiquera « Aucun élément détecté » en rouge si l'on ne reconnait rien. Sinon cette zone indiquera le label de la scène identifiée en vert.

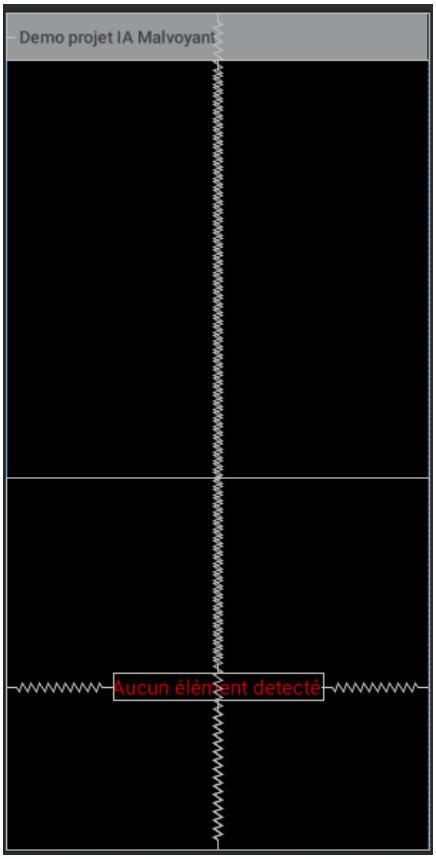



*Figure 9 : Illustration de l'interface de l'application*

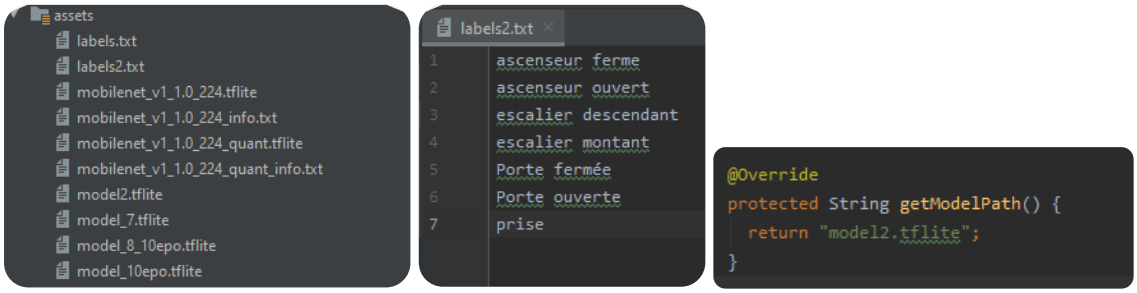
### 3.3.3 Les composantes principales de l'application

Cette application possède un certain nombre de classe mais seules certaines d'entre elles sont importantes pour comprendre le fonctionnement de l'application.

#### *3.3.3.1 ClassifierFloatMobileNet*

Cette classe va permettre de paramétrer le modèle utilisé (.tflite).

➔ Paramétrer le modèle




Ici on spécifie le chemin vers le modèle, si l'on souhaite appliquer un autre modèle (qui est généré de la même manière que le nôtre), il faut le glisser dans le dossier **assets** et changer cette ligne. Si nécessaire, on peut changer les labels dans **label2.txt.**

➔ Adaptation de l'image

```
@Override
protected void addPixelValue(int pixelValue) {
  imgData.putFloat((((pixelValue >> 16) & 0xFF)));
  imgData.putFloat((((pixelValue >> 8) & 0xFF)));
  imgData.putFloat(((pixelValue & 0xFF)));
}
```

La fonction **AddpixelValue** transforme l'image (npfloat) en image 8 bit conformément à notre modèle de réseau de neurones (quantifié).

➔ Lancement du classifieur

```
@Override
protected void runInference() { tflite.run(imgData, labelProbArray); }
```

Ici, on fait tourner le modèle grâce à la fonction tflite.run() de la librairie. Il met à jour le **labelProbArray** : Stucture de données composées de l'ensemble des classes et de leurs probabilités de correspondances.

➔ Récupération de la sortie

```
@Override
protected float getNormalizedProbability(int labelIndex) { return labelProbArray[0][labelIndex]; }
```

Récupération du vecteur de sortie du classifieur.

### *3.3.3.2 CameraActivity*

Cette classe gère tous les composants graphiques de l'application ainsi que leurs interactions

➔ Lancement du classifieur

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
  Log.i(this.getLocalClassName(), msg: "called onOptionsItemSelected; selected item: " + item);

  if (item == mSaveImage) {
    mViewMode = VIEW_SAVE_IMAGE;
  }

  if (item == mclassify) {
      mViewMode = VIEW_CLASSIFY;
  }

  if (item == mnothing) {
      mViewMode = VIEW_NOTHING;
  }
```

C'est ici que l'on va pouvoir lancer notre classifieur.

➔ Mis à jour de l'affichage

```
@UiThread
protected void showResultsInLabelTextViewRed(String text) {
  labelTextView.setTextColor(Color.parseColor( colorString: "#C32F18"));
  labelTextView.setText(text);
}
```

Cette fonction va permettre d'afficher à l'écran (ici en rouge) à quoi correspond la scène filmée.

### *3.3.3.3 ClassifierActivity*

Cette activité va faire le lien entre le modèle **tflite** et nos composants. La sous-fonction qui nous intéresse (processImage() → mViewMode==VIEW_CLASSIFY ) tourne en arrière-plan dans un **thread** afin que l'algo puisse être exécuté en continue.

On a donc :

➔ Appel du modèle

```
final List<Classifier.Recognition> results = classifier.recognizeImage(croppedBitmap);
```

Cette fonction va mettre à jour le vecteur de sortie du classifieur selon les fonctions vues dans la classe **ClassifierFloatMobileNet.**

➔ Reconnaissance d'une scène (toute les secondes)

```
if(results.get(0).getConfidence() > SEUIL_DE_DETECTION && !results.get(0).getTitle().equals(resPrec))
  resPrec = results.get(0).getTitle();
  showResultsInLabelTextViewGreen(resPrec);
  speaker.speak(resPrec);
}
```

Si le Label 0 dépasse le seuil de détection et que l'objet affiché par l'application n'est pas celui qu'on détecte actuellement alors on affiche en vert le nom du label trouvé et on fait parler la petite voix.

➔ Réinitialisation du label (toute les 3 secondes)

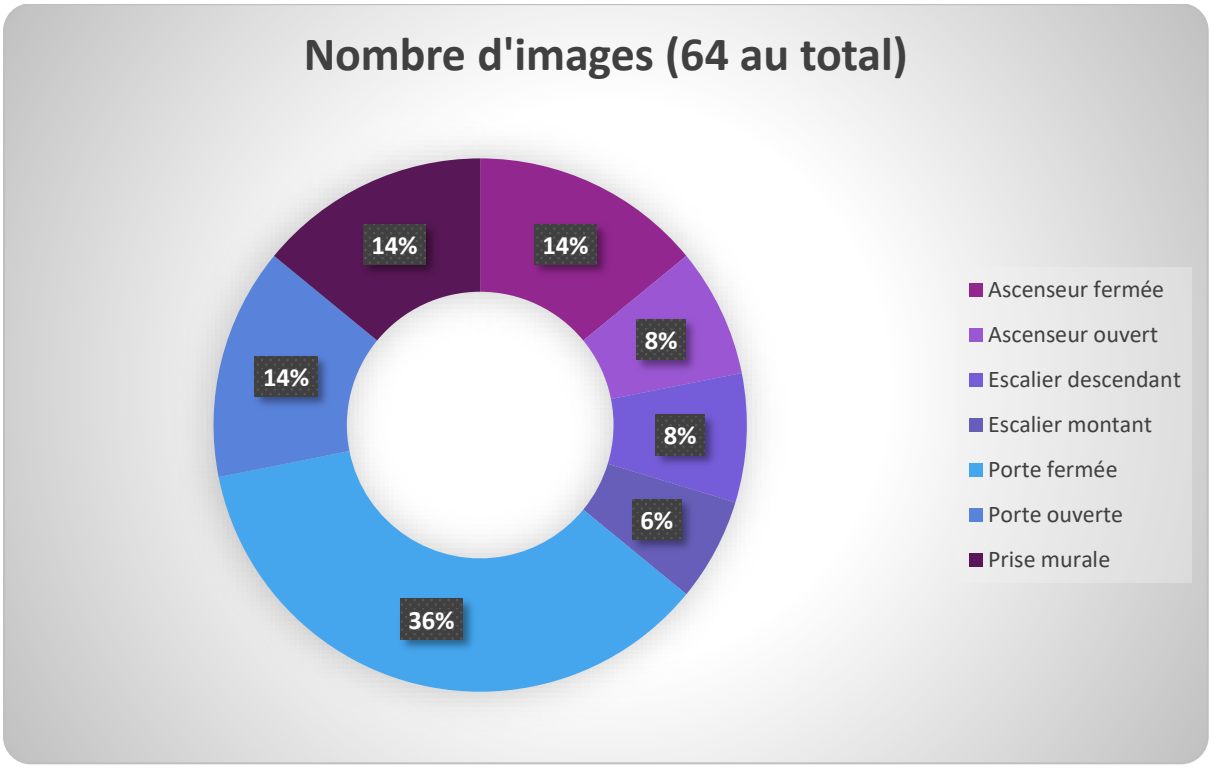
```
if(diffTimeSpeak > 3000L){
  if(results.get(0).getConfidence() <= SEUIL_DE_DETECTION) {
    lastRecoTime = currentTime;
    resPrec = "Aucun élément détecté";
    showResultsInLabelTextViewRed(resPrec);
  }
```

Toute les 3 secondes, si on ne détecte plus rien, on affiche en rouge « aucun élément détecté ».

# 4 Résultats

## 4.1 Collecte d'images

Les labels demandés lors de la collecte ont été : ascenseur fermé, ascenseur ouvert, porte fermée, porte ouverte, escalier montant, escalier descendant, prise murale.

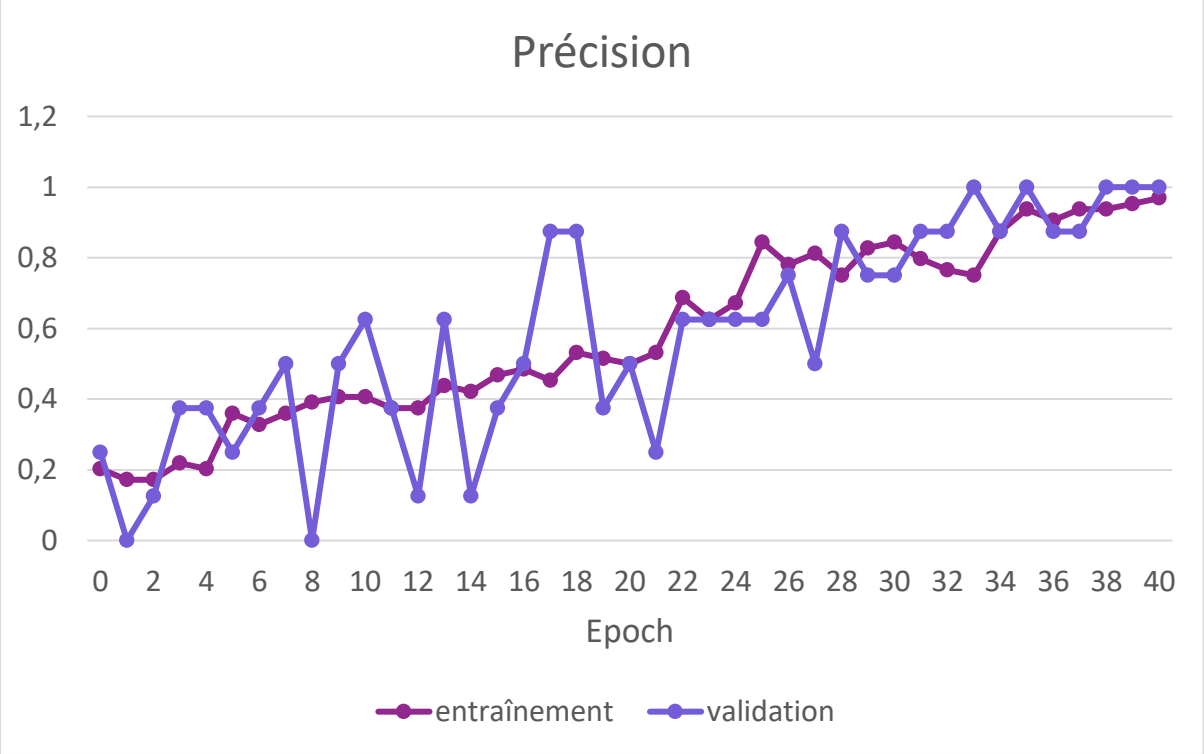


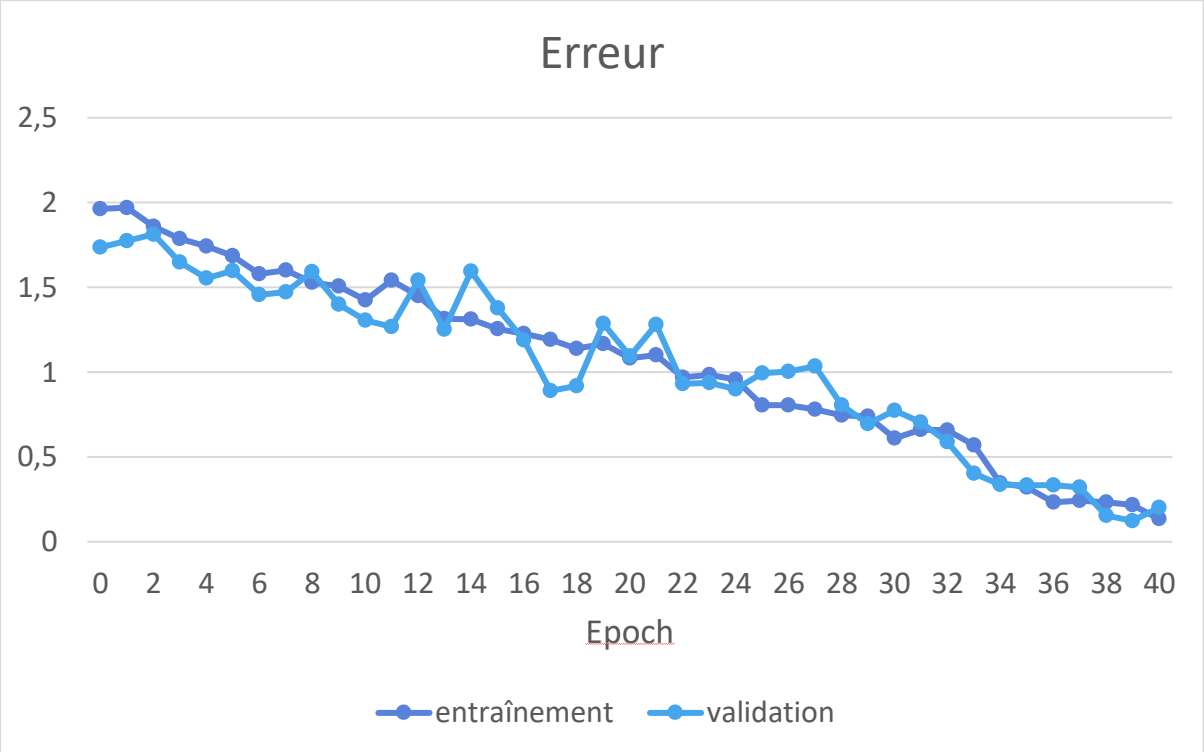

Nous avons obtenu en tout **64 images** avec en majorité des portes fermés, ce jeu de donnée est très petit mais nous avons gardé cet échantillon dans notre apprentissage afin des respecter notre démarche et de tester l'efficacité du Transfert Learning pour de petits échantillons.

Nous avons eu au total **16 participants** sur le formulaire. 4 d'entre eux ont postés plus d'une image. De plus nous avons eu **4 réponses invalides** (petits blagueurs). Un autre donateur extérieur nous a fourni **31 images supplémentaires**.

## 4.2 Modèle d'apprentissage profond

Dans notre base de données, 56 images ont servi pour l'entraînement, et 8 pour la validation. Voici les résultats de performances obtenues.


Précision
1,2
1
0,8
0,6
0,4
0,2
0
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40
Epoch
entraînement
validation


Erreur
2,5
2
1,5
1
0,5
0
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40
Epoch
entraînement
validation

## 4.3 Application Android




L'application donne des résultats plutôt satisfaisant bien que la taille du jeu donnée soit faible. Quand on filme des scènes n'appartenant à aucun de nos labels, le classifieur fonctionne bien et ne se trompe que très rarement. Cependant, pour certains labels tels que **l'ascenseur fermé** ou **les portes ouvertes**, il ne les reconnait pas ou pas immédiatement, évidemment cela peut être expliqué par le manque de données d'apprentissage pour ces labels. Dans ce cas de figure le seuil de reconnaissance est paramétré à 6%, cela peut paraitre faible mais ça fonctionne étonnement bien.

# 5 Conclusion

Ce projet nous a permis d'apprendre et de comprendre le développement d'une solution implémentant des solutions d'intelligence artificielle de A à Z. Nous avons compris que chacune des étapes a son importance pour que le résultat final puisse aboutir. De plus, bien que les résultats de notre collecte de données aient été plus faible que nos attentes, nous sommes satisfaits du résultat obtenu. Cela nous a fait prendre conscience de l'importance d'une base d'apprentissage, et que ça création n'est pas une tâche aisée.

Nous espérons que tous les travaux et résultats produit pendant ce projet puisse servir aux prochains étudiants qui souhaitent s'investir dans l'amélioration du quotidien des malvoyants et ainsi ressentir l'expérience de travailler pour une cause concrète.

# 6 Sources


- http://www.image-net.org/
- http://romain.raveaux.free.fr/document/TPApplicationAndroidpourlaclassification.pdf
- https://youtu.be/BdLjdqJwvhc
- https://youtu.be/KM3KK8VnxTI
- https://arxiv.org/pdf/1704.04861.pdf
- https://www.tensorflow.org/
- https://www.tensorflow.org/tutorials/images/transfer_learning
- https://github.com/tensorflow/examples/blob/master/lite/examples/image_classification/android/README.md
- https://www.tensorflow.org/lite/performance/post_training_quantization
- https://keras.io/api/applications/mobilenet/
- https://arxiv.org/pdf/1704.04861.pdf
- https://cocodataset.org/#home
- https://www.tensorflow.org/lite