Table des matières

Liste des tableaux

Liste des figures

1. Introduction

Nous vivons dans une société où les nouvelles technologies et les systèmes informatisés ont rendu facile la production et la diffusion d'informations. Il n'y a jamais eu auparavant autant de données aussi facilement accessibles et à portée de main. Devant une telle profusion d'informations, le besoin non seulement de stocker, mais également de traiter l'information est devenu croissant.

Actuellement, selon (Villinger 2019) une étude globale d'Avast a montré que les Suisses gardent en moyenne 1237 photos sur leur smartphone sans parler de leurs ordinateurs ou tablettes. Lors d'une enquête de police, ces images sont autant de données à traiter, un tri manuel de ces données est particulièrement chronophage et coûteux en ressources humaines.

Ce travail de Bachelor vise à proposer une solution à cette problématique de manière à répondre le mieux possible aux besoins de la Police cantonale de Genève.

La démarche entreprise dans un premier temps se résume à analyser les besoins du mandant, puis d'effectuer des recherches sur les technologies existantes permettant d'apporter une solution. Dans un second temps, l'accent est mis sur la conception et l'architecture de l'application puis la récolte des données. Dans un troisième temps, il s'agit de développer puis tester la solution pour qu'elle réponde aux besoins du mandant et qu'elle soit exploitable par ce dernier. Dans un quatrième temps, l'application est optimisée afin d'augmenter au maximum sa valeur ajoutée ou utilité pour le mandant.

Finalement, la solution est documentée afin de pouvoir être facilement déployée et reprise par un autre développeur dans le but de la compléter ou de l'augmenter.

1.1 Définitions

1.1.1 Indicateurs de performance

Résultat positif/négatif : Un résultat est positif si la prédiction indique un objet recherché, négatif s'il indique l'absence d'objet recherché.

Résultat vrai/faux : Un résultat est vrai si la prédiction (hypothèse) a été vérifiée, faux si elle n'a pas été vérifiée.

Exactitude (Accuracy) : Taux de résultats vrais positifs et négatifs, $\frac{VP+FN}{VP+FP+VN+FN}$

Précision (Specificity) : Taux de résultats vrais négatifs, $\frac{VN}{VN+FP}$

Plus la précision est haute, moins il y a de faux positifs (FP)

Sensibilité (Sensitivity or Recall) : Taux de résultats vrais positifs, $\frac{VP}{VP+FN}$

Plus la sensibilité est haute, moins il y a de faux négatifs (FN)

1.1.2 Termes techniques

Apprentissage automatique **:** *« Processus par lequel un algorithme évalue et améliore ses performances sans l'intervention d'un programmeur, en répétant son exécution sur des jeux de données jusqu'à obtenir, de manière régulière, des résultats pertinents. En anglais : « Équivalent étranger : machine learning (ML).* » (Gouvernement français, Commission d'enrichissement de la langue française, 2018, Texte 58)

Réseau de neurones artificiels **:** *« Ensemble de neurones artificiels interconnectés qui constitue une architecture de calcul. »* (Gouvernement français, Commission d'enrichissement de la langue française, 2018, Texte 58)

1.2 Portée et objectif

Le présent document fait office de travail de Bachelor. Il résume et documente également le déroulement du mandat délivré par la Police cantonale de Genève en termes d'analyse d'images et de reconnaissance d'objets dans le cadre de l'examen final de la Haute école de gestion de Genève, en vue de l'obtention du titre Bachelor of Science HES-SO en informatique de gestion.

L'objectif du document est à la fois de servir de travail de Bachelor et de documentation pour le mandant.

2. Analyse des besoins

2.1 Cahier des charges

Le mandat consiste à développer un logiciel permettant la reconnaissance d'objets. Le but est de pouvoir faciliter la reconnaissance d'objets (armes, drogues, argent) dans des images, afin de réaliser un premier triage, avant l'exploitation par la police. Les données à analyser seront issues de téléphones portables.

Exigences :

- Le logiciel doit pouvoir être entraîné avec des données d'entraînement afin d'atteindre un taux de reconnaissance le plus haut possible.
- Le logiciel doit pouvoir être réentraîné avec de nouvelles données.
- Le logiciel doit avoir une interface graphique facile d'utilisation pour l'utilisateur final.
- La précision doit être d'au moins 70%, la sensibilité de 98%.

Le mandat ne demande aucune exigence particulière de la part du mandant en termes de confidentialité.

Le présent cahier des charges a été validé par le mandant en date du 1er octobre 2019.

2.1.1 Qualités fonctionnelles

Pouvoir faciliter la reconnaissance d'objets (armes, drogues, argent) dans des photos, afin de réaliser un premier triage, avant l'exploitation par la police.

Pour ce faire, l'application doit permettre les fonctionnalités suivantes :

- Charger les données.
- Lancer une analyse.
- Consulter les résultats de l'analyse.

Voir la modélisation « Use case » au chapitre 4 « Architecture de l'application ».

2.1.2 Qualités non fonctionnelles

2.1.2.1 Sécurité

L'application ne doit pas être connectée au réseau. L'entier du code source doit être disponible.

2.1.2.2 Performance

Le temps d'une analyse ne doit pas excéder une journée de travail de 8 heures pour un volume de 10'000 images sur un ordinateur ayant une carte graphique correcte selon les standards actuels. (Intel graphique HD 620 au minimum)

2.1.2.3 Maintenabilité

L'application doit pouvoir être facilement augmentée et reprise par d'autres développeurs. Les erreurs de l'application sont archivées dans des logs.

2.1.2.4 Portabilité

L'application doit pouvoir être exécutée sur les systèmes Windows 8 et 10 et Linux Ubuntu et Debian.

2.1.2.5 Fiabilité

L'application doit éviter à tout prix une perte de données ou de compromettre le bon fonctionnement du système d'exploitation.

2.1.2.6 Utilisation

L'application doit être intuitive et facile d'utilisation. Les résultats affichés ne doivent pas être complexes ou compliqués à comprendre pour l'utilisateur final.

2.1.2.7 Disponibilité

L'application doit en tout temps pouvoir être directement disponible sur le système d'exploitation où elle est installée.

3. Choix technologiques

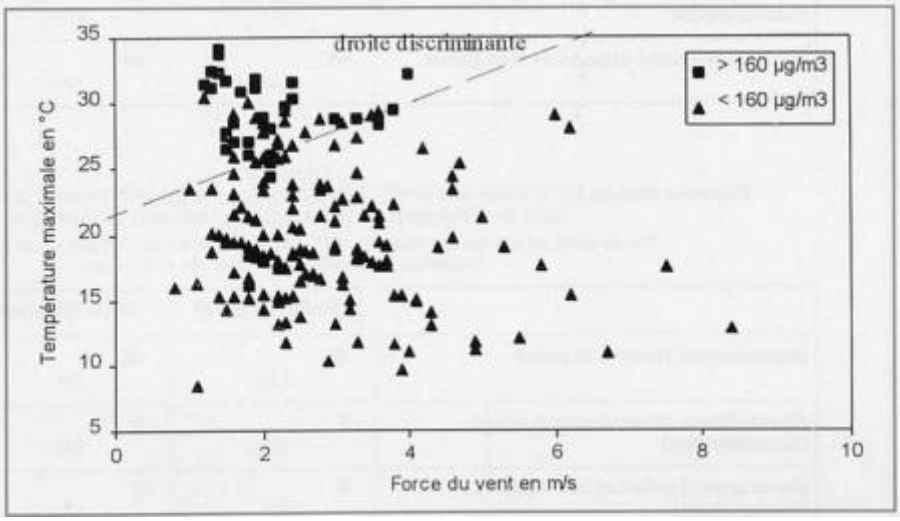
3.1 Rapide état des lieux des technologies permettant l'analyse d'images et la reconnaissance d'objets

3.1.1 Analyse discriminante

Il existe un grand nombre de méthodes statistiques traditionnelles permettant de classer des éléments ou de reconnaître des formes. Nous ne traiterons cependant que de l'analyse discriminante, car elle peut être utile à la compréhension de la solution retenue.

Extrait de la revue de statistique appliquée, selon Ulmo (1973, Première partie : l'analyse discriminante, page 18) : *« L'analyse discriminante (A.D.) est une technique de classement ou de reconnaissance des formes par opposition à la typologie ou à la classification. ».*

Figure 1 : Exemple d'application de l'analyse discriminante



Source : François Engel, Christian Viel et Hervé Chanut. « Développement d'un modèle statistique de prévision à 24 heures d'un dépassement du seuil d'information de la population pour l'ozone », Pollution atmosphérique [en ligne]. [Consulté le 11.02.2020]. Disponible à l'adresse : http://lodel.irevues.inist.fr/pollution-atmospherique/index.php?id=3538

Cette méthode répond à notre problématique consistant principalement à détecter la présence d'un objet sur une image, il suffit pour cela d'extraire les caractéristiques des images puis d'y appliquer l'analyse discriminante. Il existe d'autres approches qui pourraient répondre à cette problématique comme l'analyse en composant principal ou la classification bayésienne (Crowley 2001).

Cependant elles ne seront pas traitées dans le présent document, car elles ne sont pas ou plus utilisées dans le but de résoudre ce genre de problématique à l'heure actuelle

(VanderPlas 2016). Hormis une rapide exposition dans les chapitres suivants concernant l'apprentissage automatique et les réseaux de neurones (Bardos, Zhu 1997).

3.1.2 Apprentissage automatique

L'apprentissage automatique aussi appelé par son équivalent anglophone « machine learning » est un ensemble de méthodes à l'intersection des statistiques, de l'intelligence artificielle et de l'informatique. Il consiste à appliquer des algorithmes afin de donner aux ordinateurs la capacité d'apprendre en fonction des données qui leur sont présentées (Müller, Guido, 2017).

L'apprentissage automatique permet de détecter des objets dans des images et de les classer (Müller, Guido, 2017). Un exemple récent d'utilisation est la classification de tumeurs selon leur type et leur degré de croissance (Tang, Zawaski, Francis et al 2019).

Selon (Müller, Guido, 2017), en utilisant l'apprentissage automatique il suffit d'utiliser une grande quantité d'images pour que l'algorithme détermine les caractéristiques nécessaires pour identifier un objet, ce qui s'avère beaucoup plus compliqué avec une autre approche que l'apprentissage automatique comme l'élaboration de règles de décision nécessitant une compréhension approfondie du domaine. Qui plus est, le changement d'une seule règle peut nécessiter la réécriture de l'entier du système. Un exemple de l'échec de cette approche codée à la main consiste à détecter les visages dans les images. Aujourd'hui, chaque smartphone peut détecter un visage dans une image grâce à l'apprentissage automatique, mais la détection des visages était un problème non résolu jusqu'en 2001.

3.1.2.1 Apprentissage supervisé

Les algorithmes d'apprentissage automatique qui mettent en œuvre un apprentissage par paire des données d'entrée et de sortie (inputs/outputs) sont appelés « supervisés », car il y a une intervention et une supervision humaines (Müller, Guido 2017).

Actuellement, l'apprentissage automatique supervisé est la méthode la plus utilisée et souvent la plus efficace pour détecter un objet dans une image (Géron 2019). Ce type d'apprentissage correspond à notre problématique, car on connaît les données de sortie (outputs). Dans notre cas les données d'entrée sont les images et les données de sortie les objets recherchés.

Parmi les algorithmes d'apprentissage automatique supervisés les plus utilisés, on trouve : Les arbres de décision, les k plus proches voisins, les machines à vecteurs, la régression logistique, la classification naïve bayésienne et les réseaux de neurones profonds (Müller, Guido 2017).

3.1.2.2 Apprentissage non supervisé

Dans l'apprentissage non supervisé, seules les données d'entrée (inputs) sont connues. Les données de sortie (outputs) ne sont pas connues (Müller, Guido 2017).

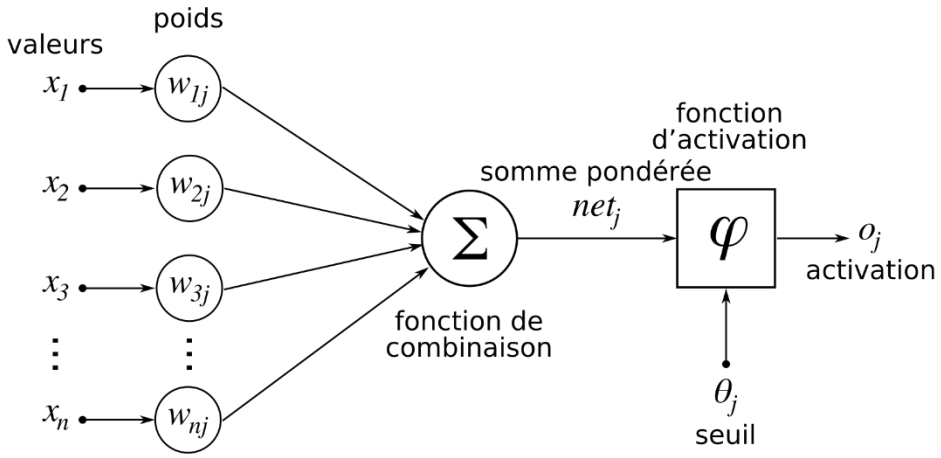
Ce type d'apprentissage est utilisé lorsqu'on ne connaît pas ou mal les données de sortie (outputs). Ce qui n'est pas le cas avec notre problématique.

3.1.3 Réseau de neurones artificiels

Un réseau de neurones artificiels est un assemblage interconnecté d'éléments, d'unités ou de nœuds de traitement simples, dont la fonctionnalité est globalement basée sur les neurones biologiques (animal ou humain). La capacité de traitement de l'information du réseau est stockée dans les forces ou poids d'interconnexion (poids synaptique), obtenus par un processus d'adaptation ou d'apprentissage à partir d'un ensemble de modèles de formation (Gurney 1997).

Les réseaux de neurones apprennent par induction, c'est-à-dire par expérience, en étant confrontés à des données en entrée (input) et en sortie (output).

Figure 2 : Schéma d'un neurone artificiel



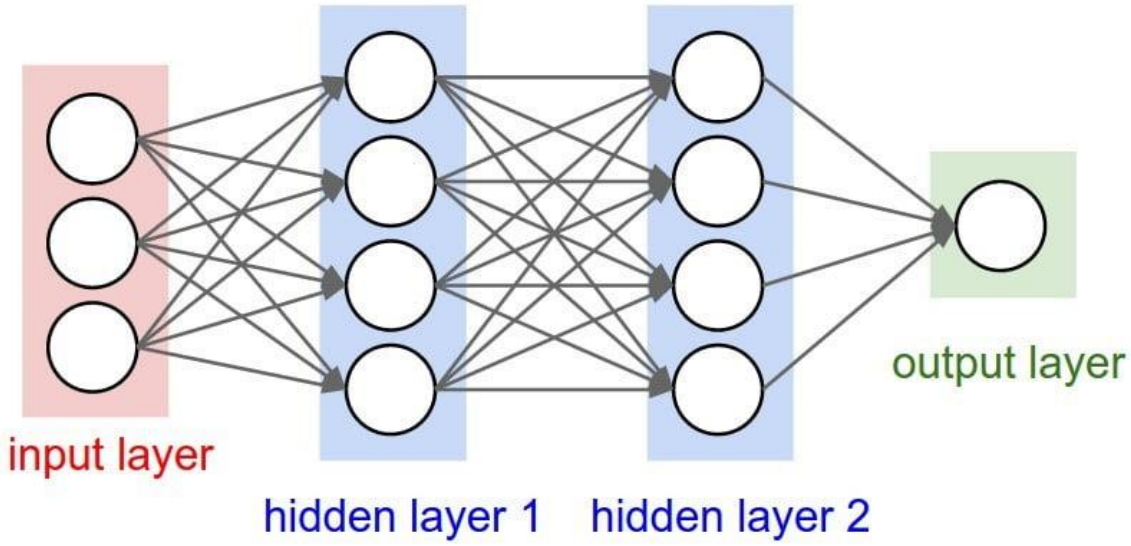
Source : Wikimédia, ArtificialNeuronModel francais [en ligne]. [Consulté le 11.02.2020]. Disponible à l'adresse : https://commons.wikimedia.org/wiki/File:ArtificialNeuronModel_francais.png

Selon (Bastien 2019) : *« En règle générale, un réseau de neurones repose sur un grand nombre de processeurs opérant en parallèle et organisés en tiers. Le premier tiers reçoit les entrées d'informations brutes, un peu comme les nerfs optiques de l'être humain lorsqu'il traite des signaux visuels.*

Par la suite, chaque tiers reçoit les sorties d'informations du tiers précédent. On retrouve le même processus chez l'Homme, lorsque les neurones reçoivent des signaux en

provenance des neurones proches du nerf optique. Le dernier tiers, quant à lui, produit les résultats du système. »

Figure 3 : Exemple d'architecture d'un réseau de neurones artificiels



(Bastien 2019)

Comme mentionné lors du précédent chapitre, les réseaux de neurones font partie de la famille des algorithmes d'apprentissage automatique (Müller, Guido 2017).

Champ d'application et utilisation

Selon (Bastien 2019), les utilisations actuelles des réseaux de neurones sont très variées. Cela va de la reconnaissance d'écriture manuscrite à la prévision des marchés financiers en passant par la reconnaissance faciale. Ils sont performants dans les domaines de la prédiction, de la reconnaissance de patterns et le traitement des signaux complexes.

Avantages et désavantages

Selon (Müller, Guido 2017), les réseaux de neurones ont refait surface en tant que modèles de pointe dans de nombreux domaines d'application de l'apprentissage automatique. L'un de leurs principaux avantages est qu'ils sont capables de capturer des informations contenues dans de grandes quantités de données et de créer des modèles incroyablement complexes. Avec suffisamment de temps de calcul, de données et un réglage soigneux des paramètres, les réseaux de neurones battent souvent d'autres algorithmes d'apprentissage automatique pour les tâches de classification et de régression. Cela nous amène aux inconvénients. Les réseaux de neurones, en

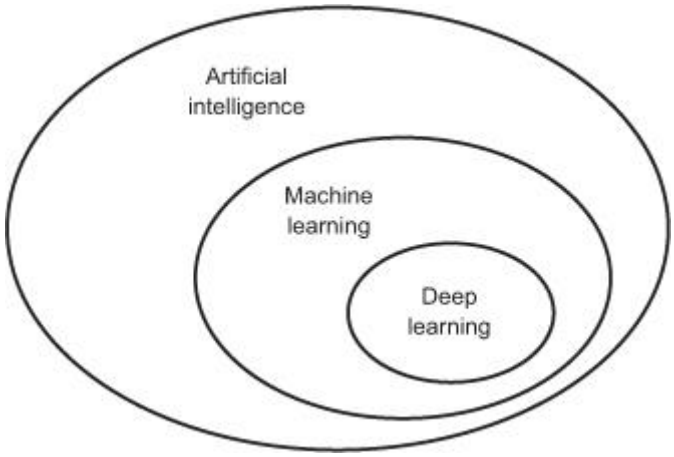
particulier ceux de grande taille, prennent souvent beaucoup de temps à se former et se construire. Ils nécessitent également un prétraitement et une préparation minutieuse des données.

3.1.4 Apprentissage profond

Selon (Chollet 2017), l'apprentissage profond (deep learning) est un sous-domaine spécifique de l'apprentissage automatique (machine learning). C'est une nouvelle approche de l'apprentissage des représentations à partir des données qui met l'accent sur l'apprentissage de couches successives de représentations de plus en plus conséquentes. Le terme « profond » dans l'apprentissage profond n'est pas une référence à une quelconque compréhension plus profonde au sens commun réalisée par l'approche ; il représente plutôt cette idée de couches successives de représentations. Le nombre de couches qui contribuent à un modèle de données définit la profondeur du modèle. Les réseaux de neurones profonds sont d'ailleurs parfois appelés « réseaux multicouches ». L'apprentissage profond moderne implique souvent des dizaines, voire des centaines de couches successives de représentations, et elles sont toutes calculées automatiquement par l'exposition aux données d'entraînement. Pendant ce temps, d'autres approches de l'apprentissage automatique ont tendance à se concentrer sur l'apprentissage d'une ou deux couches de représentations des données ; par conséquent, ils sont parfois appelés apprentissage superficiel.

Selon (Pradeep, Razaul, Mohit 2018), un système d'apprentissage en profondeur reconnaît l'image d'une personne en combinant les bords et les extrémités du corps de manière hiérarchique. Le jour peut-être n'est pas si loin où le deep learning sera étendu aux applications permettant aux machines de penser par elles-mêmes.

Figure 4 : Illustration de l’apprentissage profond par rapport à l’intelligence artificielle et l’apprentissage automatique



(Chollet 2017)

Les réseaux de neurones mettant en pratique le principe de l’apprentissage profond sont appelés réseaux de neurones profonds (DNN).

3.1.4.1 Champ d’application et utilisation

Selon (Chollet 2017), l’apprentissage profond (deep learning) a réalisé des percées importantes dans le domaine historiquement difficile de l'apprentissage automatique :

- Classification des images au niveau presque humain.
- Reconnaissance vocale au niveau quasi humain.
- Transcription manuscrite au niveau presque humain.
- Traduction automatique améliorée.
- Conversion de texte en parole améliorée.
- Assistants numériques tels que Google Now et Amazon Alexa.
- Conduite autonome au niveau presque humain.
- Amélioration du ciblage des annonces, tel qu'utilisé par Google, Baidu et Bing.
- Amélioration des résultats de recherches sur le Web.
- Capacité à répondre aux questions en langage naturel.
- Défaite d’un humain au jeu de Go.

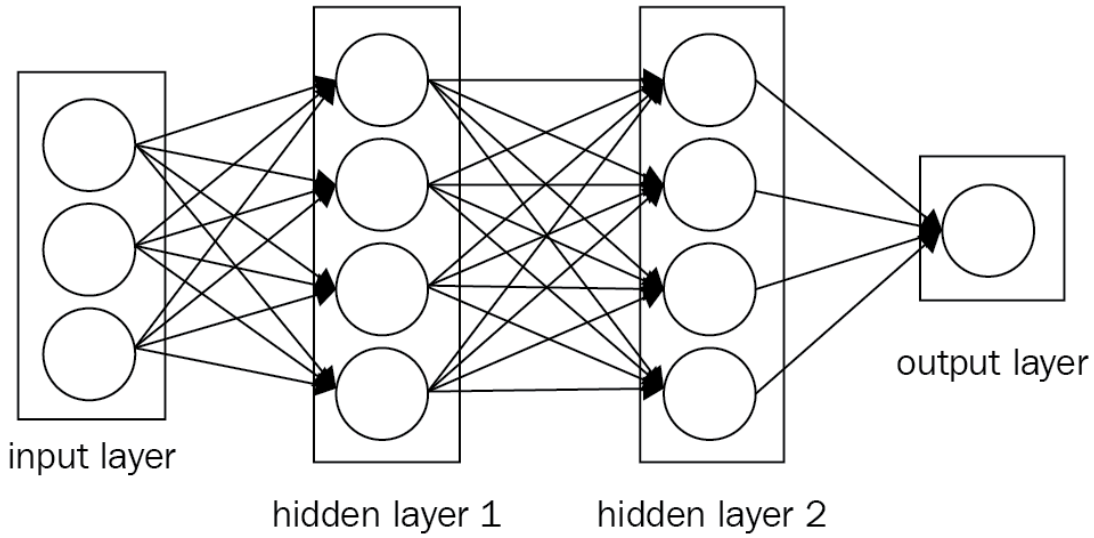
3.1.5 Réseau de neurones à convolution (CNN)

Les réseaux de neurones à convolution (CNN) sont une application des réseaux de neurones reposant sur l'application de filtres inspirés du fonctionnement de la partie du cerveau traitant du champ visuel.

Selon (Géron 2019), les réseaux de neurones à convolution (CNN) ont émergé de l'étude du cortex visuel du cerveau humain et animal, ils sont utilisés dans la reconnaissance et l'analyse d'images depuis les années 1980. Au cours des dernières années, grâce à l'augmentation de la puissance de calcul et à la forte quantité de données d'entraînement disponibles les CNN ont réussi à obtenir des performances surhumaines sur certaines tâches visuelles complexes. Ils alimentent les services de recherches d'images, les voitures autonomes, la vidéo automatique, etc.

Selon (Pradeep, Razaul, Mohit 2018), les CNN, aussi appelés « ConvNets », sont très similaires aux réseaux de neurones réguliers. Ils sont toujours constitués de neurones dont les poids peuvent être extraits des données. Chaque neurone reçoit des entrées (inputs) et effectue un produit scalaire. Ils ont toujours une fonction de perte sur la dernière couche qui est entièrement connectée. Ils peuvent utiliser une fonction de non-linéarité. Un réseau neuronal régulier reçoit les données d'entrée (inputs) comme un seul vecteur et traverse une série de couches cachées. Chaque couche cachée se compose d'un ensemble de neurones, chaque neurone étant entièrement connecté à tous les neurones de la couche précédente. À l'intérieur d'une seule couche, chaque neurone est entièrement indépendant et ne partage aucune connexion. La dernière couche complètement connectée, également appelée couche de sortie, contient des scores pour chaque classe (catégorie d'objets) permettant d'éviter les problèmes de classification d'images. En principe, il existe trois couches principales dans un simple réseau de neurones à convolution (CNN). Ce sont la couche de convolution, la couche de mise en commun et la couche entièrement connectée.

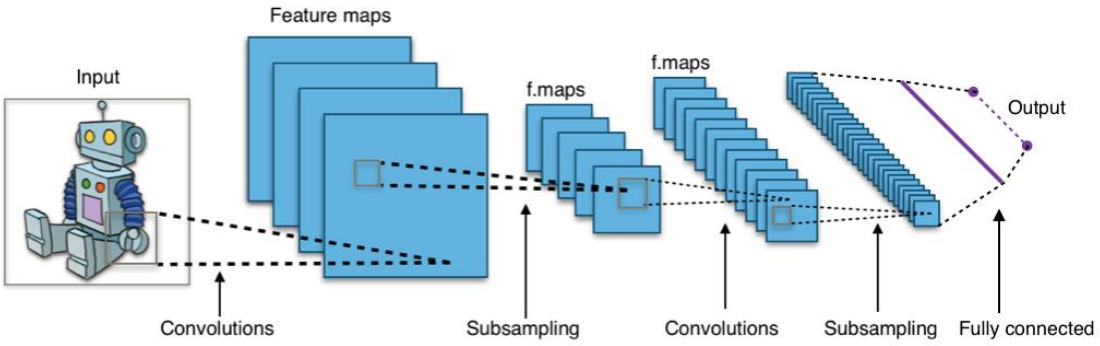
Figure 5 : Simple réseau de neurones à convolution à trois niveaux



(Pradeep, Razaul, Mohit 2018)

Selon (Pradeep, Razaul, Mohit 2018), une forme de réseau qu'on retrouve de manière systématique consiste à répéter le schéma précédant (ci-dessus) en empilant à la suite les couches de convolution et de mise en commun (Subsampling) jusqu'à parvenir à des entrées très petites. Cela permet de prétraiter de petites quantités d'informations et par conséquent d'augmenter les performances. Exemple d'un réseau de neurones à convolution appliquant ce principe avec la figure ci-dessous :

Figure 6 : Exemple d'architecture d'un réseau de neurones à convolution



Source : Wikimédia, Typical cnn.png [en ligne]. [Consulté le 11.02.2020]. Disponible à l'adresse : https://commons.wikimedia.org/wiki/File:Typical_cnn.png

Champ d'application et utilisation :

Selon (Géron, 2019), les réseaux de neurones à convolution sont utilisés pour les services de recherche d'images, les voitures autonomes, la vidéo automatique. Ils sont particulièrement efficaces dans le traitement d'images et de vidéos.

Pour notre problématique consistant à détecter des objets dans une image pour ensuite les classifier, ce type de réseau de neurones paraît approprié.

3.2 Choix des technologies permettant l'analyse d'images et la reconnaissance d'objets

3.2.1 Choix de la technologie et de l'algorithme

Parmi toutes les solutions possibles étudiées au chapitre 3.1 « Rapide état des lieux des technologies permettant l'analyse d'images et la reconnaissance d'objets » celle retenue est l'apprentissage automatique profond, plus précisément les réseaux de neurones à convolution. C'est avec cette technologie que l'on trouve l'écrasante majorité des solutions récentes (postérieur à 2010) à des problématiques similaires à la nôtre.

C'est également la solution proposée par tous les ouvrages récents étudiés dans le cadre de ce travail de Bachelor traitant d'une problématique similaire publiés après 2010 : (Géron 2019), (Chollet 2017), (Pradeep, Razaul, Mohit 2018), (Müller, Guido 2017).

Cette technologie est également la seule accessible ne nécessitant pas de connaissances poussées dans le traitement des signaux et de l'image pour être mise en place. Selon (Godec, Pancur, Ilenic et al 2019), les approches d'apprentissage approfondi pour l'analyse d'images offrent la possibilité de développer des outils conviviaux pour l'analyse exploratoire des données sans nécessiter d'expertise dans le domaine étudié.

L'accessibilité des librairies et des outils permettant l'analyse d'images et la détection d'objets a joué un grand rôle dans le choix des technologies retenues. Seuls les outils utilisant l'apprentissage automatique étaient envisageables pour un étudiant seul, sans budget, et avec les contraintes temporelles inhérentes à ce mandat. Pouvoir s'appuyer sur des ressources existantes était donc impératif.

Les arguments en faveur de réseaux de neurones profonds sont si écrasants qu'il n'a pas été jugé utile de faire un tableau de comparaison ou une matrice de préférence pour des raisons de temps et de priorité des objectifs du présent travail de Bachelor.

Par conséquent, la technologie retenue est l'apprentissage automatique profond (deep learning), car c'est celle qui répond le mieux aux besoins cités au chapitre 2 « Analyse des besoins ».

3.2.1.1 Classification ou détection d'objets ?

Le premier choix technologique ayant demandé une analyse poussée et des tests a été de savoir s'il fallait se contenter d'un modèle de classification d'images ou opter pour un modèle de détection d'objets, usuellement utilisé pour répondre à des problématiques similaires. De prime abord un modèle de détection d'objets paraît plus adapté ce qui en réalité dépend des besoins et de l'application qu'on en fait.

3.2.1.1.1 Brève définition

Un modèle de détection détecte un objet en fonction des données qu'on lui donne lors de l'entraînement. Ces données d'entraînement sont des images contenant l'objet avec sa position sur l'image afin d'apprendre au modèle à situer la position de l'objet dans l'image.

Selon Géron (2019, chapitre 14) :

> *« The task of classifying and localizing multiple objects in an image is called* object detection »

Un modèle de classification se contente de classer les images en fonction des données d'entraînement, ces données sont des images (images, inputs) avec leurs classes respectives (objets recherchés, outputs). Il extrait toutes les caractéristiques des images d'entraînement pour ensuite les faire correspondre aux classes lors de la prédiction.

Un classificateur avec un réseau de neurones à convolution est conçu de manière à ce que la couche de convolution, par un traitement vectoriel des données, puisse reconnaître une forme, quelle que soit sa position dans l'image.

Selon Géron (2019, chapitre 14) :

> *« Once the CNN has learned to recognize a pattern in one location, it can recognize it in any other location »*

Il faut aussi savoir que les modèles de détection d'objets utilisent des modèles des réseaux de neurones à convolution pour faire de la classification.

Selon Géron (2019, chapitre 14) :

> *« This simple approach to object detection works pretty well, but it requires running the CNN many times, so it is quite slow. Fortunately, there is a much*

faster way to slide a CNN across an image: using a fully convolutional network (FCN). »

3.2.1.1.2 Résultats des tests préliminaires

Tous les modèles de détection d'objets avaient une précision supérieure aux modèles de classification, cependant la sensibilité est considérablement et systématiquement inférieure. Il suffit qu'un objet soit à moitié dissimulé ou en perspective pour que les modèles de détection d'objets ne le détectent plus.

Les tests ont été effectués avec les modèles Inception v3 et MobielNet v2 comme modèles de classification. Yolo v3 et inception resnet v2 pour les modèles de détection d'objets. Les données d'entraînement sont constituées d'environ 1000 images. Les données de test sont constituées d'environ 800 images.

3.2.1.1.3 Choix du modèle

Le mandant a clairement exprimé qu'il préfère une grande sensibilité (minimum 98%) à la précision (minimum 70%). Il est également ressorti de l'analyse des besoins (chapitre 2) que le mandant n'a pas besoin de connaître la position d'un objet dans une image, mais seulement sa présence dans le but de trier l'image contenant ce dernier.

En tenant compte de l'analyse des besoins (chapitre 2) on peut déduire les critères suivants :

- Permet de détecter un objet dans le but de trier celui-ci
- Favorise la sensibilité même au détriment de la précision
- Facilement réentraînable

<u>Permet de détecter un objet dans le but de trier celui-ci :</u>

Les deux types de modèles remplissent entièrement ce critère. Selon (Géron 2019) un modèle de classification tout comme un modèle de détection permet de trier une image.

<u>Favorise la sensibilité (même au détriment de la précision) :</u>

La classification permet de prendre en compte le contexte de l'objet, c'est-à-dire la partie de l'image ne constituant pas l'objet à détecter en lui-même. Cela a pour conséquence de faire baisser la précision, car le modèle ne prendra pas seulement en compte l'objet à détecter, mais toutes les caractéristiques des images d'entraînement, dont le contexte et l'arrière-plan de l'image. Cependant cela augmente de manière non négligeable la sensibilité, car le contexte et ce qui se trouve autour de l'objet à détecter sont pris en

compte dans la prédiction. Un modèle de classification paraît préférable à un modèle de détection au vu de ce critère.

Facilement réentraînable :

Selon (Montaigu 2018), le réentraînement partiel ou total d'un modèle de détection d'objets nécessite au préalable de labéliser les images d'entraînement afin d'indiquer au modèle l'emplacement sur l'image de l'objet à détecter. Cette tâche bien que prenant peu de ressources en termes de temps pour le traitement d'une seule image (environ 3 à 10 secondes) devient conséquente sur une banque de plusieurs milliers, voire dizaine de milliers d'images.

Pour entraîner un modèle de classification, il suffit de réunir des images puis de les classer au préalable afin que le modèle sache à quelle classe (catégorie) les associer.

Tableau 1 : Comparaison détection - classification

Critères/types modèles	Détection	Classification
Permet de détecter un objet dans le but de trier celui-ci	OUI	OUI
Favorise la sensibilité	NON	OUI
Facilement réentraînable	NON	OUI

En conclusion, le type de modèle choisi est la classification, car il répond positivement à plus de critères, cependant il est intéressant de considérer la possibilité d'augmenter la précision avec un modèle de détection ad hoc.

3.2.1.2 Transfer learning et fine tuning

Selon (Montaigu 2018), il existe trois approches différentes pour mettre au point et entraîner un modèle de détection d'objets ou de classification :

- Créer soi moi-même un modèle.
- Utiliser un modèle préentraîné.
- Utiliser un modèle préentraîné et effectuer un transfert de connaissances (Transfer learning) afin de réentraîner partiellement le modèle.

Selon (Montaigu 2018), créer un modèle de classification ou de détection d'objets soi même est très coûteux en ressources et demande beaucoup de compétences, cette solution n'est pas préférable dans notre cas faute de moyens (financement, temps, une seule personne).

Utiliser un modèle préentraîné est relativement facile. Cependant, si le modèle n'a pas été entraîné à reconnaître ou classer le même type d'images que celles exploitées pour le réentraînement, les performances seront basses.

> *« Lorsque vous travaillez avec un jeu de données plutôt différent du jeu de données original utilisé pour l'entraînement du modèle, simplement l'appliquer ne fonctionnera pas. Vous devrez construire votre propre jeu de données d'entraînement et ré-entraîner le modèle sur celui-ci. ».* (Montaigu 2018)

Dans notre cas, notre jeu de données constitué d'images d'armes, de drogue et d'argent diffère des jeux de données des modèles préentraînés mis à disposition des développeurs. Il est donc nécessaire de réentraîner ne serait-ce que partiellement le modèle grâce au transfert learning.

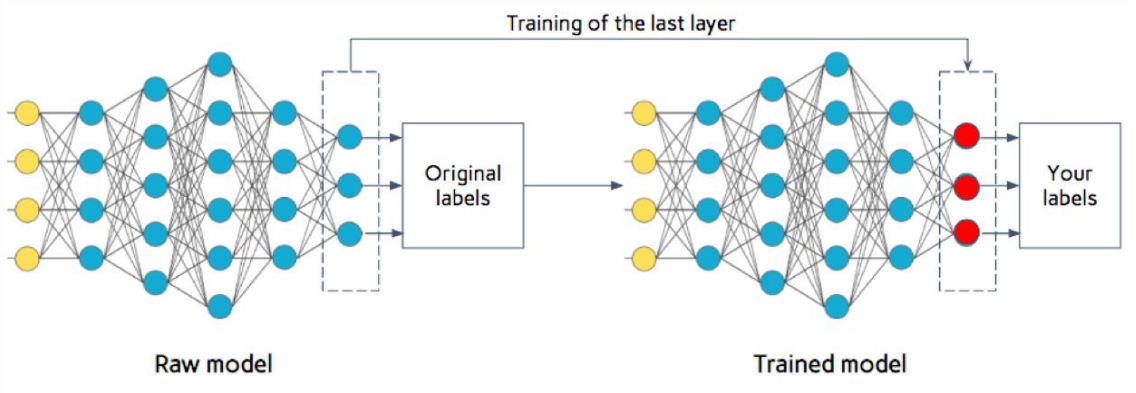
> *« Le Transfer Learning vise à transférer des connaissances d'une ou plusieurs tâches sources vers une ou plusieurs tâches cibles. Avec cette technique, il est possible d'appliquer des données apprises à partir de tâches antérieures sur de nouvelles tâches ou domaines partageant des similitudes. »*. (Montaigu 2018)

> *« Le Transfer Learning consiste à utiliser les couches convolutives pré-entraînées du modèle original et à entraîner les couches entièrement connectées sur les features extraites de notre ensemble d'images. L'idée ici est d'utiliser les connaissances acquises lors de l'entraînement du modèle sur le jeu de données original et de les utiliser pour notre problème. Cette technique est efficace dans les cas où les deux jeux de données sont semblables. ».* (Montaigu 2018)

C'est le cas avec notre jeu de données, qui constitué d'objets ayant des similitudes avec les données ayant entraîné les modèles préentraînés à disposition. Il est à noter que la

librairie TensorFlow (TensorFlow 2020a), utilisée par (Montaigu 2018) est celle retenue pour le projet (chapitre 3.2.2 choix librairie). Cette dernière permet de faire du transfert learning sur des modèles de détection ou de classification préentraînée.

Figure 7 : Schéma de transfert de connaissance (transfer learning)



(MONTAIGU, 2018)

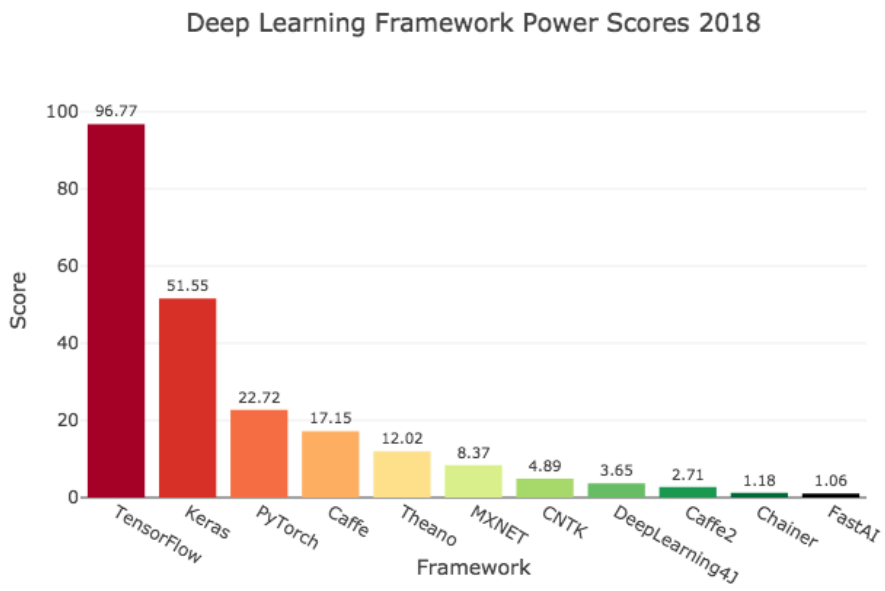
3.2.2 Choix de l'outil d'apprentissage automatique

L'outil choisi afin de pouvoir mettre en œuvre un programme d'apprentissage automatique est la librairie TensorFlow, développée par Google elle permet de mettre au point des modèles utilisant l'apprentissage automatique. (TensorFlow 2020)

> *« TensorFlow est une bibliothèque mathématique open-source, fournissant des API Python et C stables, utilisées pour plusieurs tâches de manipulation de données. Elle est très connue et utilisée pour le machine learning et les applications de deep learning telles que les réseaux neuronaux. Les cas d'utilisation pour lesquels TensorFlow est le plus connu sont la reconnaissance d'images, le traitement du langage naturel et l'analyse speech-to-text. ».* (Montaigu 2018)

TensorFlow et Keras sont les plus documentées et les plus faciles à prendre en main. De plus elles sont open source et libre d'accès. La librairie est sous licence Apache version 2 (TensorFlow 2020). Depuis octobre 2019 TensorFlow implémente la libraire Keras comme API interne. (Passos, Bailey, 2019)

Figure 8 : Comparaison librairies apprentissage automatique



(Hale 2018)

TensorFlow est également utilisé par les auteurs (Géron 2019), (Chollet 2017), (Razaul, Mohit 2018), (Müller, Guido 2017) pour faire de l'apprentissage automatique.

3.2.3 Choix du langage de programmation

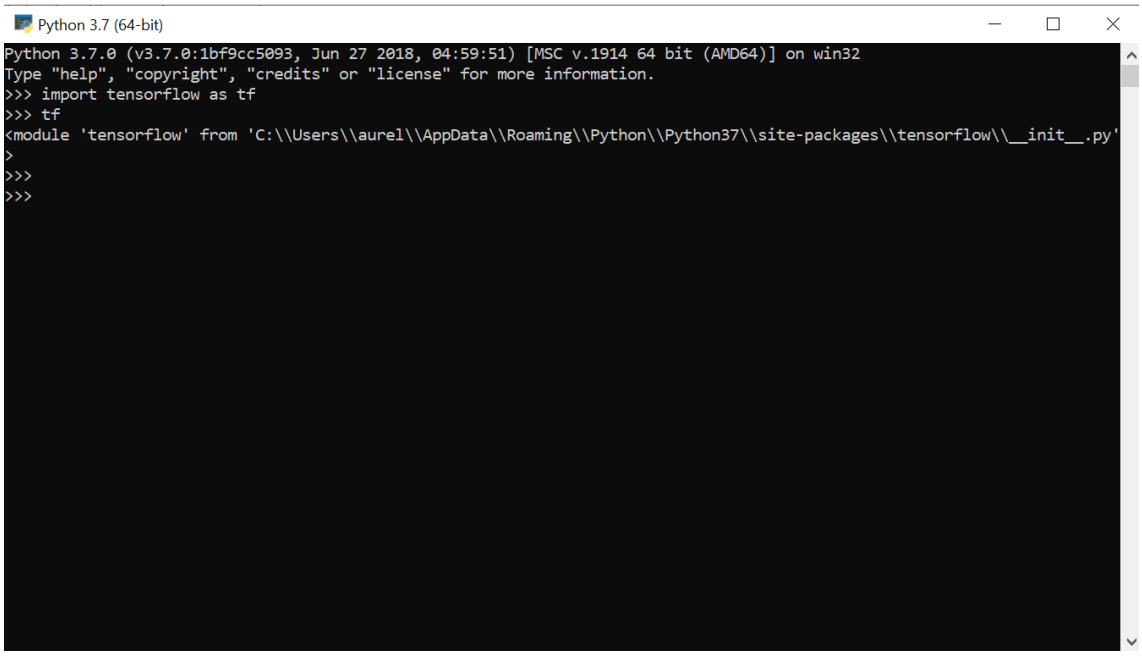
Le choix du langage va de pair avec celui de l'outil d'apprentissage automatique, car il s'agit de choisir un langage qui soit adapté et compatible avec ce dernier. Dans notre cas, Python a été choisi, car c'est le langage le plus adéquat pour travailler avec TensorFlow, et c'est également une préférence émise par le mandant. De plus, selon (Blondelle 2018), la majorité de la documentation existante sur TensorFlow est en Python.

3.2.4 Environnement de développement

L'environnement de développement ainsi que la procédure d'installation sont détaillés dans l'annexe « Manuel d'installation des outils de développement ».

La version de Python choisie est la 3.7, car c'est la plus récente compatible avec la version 2.0 de TensorFlow.

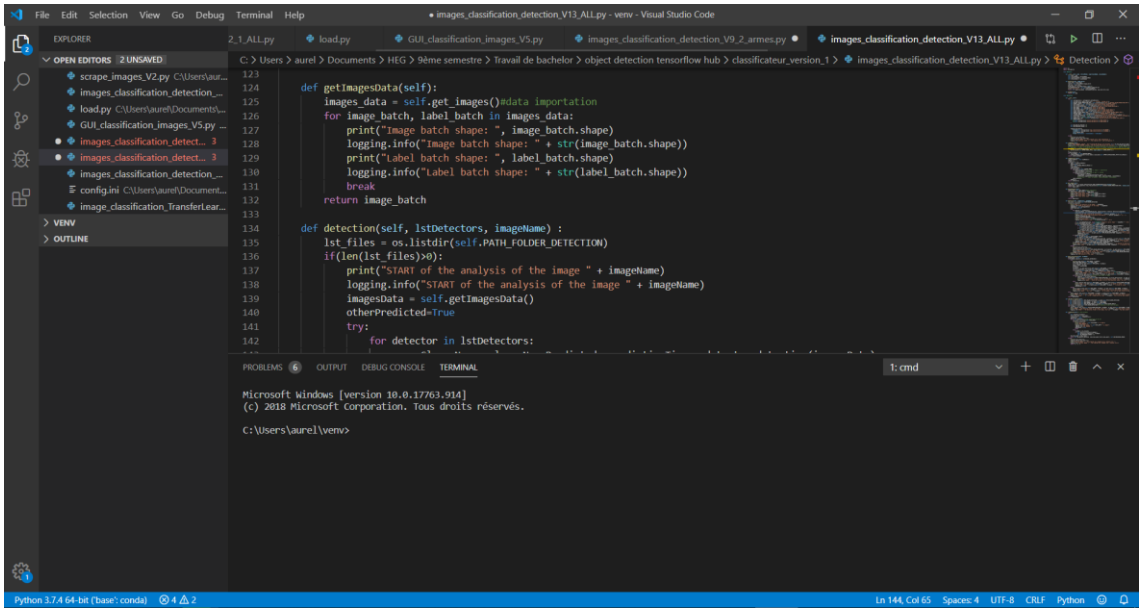
Figure 9 : Console Python



(Réalisé à l'aide de l'outil : Python 3.7)

L'IDE (environnement de développement intégré) choisi pour le développement est Visual Code de Microsoft version 1.41.1. Ce choix est plutôt personnel.

Figure 10 : IDE Visual Code



(Réalisé à l'aide de l'outil : Visual Studio Code)

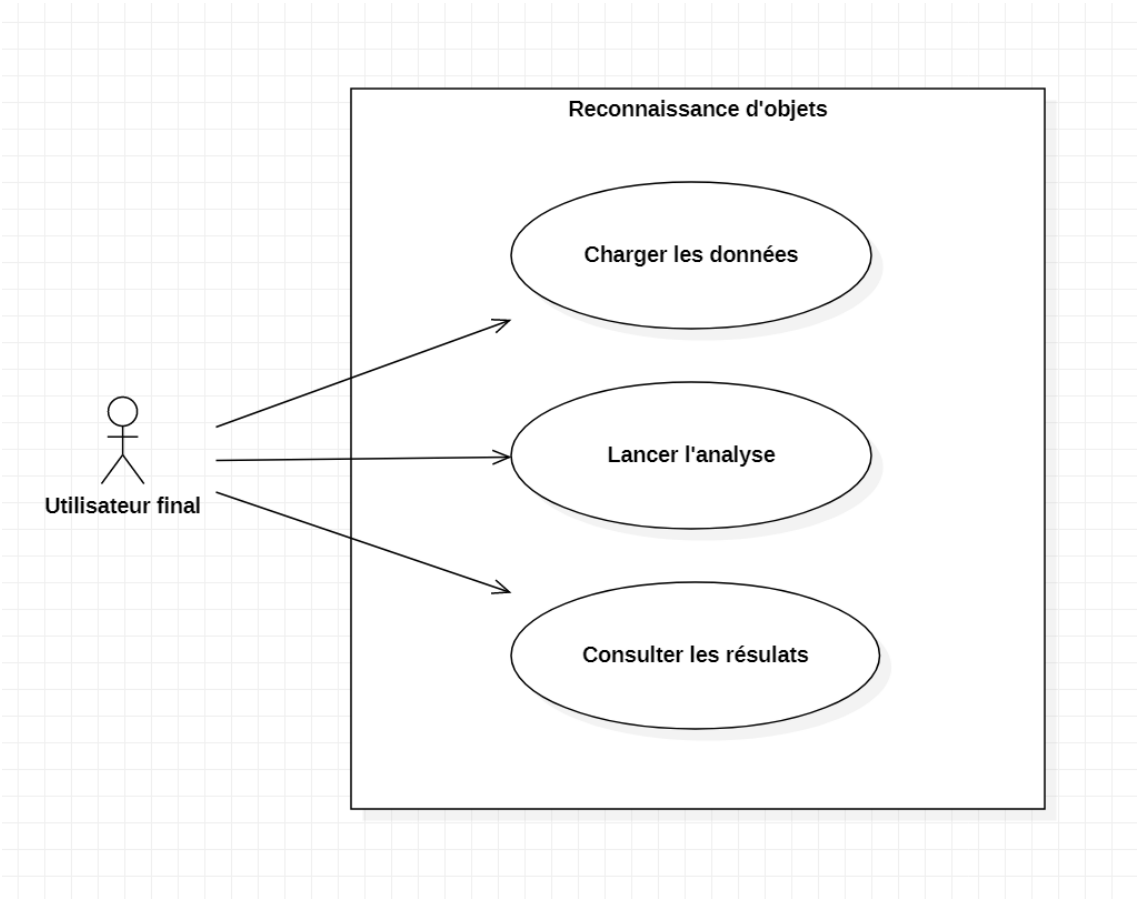
3.2.5 Environnement de déploiement

L'environnement où l'application sera déployée et utilisée est le parc informatique de la Police cantonale genevoise. Sa constitution n'est pas décrite dans ce document pour des soucis de confidentialité.

4. Architecture de l'application

4.1 Use case

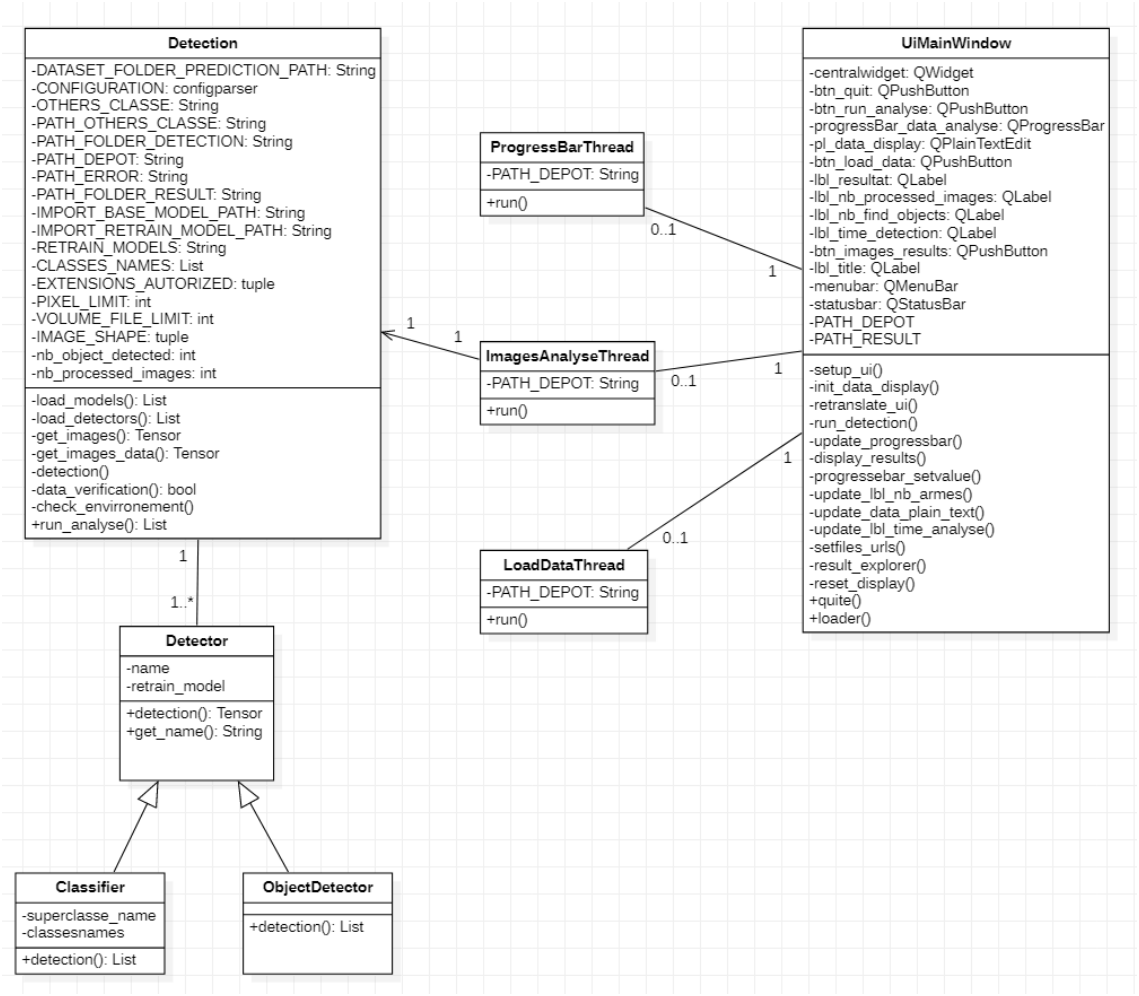
Figure 11 : Use case



(Réalisé à l'aide de l'outil : StarUML)

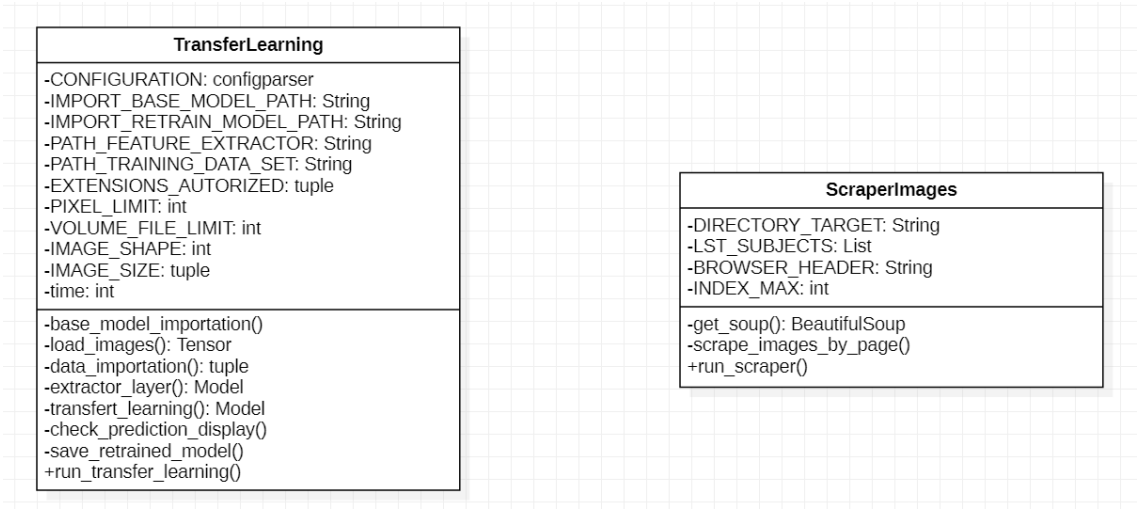
4.2 Diagramme de classes

Figure 12 : Diagramme de classes 1



(Réalisé à l'aide de l'outil : StarUML)

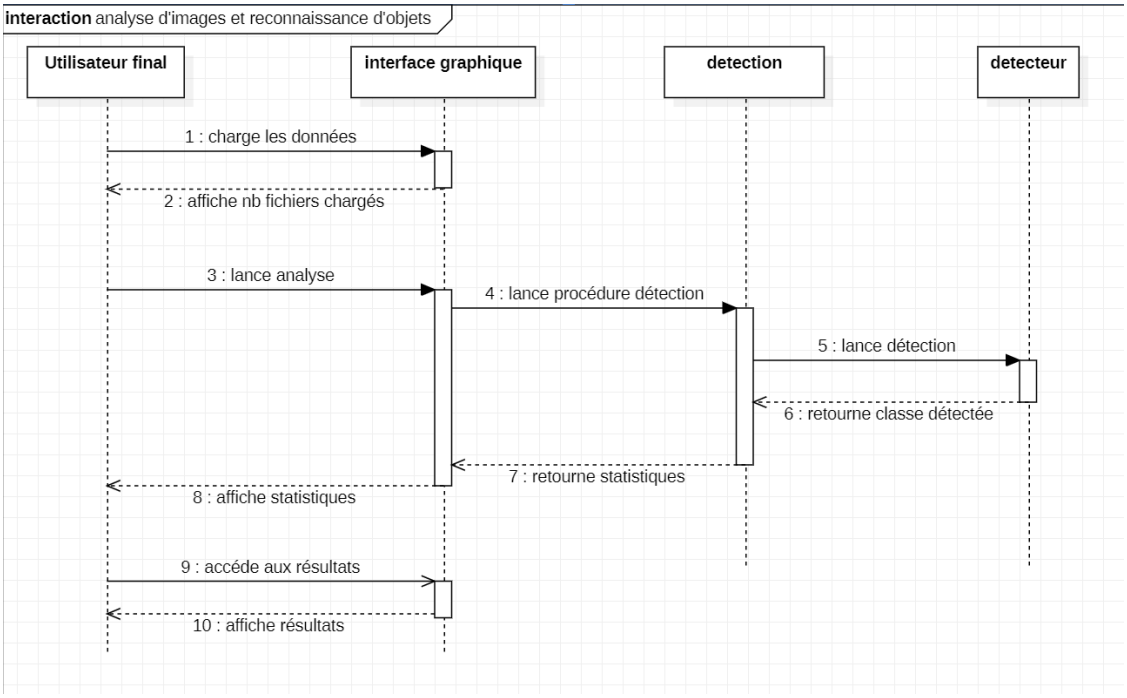
Figure 13 : Diagramme de classes 2



(Réalisé à l'aide de l'outil : StarUML)

4.3 Diagramme de séquence

Figure 14 : Diagramme de séquence



(Réalisé à l'aide de l'outil : StarUML)

Pour charger les données, lancer l'analyse et mettre à jour l'interface graphique, le programme lance des threads qui ne figurent pas sur ce diagramme par souci de concision et de compréhensibilité.

4.4 Arborescence de l'application

L'arborescence des dossiers et fichiers de l'application est la suivante :

- **analyse_images_detection_objets**
  - **config**
    - *config.ini*
  - **data**
    - **dataset_training**
    - **depot**
    - **prediction**
      - **detection**
    - **resultat**
  - **logs**
  - **models**
    - **feature_extractor_layer**
    - **mobilNet_V2**

    - **retrained_models**
- **ressources**
    - *icon.png*
- **scripts**
    - **__pycache__**
    - *__init__.py*
    - *detectors.py*
    - *object_detection.py*
    - *script_transferlearning.py*
    - *ui_object_detection.py*
- *readme.txt*
- *requirements.txt*
- *run.pyw*

5. Analyse et choix des données utilisées

5.1.1 Objets à détecter et catégories

Les objets qui doivent être détectés, voir chapitre 2 « Analyse des besoins », appartiennent aux catégories suivantes :

- Armes
- Drogues
- Argent

5.2 Caractéristiques des données

5.2.1 Définition, quantité et qualité

Définition : Les données traitées sont des images au format électronique. Les catégories d'objets à détecter « armes », « drogues » et « argent » sont définies par la loi suisse en vigueur à ce jour. (Confédération suisse 2019, 2020a, 2020c)

Quantité : Plusieurs milliers

Qualité : Les données doivent être représentatives de l'utilisation finale du programme et de nature variée, leur format ne doit pas dépasser 4'000'000 pixels.

5.2.2 Origine des données

5.2.2.1 Données d'entraînement

La majorité des données d'entraînement sont des images récupérées de manière automatisée sur des moteurs de recherche ainsi que dans la banque de données comme Image Net (Image net 2019)

Le choix de récupérer des données sur un grand nombre de sites plutôt que via des banques de données déjà prêtes provient du fait que les catégories traitées (Armes, drogues et argent) sont très peu représentées dans la banque de données mise à disposition sur internet. Après avoir demandé à la Police cantonale, la Police fédérale suisse ou encore l'École de sciences criminelles de l'Université de Lausanne, ces derniers n'ont pas pu fournir de données. Il a donc fallu improviser et trouver une solution afin de récolter suffisamment de données correspondant aux bonnes caractéristiques, voir chapitre 5 « caractéristiques des données ». Ce script a aussi l'avantage de permettre d'amasser une grande quantité d'images selon des critères très précis en vue d'ajout de nouvelles catégories. Il faut cependant noter que cette solution a comme défaut majeur que les images que retournent les moteurs de recherches ne sont pas toujours pertinentes, il faut donc effectuer un tri manuel afin d'avoir des données qui correspondent entièrement aux critères de sélection. Ce tri est chronophage et limite la quantité de données à disposition.

5.2.2.2 Données cibles

Les données que l'application sera amenée à traiter lors de son utilisation, conformément au mandat pour ce Travail de Bachelor proviennent d'images en possession de la police cantonale genevoise, voir chapitre 2 « Analyse des besoins ». Ces images proviennent de téléphones portables ou potentiellement d'ordinateurs personnels. Leur type, leur quantité et leur qualité peuvent sensiblement varier et sont difficilement prévisibles.

5.3 Prétraitement et préparation des données

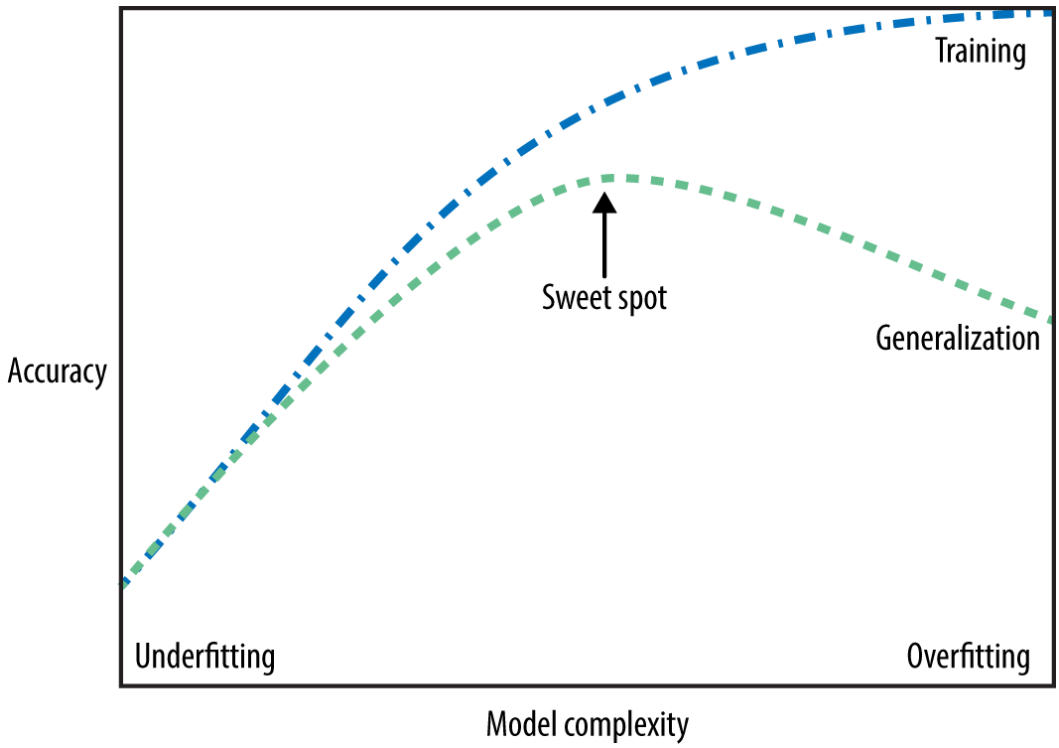
Selon (Montaigu 2018), la sélection et la préparation ainsi que le prétraitement des données sont une phase essentielle de l'apprentissage automatique qui consiste à choisir scrupuleusement les données d'entraînement de manière à ce qu'une fois entraîné le modèle se comporte de manière attendue.

Une faible quantité de données d'entraînement peut mener à un sous-apprentissage, de même qu'une trop grande quantité peut causer un surapprentissage. Les paramètres d'apprentissage qui varient selon l'algorithme jouent un rôle important, mais dans notre cas les modèles préentraînés ne permettent que rarement de les modifier.

Lorsqu'un modèle est en « surapprentissage », cela signifie que le modèle colle trop aux données d'entraînement (overffiting), ce qui dans notre cas est peu probable au vu du fait que le modèle préentraîné a été conçu pour être entraîné par plusieurs millions d'images.

Il est par contre très probable que le modèle soit en « sous-apprentissage » (underfitting), car la quantité de données à disposition est relativement faible. L'idéal est d'atteindre le point d'équilibre entre le surapprentissage et le sous-apprentissage. (Sweet spot)

Figure 15 : Illustration du principe de sur et sous-apprentissage



(Müller, Guido 2017)

Dans notre cas, nous recherchons une forte sensibilité (98%) et une précision qu'on peut considérer comme moyenne, voire faible (70%). Il est apparu à la suite de multiples tests que plus les images d'une classe (catégorie) sont variées, plus la sensibilité augmente. Le fait de prendre en compte le contexte et de présenter des images contenant l'objet dans des contextes où la présence de l'objet paraît probable augmente également la sensibilité au détriment de la précision.

Il faut partager les images dans les classes définies, dans notre cas ce sont les dossiers présents dans le set d'entraînement. Il est à noter que la présence d'une classe « autres » est nécessaire, car le modèle va dans tous les cas rendre une prédiction en

fonction de classes d'entraînement. Ce dossier « autres » se doit de contenir un grand nombre d'images diverses et variées.

Sachant qu'il faut un classificateur par type d'objets recherchés, les données de la classe « autres » doivent également contenir les classes des autres classificateurs. Si ce n'est pas le cas, il y a un risque non négligeable que la classification se fasse mal à cause d'un contexte commun de l'image. Il est nécessaire d'entraîner partiellement un modèle de classification pour chaque type d'objets.

Les modèles préentraînés ont été entraînés avec des sets de données de plusieurs millions d'images (TensorFlow Hub 2019) et paramétrés en conséquence. En ajoutant une classe « autres » on utilise le préentraînement du modèle sur des centaines (TensorFlow Hub 2019) de classes pour les faire converger. Un mécanisme similaire est utilisé pour le détecteur d'objets.

6. Développement et implémentation

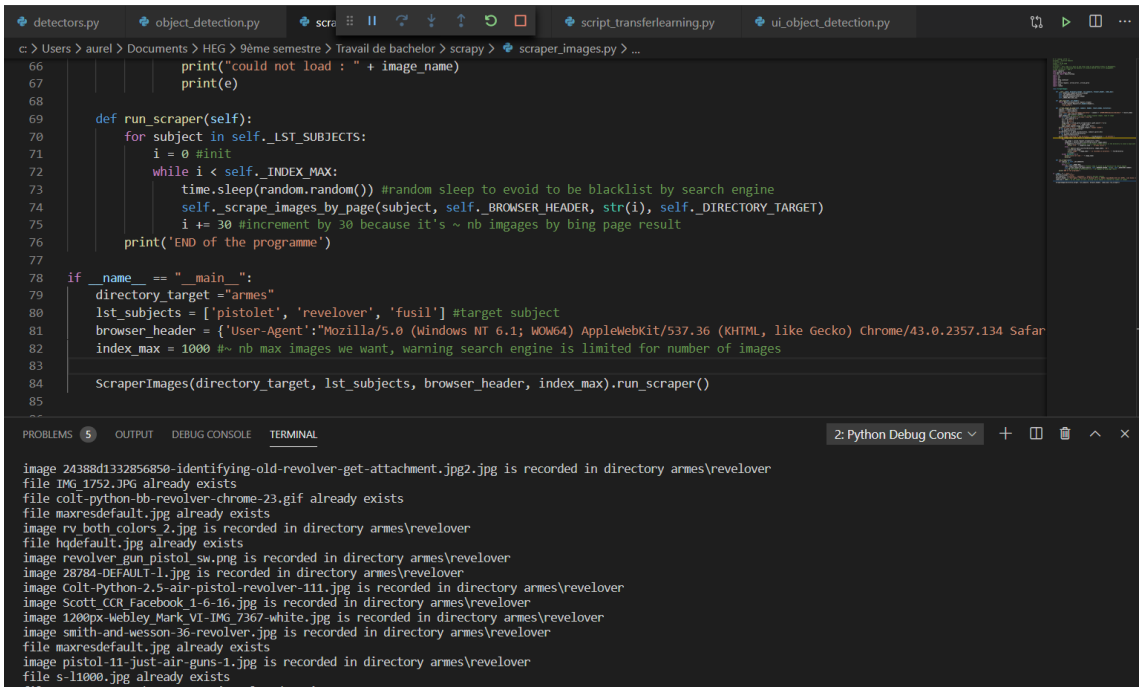
Ce chapitre décrit les différentes phases du développement de la solution. Sont également décrites les principales fonctionnalités des différentes parties (sous-programmes) de l'application. Le code est entièrement en langue anglaise afin de pouvoir être facilement repris par un autre développeur. La documentation sur le sujet étant essentiellement anglophone cela simplifie la compréhension et la lecture du code.

6.1 Récolte des données

Lors de la phase de développement, il a d'abord fallu récolter des données afin de pouvoir entraîner et tester le futur modèle de classification.

Comme mentionné dans le chapitre précédent 5.1.3 « Origine des données » le choix s'est porté vers une récolte des images présentes et accessibles sur les moteurs de recherche présents sur internet. Pour ce faire, il a fallu développer un « scraper », un script en Python qui parcourt la toile en émulant un navigateur web pour se connecter au moteur de recherche Bing (Bing 2019). Le script effectue des recherches par mot clé puis télécharge directement les fichiers sur le poste depuis lequel il est exécuté.

Figure 16 : Code source et exécution du scraper



(Réalisé à l'aide de l'outil : Visual Studio Code)

Pour utiliser le scraper, il faut définir le dossier cible et la liste de mots clés. Il suffit ensuite de lancer le script à l'aide de l'interpréteur Python. Les sujets de recherches composés de plusieurs mots clés doivent être liés avec le caractère « + ».

Mise en garde : Les fichiers téléchargés peuvent être corrompus ou contenir du code malicieux. Il est conseillé de prendre des mesures de sécurité avant d'utiliser le script. Une solution possible est d'exécuter le script sur une machine virtuelle isolée du système principal et connectée à un VPN puis d'utiliser un antivirus pour filtrer les données. Cet outil de récolte de données ne fait pas partie de l'application principale. Ce dernier est prévu pour une utilisation conforme au droit suisse.

6.2 Modèle de classification

Le développement du modèle de classification constitue la majeure partie du développement de l'application. Il consiste à mettre en place la partie de l'application permettant de détecter des objets dans des images tout en respectant le cahier des charges, voir chapitre 2 « Analyse des besoins ».

6.2.1 1re étape – modèle et structure de détection

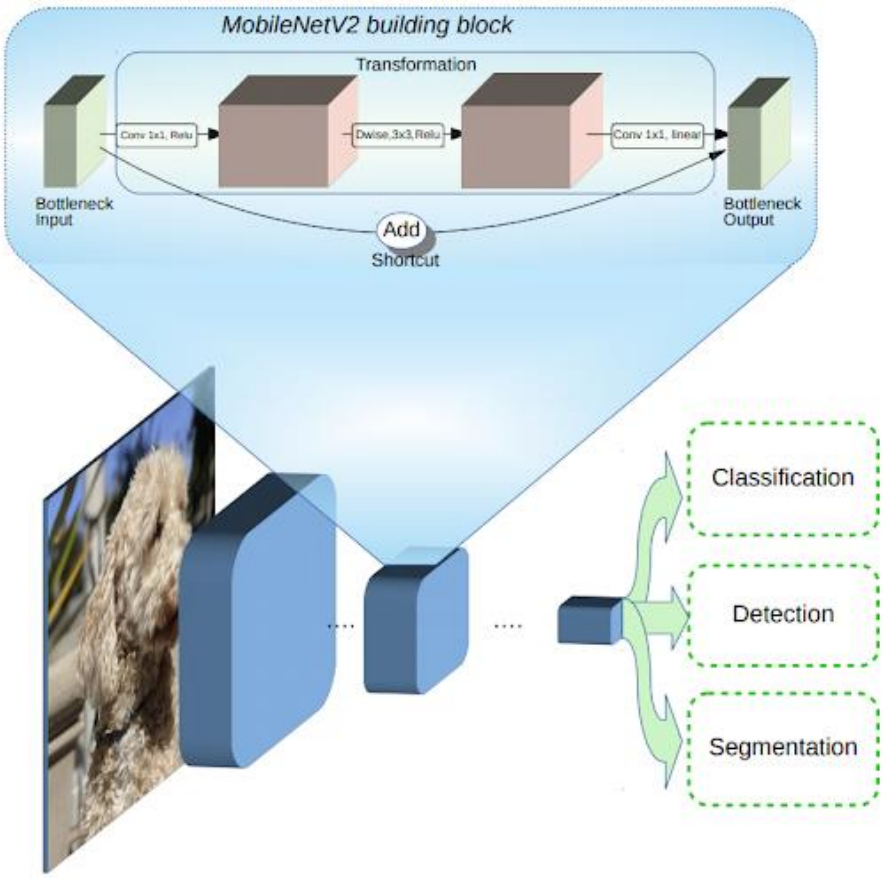
La première étape a consisté à choisir un modèle de classification préentraîné puis mettre en place une structure qui permet au modèle préentraîné de classification ou de détection de fonctionner.

6.2.1.1 Modèle préentraîné

Le modèle de classification préentraîné choisi est MobileNet V2 disponible sur TensorFlow hub (TensorFlow hub 2019).

Comme mentionné au chapitre 3 « Choix technologiques » les critères de sélection sont que les modèles doivent être un réseau de neurones artificiels préentraîné de type classification. Il existe d'autres modèles que MobileNet v2 (TensorFlow hub, 2019), répondant à ces critères comme Inception V3 (TensorFlow hub, 2019), cependant le choix s'est arrêté sur MobilNet v2 car ce modèle est plus rapide que Inception V3 et les autres modèles tout en restant suffisamment performant en termes de précision et de sensibilité.

Figure 17 : Architecture du modèle MobileNet V2



(Sandler, Howard 2018)

6.2.1.2 Structure de détection

La structure permettant au modèle de fonctionner est composée de la classe « Detection » ayant les méthodes publiques suivantes :

run_analyse() : Lance l'analyse d'une ou des images.

detection() :Analyse une image puis détermine (détecte) la classe (catégories d'objets) à laquelle l'image appartient. En fonction de la classe déterminée copie l'image dans le dossier de résultat correspondant. Le dossier de dépôt est vidé au fur et à mesure de la détection des images.

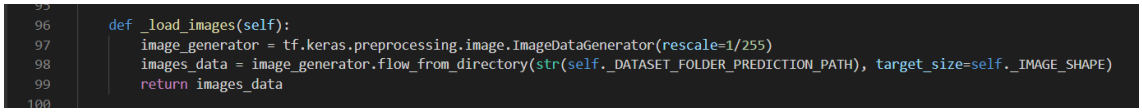
load_model() : Charge le modèle préentraîné de classification ou de détection.

load_images() : Charge l'image en mémoire puis la transforme en vecteur pour. l'analyse

load_images_data() :Formate les données pour l'analyse.

__init__ : Constructeur qui charge la configuration et initialise les attributs de la classe « Detection ».

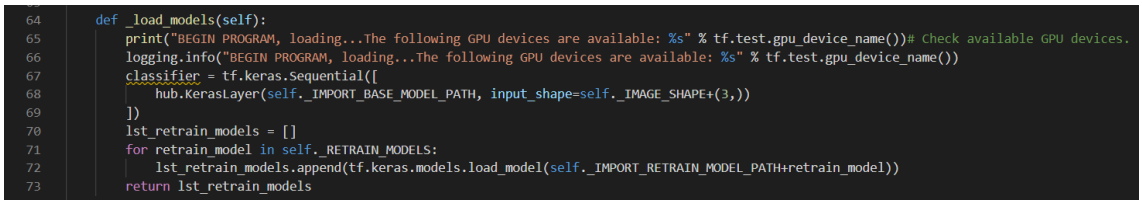
Figure 18 : Code source de la fonction qui charge les images en mémoire



```
    def _load_images(self):
        image_generator = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1/255)
        images_data = image_generator.flow_from_directory(str(self._DATASET_FOLDER_PREDICTION_PATH), target_size=self._IMAGE_SHAPE)
        return images_data
```

(Réalisé à l'aide de l'outil : Visual Studio Code)

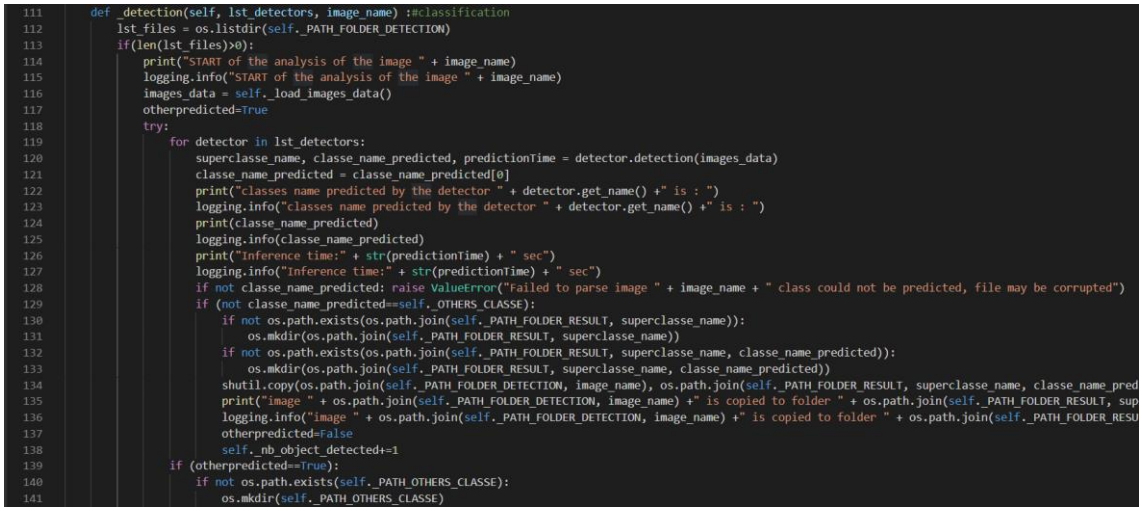
Figure 19 : Code source de la fonction qui charge le ou les modèles en mémoire



```
def _load_models(self):
    print("BEGIN PROGRAM, loading...The following GPU devices are available: %s" % tf.test.gpu_device_name())# Check available GPU devices.
    logging.info("BEGIN PROGRAM, loading...The following GPU devices are available: %s" % tf.test.gpu_device_name())
    classifier = tf.keras.Sequential([
        hub.KerasLayer(self._IMPORT_BASE_MODEL_PATH, input_shape=self._IMAGE_SHAPE+(3,))
    ])
    lst_retrain_models = []
    for retrain_model in self._RETRAIN_MODELS:
        lst_retrain_models.append(tf.keras.models.load_model(self._IMPORT_RETRAIN_MODEL_PATH+retrain_model))
    return lst_retrain_models
```

(Réalisé à l'aide de l'outil : Visual Studio Code)

Figure 20 : Extrait du code source de la méthode qui trie les images dans différents sous-dossiers en fonction des prédictions du modèle



```
def _detection(self, lst_detectors, image_name) :#classification
    lst_files = os.listdir(self._PATH_FOLDER_DETECTION)
    if(len(lst_files)>0):
        print("START of the analysis of the image " + image_name)
        logging.info("START of the analysis of the image " + image_name)
        images_data = self._load_images_data()
        otherpredicted=True
        try:
            for detector in lst_detectors:
                superclasse_name, classe_name_predicted, predictionTime = detector.detection(images_data)
                classe_name_predicted = classe_name_predicted[0]
                print("classes name predicted by the detector " + detector.get_name() +" is : ")
                logging.info("classes name predicted by the detector " + detector.get_name() +" is : ")
                print(classe_name_predicted)
                logging.info(classe_name_predicted)
                print("Inference time:" + str(predictionTime) + " sec")
                logging.info("Inference time:" + str(predictionTime) + " sec")
                if not classe_name_predicted: raise ValueError("Failed to parse image " + image_name + " class could not be predicted, file may be corrupted")
                if (not classe_name_predicted==self._OTHERS_CLASSE):
                    if not os.path.exists(os.path.join(self._PATH_FOLDER_RESULT, superclasse_name)):
                        os.mkdir(os.path.join(self._PATH_FOLDER_RESULT, superclasse_name))
                    if not os.path.exists(os.path.join(self._PATH_FOLDER_RESULT, superclasse_name, classe_name_predicted)):
                        os.mkdir(os.path.join(self._PATH_FOLDER_RESULT, superclasse_name, classe_name_predicted))
                    shutil.copy(os.path.join(self._PATH_FOLDER_DETECTION, image_name), os.path.join(self._PATH_FOLDER_RESULT, superclasse_name, classe_name_pred
                    print("image " + os.path.join(self._PATH_FOLDER_DETECTION, image_name) +" is copied to folder " + os.path.join(self._PATH_FOLDER_RESULT, sup
                    logging.info("image " + os.path.join(self._PATH_FOLDER_DETECTION, image_name) +" is copied to folder " + os.path.join(self._PATH_FOLDER_RESU
                    otherpredicted=False
                    self._nb_object_detected+=1
            if (otherpredicted==True):
                if not os.path.exists(self._PATH_OTHERS_CLASSE):
                    os.mkdir(self._PATH_OTHERS_CLASSE)
```

(Réalisé à l'aide de l'outil : Visual Studio Code)

6.2.2 2e étape – transfer learning

La deuxième étape a nécessité de réentraîner partiellement le modèle préentraîné de classification en appliquant le principe de transfert de connaissance (transfer learning) à l'aide de la librairie TensorFlow (TensorFlow 2020d). Pour ce faire, il a fallu développer un script permettant de réentraîner un modèle puis adapter le code de la structure.

Figure 21 : Code source de la fonction qui effectue le réentrainement de la dernière couche du modèle

```
91
92      def _transfert_learning(self, images_data, model):
93          print("Begin training of feature extractor layer (Transfer learning)")
94          logging.info("Begin training of feature extractor layer (Transfer learning)")
95          model.compile(
96              optimizer=tf.keras.optimizers.Adam(),
97              loss='categorical_crossentropy',
98              metrics=['acc'])
99          class CollectBatchStats(tf.keras.callbacks.Callback):
100             def __init__(self):
101                 self.batch_losses = []
102                 self.batch_acc = []
103             def on_train_batch_end(self, batch, logs=None):
104                 self.batch_losses.append(logs['loss'])
105                 self.batch_acc.append(logs['acc'])
106                 self.model.reset_metrics()
107         steps_per_epoch = np.ceil(images_data.samples/images_data.batch_size)
108         batch_stats_callback = CollectBatchStats()
109         history = model.fit_generator(images_data, epochs=2,#training here
110                                       steps_per_epoch=steps_per_epoch,
111                                       callbacks = [batch_stats_callback])
112         print("End training of feature_extractor_layer")
113         logging.info("End training of feature_extractor_layer")
114         return model
```

(Réalisé à l'aide de l'outil : Visual Studio Code)

Figure 22 : Code source de la fonction qui extrait la dernière couche du modèle

```
77      def _extractor_layer(self, images_data, classifier, image_batch):
78          feature_extractor_layer = hub.KerasLayer(self._PATH_FEATURE_EXTRACTOR, input_shape=(self._SHAPE,self._SHAPE,3))
79          feature_batch = feature_extractor_layer(image_batch)
80          print(" feature extractor layer : ")
81          logging.info(" feature extractor layer : ")
82          print(feature_batch.shape)
83          logging.info(feature_batch.shape)
84          feature_extractor_layer.trainable = False #Freeze the variables in the feature extractor layer, so that the train
85          classifier = tf.keras.Sequential([
86              feature_extractor_layer,
87              layers.Dense(images_data.num_classes, activation='softmax')
88          ])
89          classifier.summary()
90          return classifier
```

(Réalisé à l'aide de l'outil : Visual Studio Code)

6.2.3 3e étape – tests et optimisation

La troisième étape a essentiellement été composée de tests visant à augmenter les performances de la détection afin d'atteindre les objectifs fixés (70% de précision, 98% de sensibilité). À la suite de ces tests, de nombreux cas potentiellement problématiques ont été détectés :

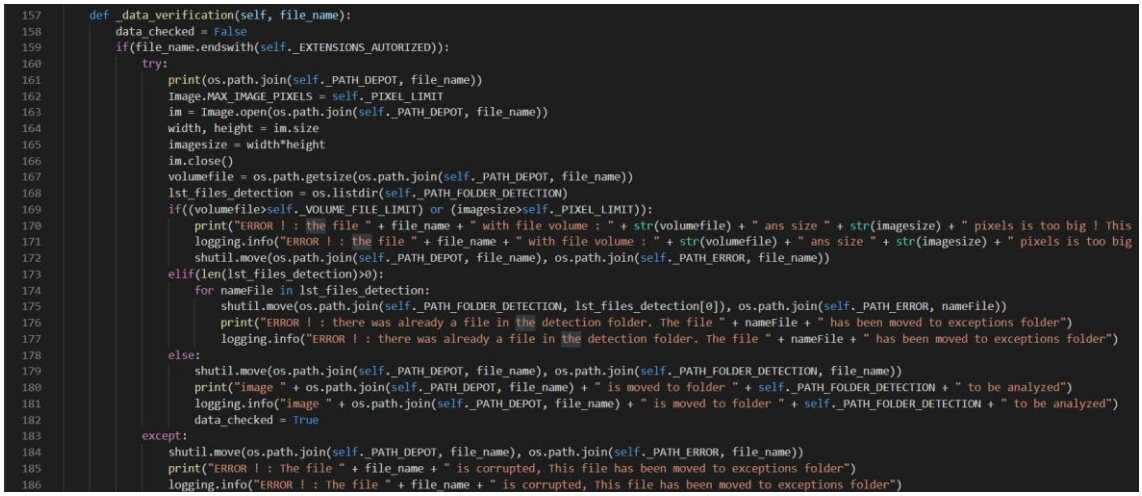
- Extension de fichiers non gérés par TensorFlow
- Fichiers corrompus
- Fichiers trop petits ou trop grands (taille en bytes et en pixels)

Pour prévenir tout crash de l'application et éviter une mauvaise classification, ces fichiers ne sont pas analysés, mais déplacés dans un dossier « exceptions » qui doit faire l'objet d'une vérification manuelle afin d'éviter de manquer des informations.

Il faut savoir que les données qui vont être soumises au programme proviennent de données sur des téléphones portables. Il peut s'y trouver des fichiers de types très variés et pouvant potentiellement poser problème.

Afin de tester les limites du modèle de classification j'ai soumis au programme des images de petite et de grande taille, voir très grande taille comme la carte du monde de la NASA faisant 21'000x21'000 pixels. À partir de 3000x3000 pixels l'image commence à devenir trop grande pour un traitement efficace, il faut savoir que l'image est redimensionnée en 220x220 pixels avant d'être traitée, l'algorithme de redimensionnement fait qu'il y une perte d'informations importante à partir d'une image 10 à 13 fois plus grande que l'image redimensionnée. Le choix a été fait de déplacer dans le dossier "exceptions" les images de plus 2000x2000 pixels afin de rester prudent. Concernant les petites images le problème reste le même que pour un traitement manuel, dès que l'image est trop pixélisée les objets figurant dessus ne sont plus reconnaissables. Rencontrer des images qui sont dans ce cas de figure est peu probable.

Figure 23 : Code source de la fonction de vérification des données entrantes



```
157     def _data_verification(self, file_name):
158         data_checked = False
159         if(file_name.endswith(self._EXTENSIONS_AUTORIZED)):
160             try:
161                 print(os.path.join(self._PATH_DEPOT, file_name))
162                 Image.MAX_IMAGE_PIXELS = self._PIXEL_LIMIT
163                 im = Image.open(os.path.join(self._PATH_DEPOT, file_name))
164                 width, height = im.size
165                 imagesize = width*height
166                 im.close()
167                 volumefile = os.path.getsize(os.path.join(self._PATH_DEPOT, file_name))
168                 lst_files_detection = os.listdir(self._PATH_FOLDER_DETECTION)
169                 if((volumefile>self._VOLUME_FILE_LIMIT) or (imagesize>self._PIXEL_LIMIT)):
170                     print("ERROR ! : the file " + file_name + " with file volume : " + str(volumefile) + " ans size " + str(imagesize) + " pixels is too big ! This
171                     logging.info("ERROR ! : the file " + file_name + " with file volume : " + str(volumefile) + " ans size " + str(imagesize) + " pixels is too big
172                     shutil.move(os.path.join(self._PATH_DEPOT, file_name), os.path.join(self._PATH_ERROR, file_name))
173                 elif(len(lst_files_detection)>0):
174                     for nameFile in lst_files_detection:
175                         shutil.move(os.path.join(self._PATH_FOLDER_DETECTION, lst_files_detection[0]), os.path.join(self._PATH_ERROR, nameFile))
176                         print("ERROR ! : there was already a file in the detection folder. The file " + nameFile + " has been moved to exceptions folder")
177                         logging.info("ERROR ! : there was already a file in the detection folder. The file " + nameFile + " has been moved to exceptions folder")
178                 else:
179                     shutil.move(os.path.join(self._PATH_DEPOT, file_name), os.path.join(self._PATH_FOLDER_DETECTION, file_name))
180                     print("image " + os.path.join(self._PATH_DEPOT, file_name) + " is moved to folder " + self._PATH_FOLDER_DETECTION + " to be analyzed")
181                     logging.info("image " + os.path.join(self._PATH_DEPOT, file_name) + " is moved to folder " + self._PATH_FOLDER_DETECTION + " to be analyzed")
182                     data_checked = True
183             except:
184                 shutil.move(os.path.join(self._PATH_DEPOT, file_name), os.path.join(self._PATH_ERROR, file_name))
185                 print("ERROR ! : The file " + file_name + " is corrupted, This file has been moved to exceptions folder")
186                 logging.info("ERROR ! : The file " + file_name + " is corrupted, This file has been moved to exceptions folder")
```

(Réalisé à l'aide de l'outil : Visual Studio Code)

L'aspect sécuritaire n'est pas géré en dehors des fichiers pouvant provoquer un crash du système. Si les données traitées contiennent des malwares ou du code nuisible, le système sera exposé avant même le démarrage de l'application. L'entier du code source est directement lisible au travers des fichiers Python, il n'y a pas de code caché ou d'exécutable binaire en dehors des modèles qui contiennent des fichiers générés par

TensorFlow. (TensorFlow 2020d) De plus, le mandant n'a pas d'exigence particulière concernant la sécurité en dehors du fait que l'application ne doit pas être connectée au réseau.

6.2.4 4e étape – gestion des exceptions, logs et traitements concurrents

La 4e étape a consisté à rendre l'application plus stable en y incorporant une gestion des exceptions et un système de logs pour pouvoir dépanner l'application en cas de panne ou d'erreur lors de l'exécution. Un système de threads (fils d'exécution) a également été implémenté afin que l'interface graphique puisse être mise à jour durant l'exécution de l'analyse. Les threads sont directement dépendants de l'interface graphique. Il n'a pas été jugé utile d'ajouter un manager pour faire le lien entre les deux, car la librairie (PyQt5 2020) ne laisse pas la possibilité de le faire sans avoir recours à des threads directement dépendants de cette dernière.

Figure 24 : Extrait du log de l'application

```
2020-02-24 14:36:10,573 - INFO - classes name predicted by the detector 1581421885_argent is :
2020-02-24 14:36:10,573 - INFO - Autres
2020-02-24 14:36:10,573 - INFO - Inference time:0.048865556716918945 sec
2020-02-24 14:36:10,612 - INFO - classes name predicted by the detector 1581413511_drogue is :
2020-02-24 14:36:10,612 - INFO - Autres
2020-02-24 14:36:10,612 - INFO - Inference time:0.039454936981201172 sec
2020-02-24 14:36:10,614 - INFO - image ./data/prediction/detection\1200px-AugstmatthornSchnee.jpg is copied to folder ./data/resultat/Autres
2020-02-24 14:36:10,616 - INFO - image ./data/prediction/detection\1200px-AugstmatthornSchnee.jpg is deleted of detection folder
2020-02-24 14:36:10,616 - INFO - END analysis of the image 1200px-AugstmatthornSchnee.jpg
2020-02-24 14:36:10,622 - INFO - image ./data/depot\1200px-Bauhaus_August_2006_UK.jpg is moved to folder ./data/prediction/detection to be analyzed
2020-02-24 14:36:10,623 - INFO - START of the analysis of the image 1200px-Bauhaus_August_2006_UK.jpg
2020-02-24 14:36:10,735 - INFO - Image batch shape: (1, 224, 224, 3)
2020-02-24 14:36:10,736 - INFO - Label batch shape: (1, 1)
2020-02-24 14:36:10,782 - INFO - classes name predicted by the detector 1581945558_armes is :
2020-02-24 14:36:10,782 - INFO - Pistolet
2020-02-24 14:36:10,782 - INFO - Inference time:0.04642820358276367 sec
2020-02-24 14:36:10,787 - INFO - image ./data/prediction/detection\1200px-Bauhaus_August_2006_UK.jpg is copied to folder ./data/resultat\Armes\Pistolet
2020-02-24 14:36:10,830 - INFO - classes name predicted by the detector 1581421885_argent is :
2020-02-24 14:36:10,831 - INFO - Autres
2020-02-24 14:36:10,831 - INFO - Inference time:0.04288458824157715 sec
2020-02-24 14:36:10,862 - INFO - classes name predicted by the detector 1581413511_drogue is :
2020-02-24 14:36:10,862 - INFO - Autres
2020-02-24 14:36:10,862 - INFO - Inference time:0.03091597557067871 sec
2020-02-24 14:36:10,863 - INFO - image ./data/prediction/detection\1200px-Bauhaus_August_2006_UK.jpg is deleted of detection folder
2020-02-24 14:36:10,863 - INFO - END analysis of the image 1200px-Bauhaus_August_2006_UK.jpg
2020-02-24 14:36:10,872 - INFO - image ./data/depot\1200px-Booklet-vega.png is moved to folder ./data/prediction/detection to be analyzed
2020-02-24 14:36:10,873 - INFO - START of the analysis of the image 1200px-Booklet-vega.png
2020-02-24 14:36:10,985 - INFO - Image batch shape: (1, 224, 224, 3)
2020-02-24 14:36:10,985 - INFO - Label batch shape: (1, 1)
2020-02-24 14:36:11,023 - INFO - classes name predicted by the detector 1581945558_armes is :
2020-02-24 14:36:11,023 - INFO - Autres
2020-02-24 14:36:11,023 - INFO - Inference time:0.036915540695190430 sec
2020-02-24 14:36:11,059 - INFO - classes name predicted by the detector 1581421885_argent is :
2020-02-24 14:36:11,059 - INFO - Autres
2020-02-24 14:36:11,059 - INFO - Inference time:0.03490591049194336 sec
```

(Visualisé à l'aide de l'outil : Notepad++)

6.3 Interface graphique

La dernière étape du développement a été de mettre au point une interface graphique permettant à l'utilisateur final d'utiliser facilement l'application. Cette interface est conçue pour être simple et facile d'utilisation conformément à la demande du mandant. Elle répond directement aux besoins analysés au chapitre 2 "Analyse des besoins".

L'outil utilisé pour concevoir l'interface est QT Creator (QT 2020), le choix s'est porté vers cette librairie, car elle est supportée par plusieurs langages dont C++ et Python. Elle permet de déployer l'interface en C++ pour pouvoir embarquer l'interpréteur Python afin de simplifier l'installation de l'application. Pour des raisons de manque de temps et

pour simplifier le développement, la solution choisie a été d'utiliser la librairie PyQT (PyQT5 2020) pour convertir l'interface QT en Python.

6.3.1 Composition de l'interface graphique

L'interface graphique est composée d'une simple et unique fenêtre. Elle permet en termes de fonctionnalité de :

1) Charger des fichiers dans le dépôt d'images.

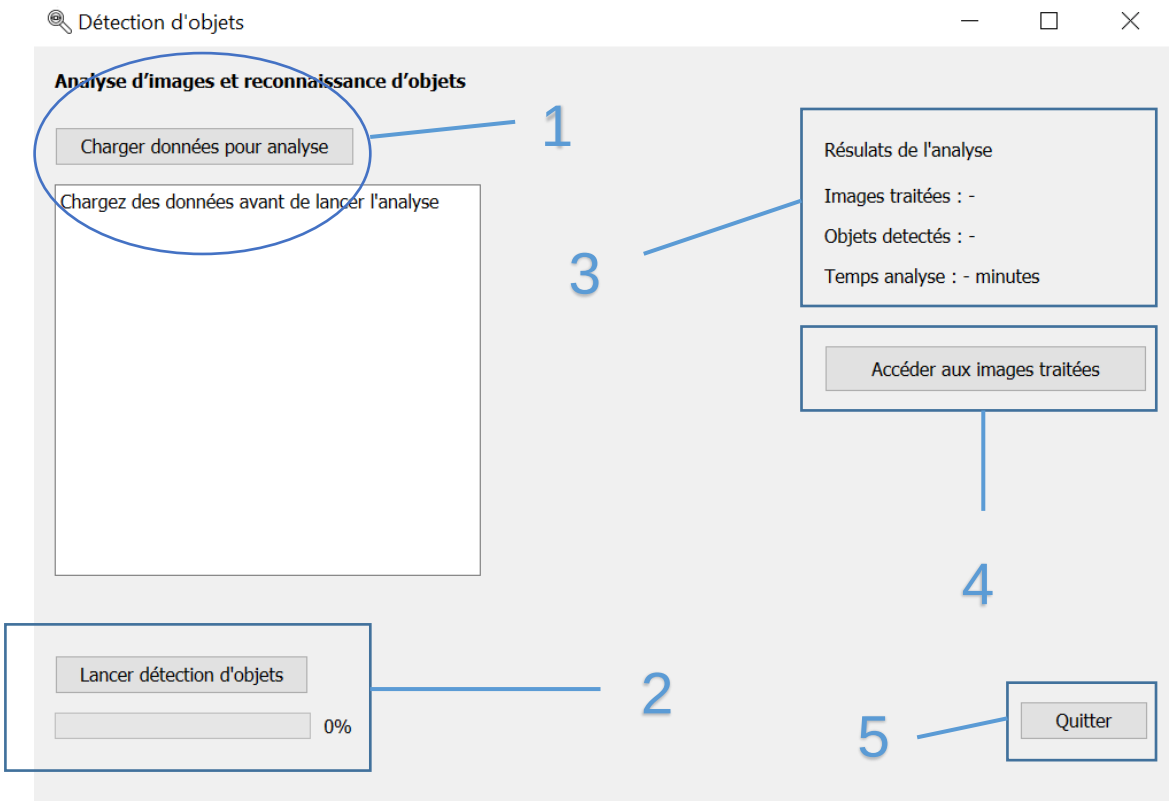
2) Lancer l'analyse (affiche la progression durant analyse)

3) Afficher les statistiques relatives aux résultats

4) Afficher et accéder aux résultats

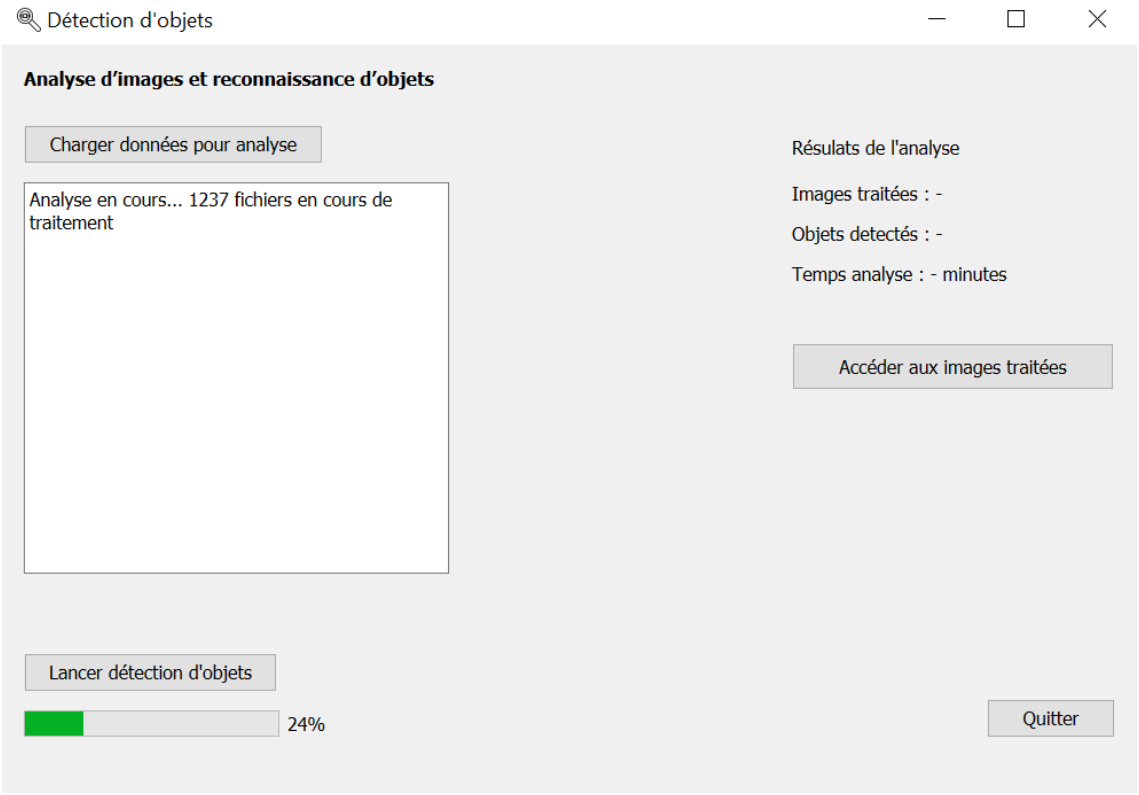
5) Quitter l'application

Figure 25 : Interface graphique au démarrage du programme



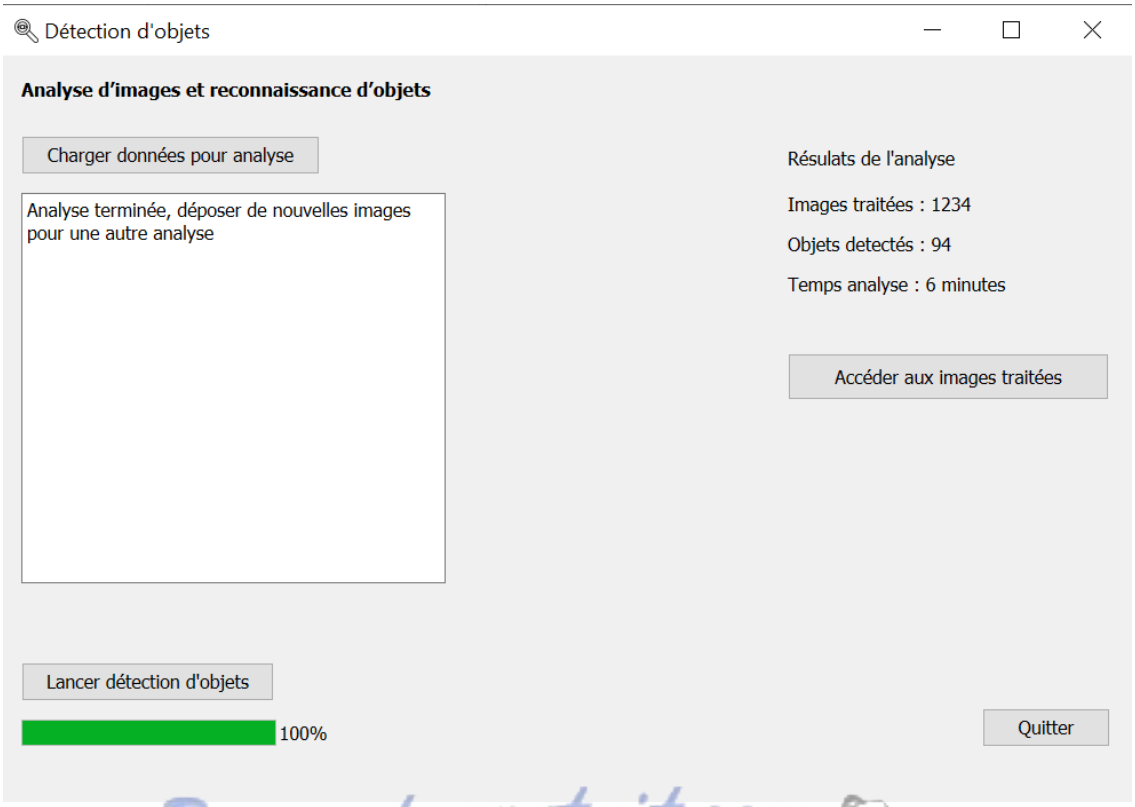
(Réalisé à l'aide de l'outil : Visual Studio Code)

Figure 26 : Interface graphique du programme pendant l'exécution de l'analyse



(Réalisé à l'aide de l'outil : Visual Studio Code)

Figure 27 : Interface graphique du programme après l'analyse



(Réalisé à l'aide de l'outil : Visual Studio Code)

L’icône de l’application (figure ci-dessous) a été réalisée à l’aide du logiciel Gimp 2.10.14 en modifiant une image libre de droits, téléchargée sur le site Flaticon.com (Flaticon 2020)

Figure 28 : Icône de l’application

Réalisé à l’aide de l’outil : Gimp 2.10.14

6.4 Normes et conventions

Le code respecte les principes de l’orienté objet et les bonnes pratiques de programmation PEP8 conseillés par Python Software Fondation (Rossum 2013) également appliqués par TensorFlow (TensorFlow 2.0 2019).

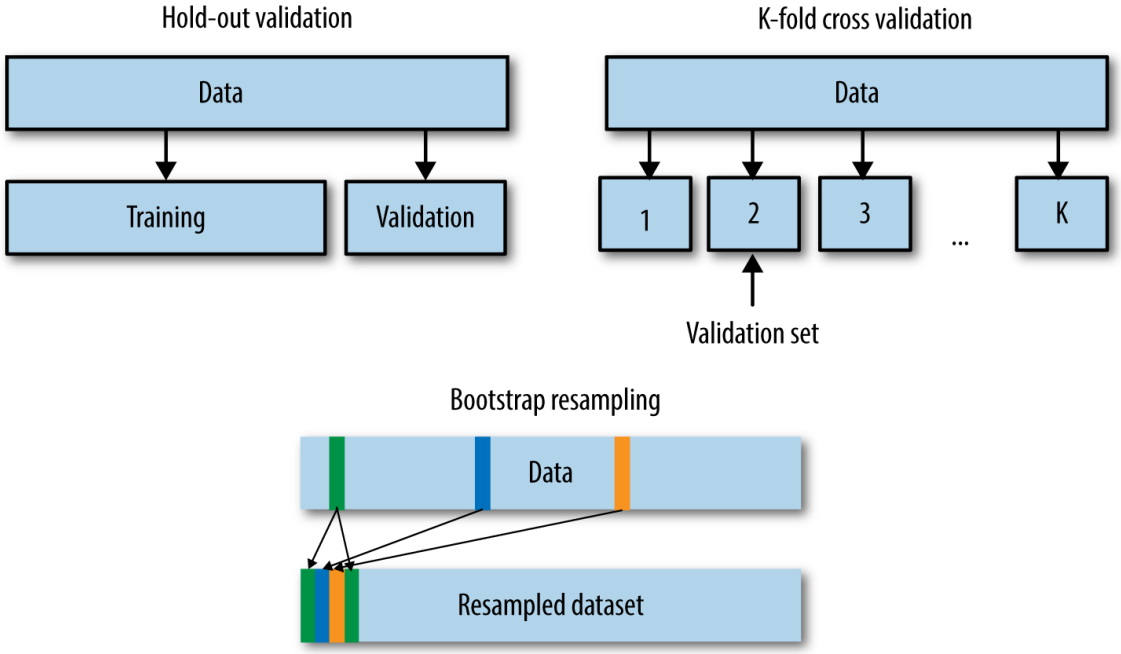
7. Tests et performances

7.1 Méthode de validation

Selon (Zheng 2015), la méthode de validation d’un modèle d’apprentissage automatique la plus simple est appelée « Hold-out ». Elle consiste à utiliser la majorité des données à disposition pour l’entraînement et le reste pour la validation.

Selon (Zheng 2015), il existe d’autres méthodes de validation comme la validation croisée par k fold consistant à diviser les données en plis pour l’entraînement et la validation. La méthode bootstrap permet de faire du rééchantillonnage et d’utiliser les mêmes données pour l’entraînement et la validation.

Figure 29 : Méthode de validation



(Zheng 2015)

Dans notre cas c'est la méthode de validation « Hold-out » qui a été choisie pour sa simplicité. Les autres méthodes répondent à des problématiques qui ne nous concernent pas directement comme le manque de données ou une répartition déséquilibrée de ces dernières.

Tableau 2 : Répartition des données de validation

Données initiales – 14'000 images	
Données entraînement	Données de validation
9500 images	4500 images

Ci-dessous le résultat du test effectué avec le jeu de données de validation :

Tableau 3 : Jeu de données de validation

Jeu de données de validation	
Quantité	4500
Qualité	Format de 200x200 à 2000x2000 pixels, aucune donnée corrompue.
Données négatives (objets non recherchés)	paysages, personnes, voitures, trains, véhicules, ordinateurs, meubles, vêtements, animaux de compagnie, fêtes, nature, nourriture : 4300 images
Données positives (objets recherchés)	armes, drogues, argent : 200 images
Résultats	Précision 96.84%, Sensibilité 99.00% 2 faux négatifs, 140 faux positifs
Temps analyse	15 minutes (I7 8565u, Intel Graphics HD 620, RAM 16GB, SSD, Windows 10)

7.2 Résultats des tests

Pour mettre au point les jeux de données de tests, les banques de données en ligne n'ont pas été utilisées, car elles ont le défaut d'être très homogènes et d'avoir des données d'une qualité homogène. Elles ne sont donc pas représentatives des données que le programme aura à traiter lors de son utilisation finale.

Pour les objets non recherchés (données négatives) la composition de données se veut représentative des données qui seront rencontrées lors de l'utilisation du logiciel, c'est-à-dire des images qui proviennent de téléphones portables et d'ordinateurs personnels. Cependant faute d'avoir réussi à trouver des études sérieuses sur le sujet, on ne peut garantir que ces données sont bel et bien représentatives. Afin de pallier ce problème et diminuer le risque d'avoir un jeu de test peu représentatif, plusieurs jeux de données de taille et de composition variées ont été rassemblés. Cette démarche permet d'avoir un

aperçu plus vaste du comportement du programme en fonction des données qui lui sont présentées.

Parmi les objets non recherchés, la présence de données corrompues est utile afin de tester la capacité du système à éliminer ce type de données afin qu'elles ne faussent pas les mesures. Ces dernières n'ont pas été prises en compte dans le calcul de la précision ou de la sensibilité. L'utilisateur final devra vérifier le dossier d'exception afin d'éviter que des données partiellement corrompues, mais pouvant fournir des informations soient mises de côté.

Ci-dessous, les résultats des tests sous forme de tableau :

Tableau 4 : Jeu de données de test 1

Jeu de données de test 1	
Quantité	7800
Qualité	Format de 200x200 à 2000x2000 pixels, une petite partie des données corrompues.
Données négatives (objets non recherchés)	paysages, personnes, voitures, trains, véhicules, ordinateurs, meubles, vêtements, animaux de compagnie, fêtes, nature, nourriture : 3400 images
Données positives (objets recherchés)	armes, drogues, argent : 4400 images
Résultats :	Précision 96.5%, Sensibilité 97.93% 98 faux négatifs, 81 faux positifs, 176 exceptions
Temps analyse	33 minutes (I7 8565u, Intel Graphics HD 620, RAM 16GB, SSD, Windows 10)
Conclusion	Lorsque les données contiennent des objets recherchés à la fois variés et en nombre, la sensibilité baisse. Dans ce cas, la variété d'objets recherchés est grande (forme, position dans l'image, contexte) ce qui explique une sensibilité plus basse. La majorité des faux négatifs sont

	prédits dans le domaine des drogues. Concernant la précision elle est plus haute que prévu, cependant il y a plus d'objets détectés, mais mal classés.

Tableau 5 : Jeu de données de test 2

Jeu de données de test 2	
Quantité	1237 soit le nombre de photos en moyenne sur les téléphones portables suisses selon (Villinger 2019)
Qualité	Format de 200x200 à 2000x2000 pixels, une petite partie des données corrompues.
Données négatives (objets non recherchés)	paysages, personnes, blagues (mème), fleurs, groupes de musique, selfies, immeubles, animaux de compagnie, nourriture : 1217 images
Données positives (objets recherchés)	armes, drogues, argent : 20 images
Résultats :	Précision 93.83%, Sensibilité 100% 0 faux négatif, 74 faux positifs, 17 exceptions
Temps analyse	6 minutes (I7 8565u, Intel Graphics HD 620, RAM 16GB, SSD, Windows 10)
Conclusion	Les images de nourriture font baisser la précision, car elles sont dans certains cas difficilement différentiables des images contenant de la drogue. Les guitares électriques peuvent être interprétées comme des armes.

Tableau 6 : Jeu de données de test 3

Jeu de données de test 3	
Quantité	10’000
Qualité	Format de 200x200 à 2000x2000 pixels, une partie des données corrompues.
Données négatives (objets non recherchés)	Paysages, blagues, humour, véhicules, animaux de compagnie, concerts, fêtes, matériaux, meubles, montagnes, nourriture, oiseaux, selfies, photos de vacances : 9970 images
Données positives (objets recherchés)	armes, drogues, argent : 30 images
Résultats :	Précision 93.33%, Sensibilité 100% 0 faux négatifs, 642 faux positifs, 335 exceptions
Temps analyse	31 minutes (I7 8565u, Intel Graphics HD 620, RAM 16GB, SSD, Windows 10)
Conclusion	Avec un grand nombre de données la précision reste stable. Cependant, bien que cela ne soit pas proportionnel le nombre de faux positifs augmente, ce qui donne la fausse impression que la précision a chuté de manière drastique. Un cas d’arme a été révélé par le programme alors qu’il avait échappé à ma vigilance lors de la préparation des données.

7.3 Cas limites et curiosités

Lors des tests certains cas sont apparus révélant les limites du système. Plusieurs sont notables, car ils révèlent des objets recherchés là où il n’est pas certain qu’un humain les aperçoit. Lors des tests certains objets qui avaient échappé à ma vigilance pendant la préparation des données de tests ont été détectés par le programme. Ce fut le cas d’un montage à but humoristique avec un fusil de chasse.

Cela fut aussi le cas d'armes dissimulées comme les bagues revolver ou encore les fusils intégré à un meuble ou maquillé en outils de travail comme des perceuses. Selon la position de l'arme dans l'image un rapide coup d'œil ne suffit pas à un humain pour les voir, c'est le cas par exemple de certaines armes portées à une ceinture.

Figure 30 : Bague revolver

Source : HAUTE MACABRE, 2020. [en ligne]. [Consulté le 18.02.2020]. Disponible à l'adresse : https://hautemacabre.tumblr.com/post/39950227494/new-on-haute-macabre

Figure 31 : Armes à feu à moitié dissimulées

Source : LLOD, 2020. SUPER SECRET GUNS! Hidden in plain sight, with Tactical Walls [en ligne]. [Consulté le 18.02.2020]. Disponible à l'adresse : https://gunstreamer.com/watch/super-secret-guns-hidden-in-plain-sight-with-tactical-walls_fYe9HUM7DOeCP1W.html

Bien évidemment si ces armes étaient complètement dissimulées le programme ne pourrait y reconnaître aucune forme à moins de prendre en compte le contexte sans l'objet recherché.

À l'inverse des cas cités précédemment, le programme est très mauvais pour faire la différence entre de la cocaïne et de la farine sur la seule base d'une photographie, au même titre qu'un humain le serait sans doute Une image ne suffit pas forcément à déterminer la nature d'un objet qui s'y trouve.

Si on veut trouver des objets qui sont soit difficilement détectables dans une image ou en partie dissimulés, il faut se servir du contexte en fournissant des données d'entraînement prenant en compte le contexte, voire créer une catégorie (classe) exclusivement pour cette raison. Le détecteur d'objets pourrait être amené à devenir un détecteur de contexte. Cependant cette possibilité à ses limites, car fatalement la précision viendra à baisser.

8. Déploiement

8.1 Prérequis

Prérequis à l'installation du programme :

- Logiciel : Un système d'exploitation récent (Windows 8 et 10, Ubuntu 18.04, macOS 10.9 Mavericks)
- Matériel : Un processeur I5 8^{e} génération, 4GB de mémoire, une carte graphique (Minimum Intel HD 620)

Le programme est portable et multiplateforme grâce à l'interpréteur Python. Il est théoriquement possible de l'installer sur tous les systèmes supportant Python 3.7 et le driver graphique CUDA toolkit.

8.2 Installation de l'environnement d'exécution

Pour pouvoir exécuter le programme il est nécessaire d'installer Python 3.7, puis d'installer les librairies requises avec les bonnes versions. (Détails dans le document annexe « Manuel d'installation de l'application »). Cette étape est délicate, car le moindre écart de version peut amener à des conflits difficiles à résoudre.

8.3 Installation du programme

Une fois l'environnement d'exécution installé, pour installer le programme il suffit de décompresser le fichier zip de déploiement puis de placer le dossier principal du programme à l'emplacement voulu.

Il est possible sur la majorité des systèmes d'exploitation d'ajouter un raccourci vers le fichier run.pyw afin de rendre l'exécution plus facile et conviviale.

8.4 Exécution du programme

Pour exécuter le programme, il suffit de double cliquer sur le fichier run.pyw ou de lancer ce dernier en invite de commande. (Détails dans le document annexe « Manuel d'utilisation de l'application »).

9. Conclusion

La conception puis le développement furent très instructifs, j'ai appris beaucoup sur le langage Python et surtout sur la librairie TensorFlow qui est une merveille d'ingéniosité et d'ingénierie.

Le plus difficile fut le tiraillement constant entre ressources à disposition (en termes de temps et de connaissances) et l'objectif de parvenir à la fin du projet dans le temps imparti en respectant les exigences de la HEG concernant le travail de Bachelor et la volonté de toujours améliorer le logiciel pour répondre au cahier de charges de manière complète et pertinente.

Les domaines de l'intelligence artificielle et de l'apprentissage automatique (machine learning) sont fascinants, mais peuvent s'avérer incroyablement complexes. Je m'intéresse au sujet depuis plusieurs semestres et je n'ai pas de difficulté en mathématiques et en statistiques, cependant je dois avouer que je suis encore loin d'avoir les compétences pour réaliser moi-même un réseau de neurones exploitable par une entreprise. Il a fallu prendre le temps de bien choisir les technologies utilisées sur des critères qui font baisser le risque de ne pas parvenir à atteindre les objectifs fixés au départ tout en prenant le temps de se renseigner sur les technologies actuelles.

Ce travail de Bachelor m'a conforté dans mon idée que lorsqu'une solution existe déjà et qu'elle est disponible, il faut s'en servir plutôt que de vouloir réinventer quelque chose d'existant pour gagner un peu de performance qui n'est pas requise pour la réussite du projet. Il est probable que si j'avais voulu construire mon propre modèle de classification par mes propres moyens, je n'aurais jamais pu finir le projet ni même parvenir à quelque

chose de plus efficace que les réseaux de neurones développés et paramétrés par les ingénieurs de Google (Google AI 2020).

Au début du projet, j'ai commis une erreur en sous-estimant le temps consacré à la mise en place des outils de développement et à la prise en main de la librairie TensorFlow 2.0 (TensorFlow 2020a). J'ai une dizaine de projets informatiques menés à bien et professionnels à mon actif et je n'ai jamais rien vu d'aussi difficile à prendre en main bien qu'incroyablement accessible au vu de la complexité du sujet.

9.1 Perspectives

9.1.1 Réentraînement du modèle

Pour améliorer les performances, la principale possibilité consiste à récolter et sélectionner judicieusement des données pour un nouvel entraînement du modèle. La sélection et le prétraitement des données sont des étapes cruciales. Il est nécessaire de définir un inventaire des objets qu'on veut détecter puis réunir les images présentant les objets à détecter sous des formes et des angles de vue variés. Il est aussi conseillé d'utiliser des images au format varié. Enfin, il faudra mener des tests puis adapter les données d'entraînement en conséquence jusqu'à parvenir au résultat souhaité.

9.1.2 Ajout de nouvelles catégories

Comme mentionné dans le chapitre 2 « Analyse des besoins » il est possible de réentraîner facilement le modèle en rassemblant des données et en exécutant un script de « Transfer learning » sans oublier de modifier les paramètres en conséquence.

Le modèle peut donc être complété pour traiter de nouvelles catégories elles-mêmes divisées en sous-catégories d'un même type d'objet. Il existe une limite à l'ajout de nouvelles catégories, car plus le nombre de catégories augmentera, plus le temps de traitement augmentera. Les catégories peuvent également devenir des contextes plutôt que des objets enfin d'augmenter la sensibilité ou rechercher des images avec certaines caractéristiques autres que la présence d'objets particuliers. Cependant cette démarche aura probablement comme conséquence de faire baisser la précision du modèle.

9.1.3 Ajout d'un détecteur ou d'un nouveau classificateur

Pour augmenter l'efficacité de l'application, une piste intéressante serait de rajouter des classificateurs ou des détecteurs d'objets afin d'augmenter la sensibilité ou la précision. La structure du code et l'algorithme permettent de lancer une seconde analyse durant la première.

Le code respecte les principes de l'orienté objet et les bonnes pratiques de programmation PEP8 conseillées par Python Software Fondation (Rossum 2013) et

également appliquées par TensorFlow (TensorFlow 2.0 2019). Par conséquent il ne devrait pas être trop difficile pour un développeur de remplacer le classificateur par un détecteur ou d'y ajouter de nouveaux traitements.

9.1.4 Mise en ligne et application distante

Il est possible d'adapter le programme pour le mettre sur un serveur et le déployer comme application distante, voire de créer une API. Cependant cela ajoute une contrainte de sécurité considérable qu'il faudrait absolument gérer. La possibilité de mettre l'application sur un serveur dédié ou mieux sur le service en ligne de Google AI Platform (Google AI 2020) permettrait d'augmenter les capacités de traitement de l'application tout en la rendant plus accessible.

9.1.5 Traitement des vidéos

Le traitement des vidéos n'a pas été implémenté faute de temps. Il ne devrait pas être difficile de rajouter cette fonctionnalité supplémentaire, car les vidéos sont en grande partie constituées d'images. La partie son ne pourra malheureusement pas être traitée par le programme.

Bibliographie

BARDOS, M, ZHU, W.H, 1997. Comparaison de l'analyse discriminante linéaire et des réseaux de neurones. Application à la détection de défaillance d'entreprises. [en ligne], France. Numdam. [Consulté le 11.02.2020]. Disponible à l'adresse : http://www.numdam.org/article/RSA_1997__45_4_65_0.pdf

BARREIRO LINDO, Flávio, 2018. Interprétation d'images basée sur la technologie des réseaux de neurones [en ligne]. Genève : Haute école de gestion de Genève. Travail de Bachelor. [Consulté le 30.10.2019]. Disponible à l'adresse : http://doc.rero.ch/record/323738

BLONDELLE, Mélina, 2018. Sondage : quels sont les meilleurs langages de programmation pour l'IA ?. [en ligne]. France. Développer.com. [Consulté le 16.02. 2020]. Disponible à l'adresse : https://www.developpez.com/actu/213780/Sondage-quels-sont-les-meilleurs-langages-de-programmation-pour-l-IA-Quels-sont-vos-criteres/

BASTIEN L., 2019. Réseau de neurones artificiels : qu'est-ce que c'est et à quoi ça sert ?. [en ligne]. France. Lebigdata.fr [Consulté le 11.02.2020]. Disponible à l'adresse : https://www.lebigdata.fr/reseau-de-neurones-artificiels-definition

BING , 2019. Moteur de recherche en ligne [en ligne]. Étas-unis. Microsoft [Consulté le 17.10.2019]. Disponible à l'adresse : http://www.bing.com/images

CROWLEY, James L., 2001. Reconnaissance de Forme Statistique. [en ligne]. France. École nationale supérieure d'informatique et de mathématiques appliquées. [Consulté le 11.02.2020]. Disponible à l'adresse : http://www-prima.imag.fr/Prima/Homepages/jlc/Courses/2000/ENSI2.RF/ENSI2.RF.S1.pdf

CHOLLET, François, 2017. *Deep Learning with Python* [en ligne]. Édition 1. Étas-unis. O'Reilly Media, Inc. [Consulté le 11.02.2020]. ISBN: 9781617294433. Disponible à l'adresse : https://learning.oreilly.com/library/view/deep-learning-with/9781617294433/ [accès par abonnement]

CONFÉDÉRATION SUISSE, 2019. Loi fédérale sur les armes, les accessoires d'armes et les munitions. [en ligne]. Loi sur les armes, LArm du 20 juin 1997 (État le 14 décembre 2019). Suisse. [Consulté le 15.02.2020]. Disponible à l'adresse : https://www.admin.ch/opc/fr/classified-compilation/19983208/index.html

CONFÉDÉRATION SUISSE, 2020a. Loi fédérale sur les stupéfiants et les substances psychotropes. [en ligne]. Loi sur les stupéfiants, LStup du 3 octobre 1951 (État le 1er février 2020). Suisse. [Consulté le 15.02.2020]. Disponible à l'adresse : https://www.admin.ch/opc/fr/classified-compilation/19981989/index.html

CONFÉDÉRATION SUISSE, 2020b. Loi fédérale sur l'unité monétaire et les moyens de paiement. [en ligne]. LUMMP du 22 décembre 1999 (État le 1er janvier 2020). Suisse. [Consulté le 15.02.2020]. Disponible à l'adresse : https://www.admin.ch/opc/fr/classified-compilation/19994336/index.html

CUDA, Nvidia, 2019. CUDA Toolkit Archive. [en ligne]. Étas-unis. Google. [Consulté le 16.02.2020]. Disponible à l'adresse : https://developer.nvidia.com/cuda-toolkit-archive

DUGERDIL, Philippe, 2018. Module 625-1 Organisation du développement-GL [document PDF]. Support de cours : Cours module 625-1 «Organisation du développement logiciel», Haute école de gestion de Genève, filière Information de gestion, année académique 2018-2019

FLATICON 2020, Analysis free icon. [en ligne]. Étas-unis. Freepik Company. [Consulté le 05.02.2020]. Disponible à l'adresse : https://www.flaticon.com/free-icon/analysis_249164

GOUVERNEMENT FRANÇAIS, Commission d'enrichissement de la langue française, 2018. Vocabulaire de l'intelligence artificielle. [en ligne]. France. Légifrance. [Consulté le 11.02.2020]. Disponible à l'adresse : https://www.legifrance.gouv.fr/jo_pdf.do?id=JORFTEXT000037783813

GÉRON, Aurélien, 2019. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. [en ligne]. 2de Édition. États-unis. O'Reilly Media, Inc. [Consulté le 11.02.2020]. ISBN: 9781492032649. Disponible à l'adresse : https://learning.oreilly.com/library/view/hands-on-machine-learning/9781492032632 [accès par abonnement]

GODEC, P., PANCUR, M., ILENIC, N. et al, 2019. Democratized image analytics by visual programming through integration of deep models and small-scale machine learning. Nature commun 10. article number 4551. [Consulté le 08.02.2020]. Disponible à l'adresse : https://www.nature.com/articles/s41467-019-12397-x

GOOGLE API 2020. Open Images Dataset V5 [en ligne]. États-unis, Google [Consulté le 15.01.2020]. Disponible à l'adresse : https://storage.googleapis.com/openimages/web/index.html

GOOGLE AI 2020. AI Platform [en ligne]. États-unis, Google [Consulté le 19.02.2020]. Disponible à l'adresse : https://cloud.google.com/ai-platform?hl=fr

HALE, Jafe, 2018. Deep Learning Framework Power Scores 2018. [en ligne]. États-unis, Towards Data Science. [Consulté le 16.02.2020]. Disponible à l'adresse : https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a

IMAGE NET, 2019. Research updates on improving ImageNet data. [en ligne]. 2016. États-unis. Stanford vision Lab, Standford University, Princeton University [Consulté le 20.10.2019]. Disponible à l'adresse : http://www.image-net.org/

ISMAILI, Zakariyaa, 2019. Top 10 des Librairies de Deep Learning sur Python. [en ligne]. France. le-datascientist.fr [Consulté le 24.12.2019]. Disponible à l'adresse : https://le-datascientist.fr/top-10-des-librairies-de-deep-learning-sur-Python

MÜLLER Andreas C, GUIDO Sarah, 2017. *Introduction to Machine Learning with Python*. [en ligne]. États-unis, O'Reilly Media, Inc. [Consulté le 11.02.2020]. ISBN: 9781449369415. Disponible à l'adresse : https://learning.oreilly.com/library/view/introduction-to-machine/9781449369880/ [accès par abonnement]

MAGDA, Gregorova, 2018. Course 9 - 02 Classification performance measures [document Jupyter]. Support de cours : Cours « Data mining », Haute école de gestion de Genève, filière Information de gestion, année académique 2018-2019

MONTAIGU, Matthieux, 2018. Comment construire un modèle de reconnaissance de produits sur mesure avec TensorFlow ?. [en ligne]. Italie, artefact.com [Consulté le 28.10.2019]. Disponible à l'adresse : https://artefact.com/fr-fr/news/comment-utiliser-TensorFlow-et-ses-ressources-open-source-pour-construire-un-modele-de-reconnaissance-de-produit-sur-mesure/

MODULE 656, 2019. Descriptif de module 656. [document PDF]. Document de référence pour Travail de Bachelor. Haute école de gestion de Genève, filière Information de gestion, année académique 2018-2019

PRADEEP, Pujari, Md. RAZAUL Karim, MOHIT, Sewak, 2018. Practical Convolutional Neural Networks. [en ligne]. Édition 1. États-unis. O'Reilly Media, Inc. [Consulté le 11.02.2020]. ISBN: 9781788392303. Disponible à l'adresse : https://learning.oreilly.com/library/view/practical-convolutional-neural [accès par abonnement]

PASSOS, Alex, BAILEY, Paige, 2019. TensorFlow 2.0 and Keras (#AskTensorFlow) [enregistrement vidéo]. *Youtube* [en ligne]. 3 juillet 2019. [Consulté le 29.12.2019]. Disponible à l'adresse : https://www.youtube.com/watch?v=wGI_VtE9CJM

PYQT5, 2020. Python bindings for the Qt cross platform application toolkit. [en ligne]. Royaume-Uni. Riverbank Computing. [Consulté le 02.01.2020]. Disponible à l'adresse : https://pypi.org/project/PyQt5/

QT, 2020. One framework. One codebase. Any platform. [en ligne]. Espoo, Finlande. The QT Compagny. [Consulté le 30.12.2019]. Disponible à l'adresse : https://www.qt.io/

ROSSUM, Guido van, 2013. PEP 8 -- Style Guide for Python Code. . [en ligne]. États-unis. Python Software Foundation. [Consulté le 14.02.2020]. Disponible à l'adresse : https://www.Python.org/dev/peps/pep-0008/

SANDLER Mark, HOWARD, Andrew, 2018. MobileNetV2: The Next Generation of On-Device Computer Vision Networks. [en ligne]. États-unis. Google AI blog. [Consulté le 14.02.2020]. Disponible à l'adresse : https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html

TANG, T.T., ZAWASKI, J.A., FRANCIS, K.N. et al. 2019. Image-based Classification of Tumor Type and Growth Rate using Machine Learning: a preclinical study. [en ligne]. Scientific report 9. article number : 12529 [Consulté le 11.02.2020]. Disponible à l'adresse : https://www.nature.com/articles/s41598-019-48738-5

TENSORFLOW HUB, 2019. [TF2] Imagenet (ILSVRC-2012-CLS) classification with MobileNet V2. . [en ligne]. États-unis. Google. [Consulté le 10.12.2019]. Disponible à l'adresse : https://tfhub.dev/google/tf2-preview/mobilenet_v2/classification/4

TENSORFLOW 2.0, 2019. An end-to-end open source machine learning platformV2. . [en ligne]. États-unis. Google. [Consulté le 13.11.2019]. Disponible à l'adresse : https://www.TensorFlow.org/

TENSORFLOW, 2020a. TensorFlow White Papers. [en ligne]. États-unis. Google. [Consulté le 15.02.2020]. Disponible à l'adresse : https://www.TensorFlow.org/about/bib

TENSORFLOW, 2020b. GPU support. [en ligne]. États-unis. Google. [Consulté le 15.02.2020]. Disponible à l'adresse : https://www.TensorFlow.org/install/gpu?hl=it

TENSORFLOW, 2020c. Install TensorFlow with pip. [en ligne]. États-unis. Google. [Consulté le 08.12.2019]. Disponible à l'adresse : https://www.tensorflow.org/install/pip?lang=python3

TENSORFLOW, 2020d. Transfer learning with TensorFlow Hub. [en ligne]. États-unis. Google. [Consulté le 27.12.2019]. Disponible à l'adresse : https://www.tensorflow.org/tutorials/images/transfer_learning_with_hub

ULMO J., 1973. Différents aspects de l'analyse discriminante. [en ligne]. France. Numdam. [Consulté le 11.02.2020]. Disponible à l'adresse : http://www.numdam.org/article/RSA_1973__21_2_17_0.pdf

VILLINGER, Sandro, 2019. Snap Happy: Avast reveals which countries store the most photos. [en ligne]. République tchèque. Avast. [Consulté le 05.02.2020]. Disponible à l'adresse : https://blog.avast.com/which-countries-store-the-most-photos

ZHENG, Alice, 2015. *Evaluating Machine Learning Models*. [en ligne]. États-unis. O'Reilly Media, Inc. [Consulté le 18.02.2020]. ISBN: 9781491932445. Disponible à l'adresse : https://learning.oreilly.com/library/view/evaluating-machine-learning/9781492048756/ [accès par abonnement]

Annexe 1 : Manuel d'installation de l'application

1) Prérequis

Prérequis à l'installation du programme :

- Logiciel : Un système d'exploitation récent (Windows 8 et 10, Ubuntu 18.04, macOS 10.9 Mavericks).
- Matériel : Un processeur I5 8^{e} génération, 4GB de mémoire, une carte graphique (Minimum Intel HD 620).

Le programme est portable et multiplateforme grâce à l'interpréteur Python. Il est théoriquement possible de l'installer sur tous les systèmes supportant Python 3.7, la librairie Python TensorFlow 2.0 (TensorFlow 2020a) et le driver graphique CUDA toolkit (CUDA 2019).

2) Installation de l'interpréteur Python

Pour installer Python veuillez-vous référer au manuel d'installation Python correspondant à votre système d'exploitation. Il est fortement conseillé d'utiliser la version 3.7 de Python disponible à l'adresse : https://www.Python.org/downloads/release/Python-370/

Pour pouvoir exécuter le programme il est nécessaire d'installer Python 3.7 avec les librairies contenues dans le fichier requirements.txt

Le plus simple consiste à installer les librairies requises en exécutant la commande suivante : « pip install -r requirements.txt ».

Mise en garde : Cette étape est délicate, car le moindre écart de version peut amener l'application à ne pas fonctionner à cause de conflits de version.

En cas de difficulté avec TensorFlow (Tensorflow 2020c) veuillez-vous référer à la documentation disponible à l'adresse : https://www.tensorflow.org/install/pip?lang=python3

3) Installation des drivers graphiques

Pour des raisons de performance il est conseillé d'installer la librairie CUDA toolkit (CUDA 2019) disponible à l'adresse : https://developer.nvidia.com/cuda-toolkit-archive

Cette étape peut-être délicate en fonction de votre système d'exploitation et de votre carte graphique. En cas de problème, veuillez-vous référer à la documentation de TensorFlow (TensorFlow 2020) disponible à l'adresse : https://www.TensorFlow.org/install/gpu?hl=it

2) Installation du détecteur d'objets

Une fois l'environnement d'exécution en place, pour installer le programme il suffit de décompresser le fichier zip de déploiement, puis de placer le répertoire principal du programme à l'emplacement voulu.

Il est possible sur la majorité des systèmes d'exploitation d'ajouter un raccourci vers le fichier de lancement de l'application run.pyw afin de rendre l'exécution plus facile et conviviale. Modifier les chemins d'accès dans le fichier « config.ini » en fonction de votre système d'exploitation.

5) Désinstallation

Pour désinstaller l'application, il suffit de supprimer le répertoire principal dans lequel elle se trouve.

Annexe 2 : Manuel d'utilisation de l'application

1) Prérequis

Pour pouvoir exécuter le programme, il est nécessaire d'avoir au préalable installé l'application avec son environnement d'exécution. La procédure d'installation est décrite dans le document intitulé « Manuel d'installation de l'application ».

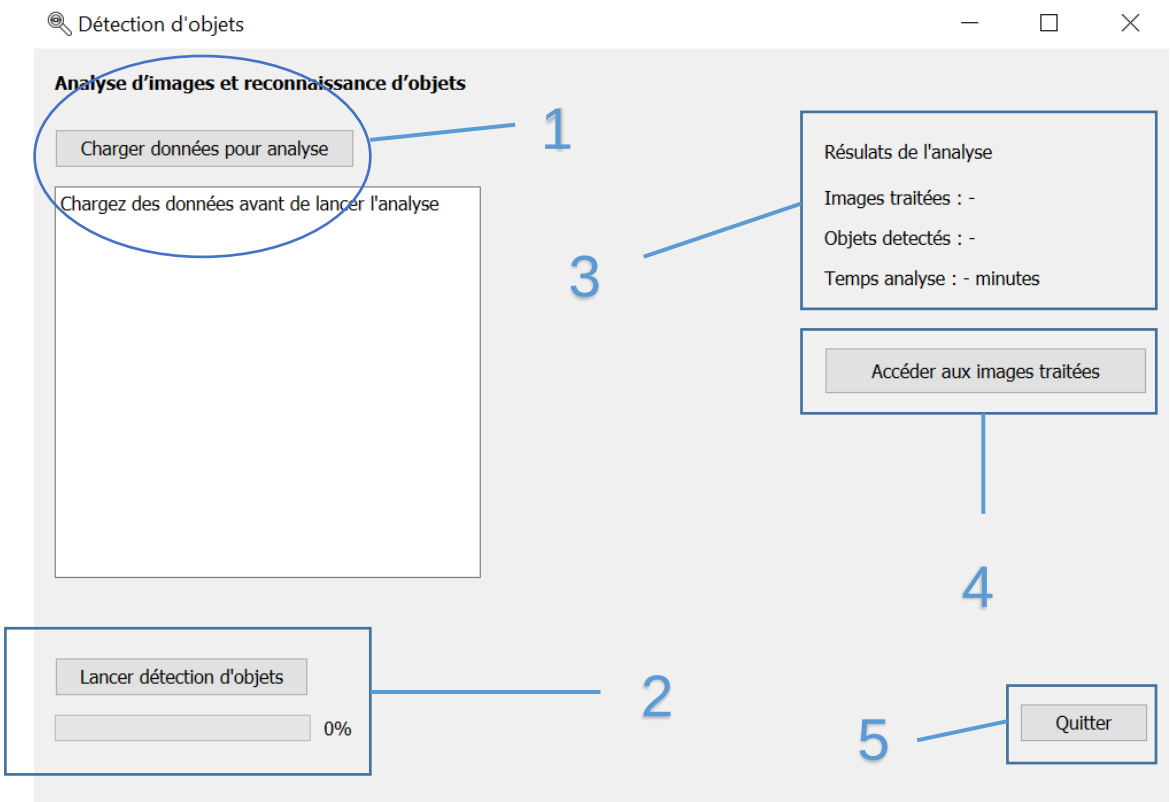
2) Description de l'application

L'application est composée d'une simple et unique fenêtre.

Fonctionnalités :

1. Charger des fichiers dans le dépôt d'images.
2. Lancer l'analyse (affiche la progression durant analyse).
3. Afficher les statistiques relatives aux résultats.
4. Afficher et accéder aux résultats.
5. Quitter l'application.

Figure 32 : Interface graphique au démarrage du programme



(Réalisé à l'aide de l'outil : Visual Studio Code)

3) Démarrer le programme

Pour lancer l'application, il suffit de double cliquer sur le fichier run.pyw se trouvant dans le répertoire principal de l'application.

Il est également possible d'exécuter le fichier via une console ou un terminal avec la commande suivante : « python3 run.pyw »

4) Lancer une analyse

a. Chargement des données

Avant de lancer une analyse, il est nécessaire de placer les images dans le dossier de dépôt, d'une des manières suivantes :

- Placer manuellement les images dans le dossier de dépôt se trouvant dans le répertoire de dépôt qui est dans le dossier data.
- Cliquer sur le bouton « chargez données pour analyse » puis sélectionner les images à charger. Les images sélectionnées seront copiées vers le dépôt. Vous pouvez répéter l'opération autant de fois que nécessaire avant l'analyse.

Mise en garde : Il est vivement conseillé d'utiliser une copie et non des données originales afin d'éviter toute perte de données. Les données mises dans le dépôt sont déplacées vers les dossiers de résultats après analyse.

b. Lancement de l'analyse

Pour lancer l'analyse, cliquez sur le bouton « lancer l'analyse ». L'application prendra quelques secondes pour s'initialiser puis l'analyse commencera. Le suivi de la progression de l'analyse est possible grâce à la barre de progression.

Note : Le dossier de résultats n'est pas réinitialisé automatiquement, il est nécessaire de libérer le dossier avant chaque analyse.

c. Consultation des résultats

Une fois l'analyse terminée, les statistiques (nombre d'images traitées, nombre d'objets détectés, le temps d'analyse) s'afficheront en haut à droite de l'application. Pour accéder aux résultats, cliquez sur le bouton

« Accéder aux images traitées ». Les résultats sont consultables dans le dossier « résultats » se situant dans le dossier « data ».

5) Arrêter l'application

Cliquer sur le bouton arrêter pour quitter l'application. Les données contenues dans le dossier « résultats » ne seront ni sauvegardées ni supprimées.

En cas de panne, il est possible de consulter les fichiers logs de l'application se trouvant dans le même dossier que l'application. Le code source est directement disponible par les fichiers exécutables Python.

Mise en garde : L'application effectue un tri des données à la manière d'un filtre, cependant le fait qu'une image soit classée dans une catégorie ne signifie pas que l'objet est nécessairement présent sur l'image. Le fait qu'un objet pouvant ressembler à celui recherché initialement soit catégorisé d'une certaine manière ne détermine en rien sa nature. Un deuxième tri humain est donc nécessaire.

Annexe 3 : Manuel de mise à jour du modèle de classification

La mise à jour et l'ajout d'une nouvelle catégorie d'objets sont possibles en trois étapes :

1. Réunir un set de données d'entraînement.
2. Exécuter le script de réentraînement.
3. Modifier la configuration en conséquence.

L'ajout d'une nouvelle catégorie entraîne un temps d'analyse plus grand, à moins de réentraîner entièrement le modèle, ce qu'il est également possible de faire.

Étape 1 : Réunir un set de données d'entraînement

La première étape consiste à réunir un set de données d'entraînement. Cela nécessite de réunir une grande quantité et variété d'images. Il est également nécessaire de prendre en compte la qualité des données afin d'obtenir une précision et une sensibilité satisfaisantes. Pour plus d'informations, veuillez-vous référer au chapitre 5.2 « Prétraitement et préparation des données ».

Les données récoltées doivent être placées dans le dossier « dataset_training » se trouvant dans le dossier « data ». Il est nécessaire que les données soient réparties par catégorie (classe). Pour ce faire, il suffit de créer des sous-dossiers portant le nom des catégories.

Étape 2 : Exécuter le script de réentraînement

Pour exécuter le script de réentraînement, vous pouvez exécuter le fichier « script_transferlearning.py » en ligne de commande ou cliquer sur le fichier.

Le réentraînement peut prendre plusieurs heures en fonction de la quantité des données d'entraînement et de la puissance de votre ordinateur.

Étape 3 : Modifier la configuration en conséquence

Une fois le réentraînement fini, vous trouverez toutes les informations utiles afin de mettre à jour la configuration de l'application dans le fichier log « transfer_learning.log ».

Le modèle réentrainé est enregistré dans le dossier « models\retrained_models\saved_models ».

Vous pouvez au préalable le renommer si vous le jugez utile.

Pour modifier la configuration, il faut ouvrir le fichier « config.ini » se trouvant dans le répertoire principal de l'application.

Il est nécessaire de mettre à jour les champs suivants :

RETRAIN_MODELS : Liste des modèles séparés par le caractère « , ».

CLASSES_NAMES : Liste des superclasses/classes séparées par le caractère « : ».

Il est très important de respecter l'ordre des superclasses/classes et des modèles. À chaque modèle correspond une superclasse séparée des classes par les caractères « : ». Les classes sont séparées entre elles par le caractère « , ». Le non-respect de cette syntaxe entraînera une erreur lors du chargement de l'application.

Exemple de configuration :

Modèle : 1581425002_armes

Superclasse : armes

Classes : fusil, pistolet, carabine, couteau_combat

Annexe 4 : Document de vision

Objectifs du document

Le but de ce document est de définir l'objectif et la portée du projet. Il sert de document de référence pour toutes les parties prenantes. Il définit le contexte, les besoins et fonctionnalités de haut niveau de la solution. Le document se concentre exclusivement sur les informations nécessaires et utiles aux parties prenantes et aux utilisateurs cibles.

Limite et portée

Ce projet constitue un travail de Bachelor réalisé dans le cadre de l'examen final de la Haute école de gestion de Genève, en vue de l'obtention du titre Bachelor of Science HES-SO en informatique de gestion. L'objectif du projet est de proposer une solution automatisée aux besoins de la Police cantonale de l'État de Genève en ce qui concerne l'analyse d'images et de reconnaissance d'objets afin d'effectuer un premier tri automatisé des images en leur possession. L'application est prévue pour un usage conforme au droit suisse.

Références

Documentation

- Travail de Bachelor : HAMOUTI Aurélien, 2020 « Analyse d'images et reconnaissance d'objets ».
- Manuel d'installation de l'application.
- Manuel d'utilisation.
- Manuel de mise à jour.

Procès-verbaux

- PV réunion avec le mandant : 01.10.2019.
- PV réunion avec le mandant : 12.11.2020.
- PV réunion avec le mandant : 29.01.2020.

Positionnement

Opportunité commerciale

La Police cantonale de Genève économiserait du temps et potentiellement des ressources humaines lors de l'analyse d'images au format électronique.

Position du problème

Le problème	La tâche de trier manuellement des milliers d'images provenant de téléphones portables ou d'ordinateurs personnels est chronophage.
Affecte	Les enquêteurs de la Police cantonale de Genève
L'impact du problème est	Perte de temps et de ressources humaines
Une solution satisfaisante serait	Une solution automatisée permettant d'effectuer un premier tri des images.

Description des intervenants et des utilisateurs

Les intervenants

Nom	Description	Rôle
M. David Billard	Responsable de l'encadrement du travail de Bachelor	Encadrement, directeur de travail de Bachelor

Nom	Description	Rôle
M. Sébastien Capt	Mandant, membre du Département de la sécurité, de l'emploi et de la santé de l'État de Genève (DSES)	Représente le mandant durant le travail de Bachelor

Utilisateur

Nom	Description	Rôle
-	Enquêteur de la Police cantonale de Genève	utilise la solution

Vue d'ensemble du produit

Le mémoire est composé des parties suivantes :

- Une partie théorique
- La solution sous forme d'un programme exécutable
- La documentation liée à la solution

Caractéristiques essentielles du produit

- Détecte la présence d'objet sur des images au format électronique
- Effectue un tri des images en fonction de la catégorie des objets détectés

Copyright

« Le mémoire du travail de bachelor appartient à la HEG qui se réserve le droit de le diffuser en entier ou en partie. » extrait du plan modulaire du travail de bachelor (Module 656 2019)

Cette annexe s'inspire du cours 625-1 « Organisation du développement logiciel » donné par Philippe Dugerdil, Professeur HES à la Haute école de gestion de Genève (Dugerdil 2018)

Annexe 5 : Code source et données

L'entier du code source se trouve dans un DVD joint au présent travail de bachelor, également disponible sur un dépôt Github à l'adresse suivante : https://github.com/aurelienHamouti/Image-analysis-and-object-recognition

Le dossier contient :

- Un dossier déploiement de l'application intitulé « analyse_images_detection_objets ».
- Un dossier intitulé « outils » contenant le script scraper.
- Un dossier intitulé « documentation » contenant les manuels d'installation, d'utilisation de l'application et de mise à jour du modèle.
- Un dossier intitulé « data » contenant des données à titre d'exemple.