UNIVERSITÉ DE BOURGOGNE

ESIREM

INGÉNIÉRIE DES LOGICIELS ET DES CONNAISSANCES

Systèmes Intelligents avancés

Auteur :
E. GOUINGUENET
J. MASSARD

Enseignant :
Olivier BROUSSE

26 janvier 2021

Table des matières

Table des figures

1 Propos du rapport

Le projet de Systèmes intelligent avancé a été articulé en trois phases.
- — Construction du dataset
- — Choix de la technologie
- — Application direct avec critère de mesure de performance

Dans la première phase nous avons récupéré puis annoté, à l'échelle de la classe entière (28 étudiants) plus de 2600 images. Nous nous sommes assurés de l'unicité de celle-ci via l'utilisation d'un hash de l'image. Les annotations ont été réalisé avec l'outil labelme, créant des fichiers au format XML, un par image. Le format choisi a été celui de PASCAL VOC.

S'en suit la seconde phase, celle qui a le plus de valeur ajouté, pour nous. La première phase est également intéressante sur comment choisir le découpage de la "labelisation", quelles "features" nous souhaitons conservés (et donc détecter), etc... Mais le côté répétitif de l'opération (plus de 200 images pour notre groupe) vient largement nuire au sentiment général vis à vis de cette première étape.

S'en ai suivis une phase de recherche bibliographique et technologique : quelle est la meilleure solution en considérant performance de détection et rapidité d'exécution. De plus nous devions composé avec nos limitation machines.

Enfin, le développement d'une application en ligne de commande, simple mais efficace nous facilitant les tests, et implémentant les services demandés à savoir : l'évaluation à l'aide de la méthode du mAP [3] implémenté par Cartucho [2] ainsi que l'inférence en direct d'un flux vidéo (caméra et/ou téléphone).

Ce rapport détail la vie du projet mais présente aussi les résultats obtenus sur la détection de masque.

2 Vie du projet

La première phase étant identique pour tout le groupe, concentrons le développement de ce rapport sur la seconde.

2.1 Recherches

Nous nous sommes très rapidement concentrés vers des modèles de détection d'objet. Ayant écarter les méthodes de classifications dès le départ bien que celle-ci soit pertinente. De plus nous observions qu'aucune méthode étaient prédominante et que toutes présentait de léger avantages et inconvénients. Une liste non exhaustives des modèles est présent dans le repository github Tensorflow [8].

Pour ce qui est de la plate forme technologique utilisé, n'étant pas familier avec le monde de l'intelligence artificiel et ses librairies. Ne souhaitant pas être perdu, nous avons donc choisi de nous baser sur les maigres bases que nous possédions. D'où le choix de Keras.

Pour ces mêmes raisons nous avons préférez nous basés sur un "repository" github de détection d'objet [4]. Nous servant de socle solide pour démarrer les entraînements.

2.2 Environnements techniques

Malgré la spécification de l'utilisation d'une machine local le jour de l'évaluation. Nous avons néanmoins décidé d'utiliser les ressources mis à disposition par Google via l'outil Google Collab. En effet, nos équipements personnels sont un ordinateur portable ne permettant absolument pas d'entraîner un modèle ainsi qu'un ordinateur relativement puissant (i7-8700K @ 4.9GHz, 32 Go de RAM @ 3600mHz et d'une GTX1060 6Go) mais de la simple navigation web pendant l'entraînement en provoque l'arrêt, ingérable.

Une fois la direction choisie et les premiers essais réalisés, nous sommes passer sur un développement local pour se rapprocher d'une architecture de projet dé-ployable sur la machine qui allait être utilisé lors de l'évaluation. Cela nous permettaient également d'avoir une arborescence de fichier

facilement déployable dans la session collab qui est, pour rappel, supprimé à chaque fin d'utilisation.

2.3 Solution choisie

De part nos recherches nous avons rapidement sélectionner deux solutions : Keras et les nombreux modèles disponibles ainsi que Detectron 2 de Facebook [7]. Les éléments cités plus haut, à savoir nos connaissances de bases autour de keras, nous avons choisi ce dernier.

Non sans peine nous avons découvert les subtiles différences des versions 1.X et 2.X de Tensorflow, qui nous ont coûtés beaucoup de temps précieux passé en débuggage.

2.4 Difficultés rencontrées

Le premier lot de difficultés de ce projet est lié à la réalisation du datasets, en effet, d'autres groupes ne se sont pas contraints au formalisme que nous nous étions tous imposés. Ce qui a eu pour effet de nous contraindre a reformalisé leur contributions dans le dataset commun sous peine de retarder nos premiers tests puisqu'il était alors impossible de charger un dataset dont la formalisation est irrégulière.

Nous avons donc palier a ce soucis en appliquant des traitements sur le dataset avant même d'essayer de le charger. Notamment un système de vérification entre le nombre d'image et de fichier XML, que toutes les images avaient bien un seul et unique XML associés et vis versa, écarter les images ou XML isolés du dataset. Et supprimer les fichiers dont le hash n'était unique.

Ces événements fâcheux ont fortement impacté notre moral de travail vis à vis du projet.

Une fois ces problèmes d'ordre organisationnels réglés, nous avons pu commencer nos tests via Collab. Durant cette phase il était compliqué pour l'outil de naviguer dans l'arborescence de fichiers repartis dans le cloud en un temps acceptable. D'autant plus qu'il était nécessaire d'uploader ces derniers pour y avoir accès et par conséquent de travailler dans un environnement distribué. Cette contrainte nous a fait utiliser des moyens alternatifs pour la gestion et l'utilisation des images telle que la mise en place de dataframe ainsi que de charger les modèles via des fichiers csv généré a partir du dataset lui-même charger dans un dataframe. Tout les développements en rapport avec cette gestions par csv nous a fait gagner beaucoup de temps sur la mise en place du calcule de mAP [6] via Cartucho/mAP [2].

Un autre point de difficulté, plutôt lié à nos machines cette fois-ci, a été les installations fastidieuses des drivers Nvidia spécifiques aux machines learning (Cuda, Nvidia Toolkits, VS_buildtools) et leur configuration dans le PATH système. Difficulté qui a nécessité du temps mais qui en réalité ne représentait aucune avancée concrète sur le projet.

2.5 Solutions mises en place

L'une des premières solutions que nous avons essayés fut un article d'initiation à l'intelligence artificiel qui proposait d'apprendre pas à pas à créer une application détectant les Kangourous [1]. L'analogie plus que frappant avec notre projet nous a intéressé mais cette dernière malgré une publication récente en 2019 et mise à jour depuis était basé sur Tensorflow 1.X et des objets pythons peu évident à manipuler et débugger. Malgré cela nous avons pu lancé ce dernier, commencer a l'entraîner mais les résultats semblait décevant. L'application initial de cette article se basaient sur le dataset COCO et des modèles pré-entrainés sur ce dernier.

Nous avons ensuite pu créer, à partir d'un clonâge de notebook collab entrainé un premier modèle. Qui a rapidement overfit mais avec quelques ajustement nous a procuré des résultats décent [11].

Un troisème repository Github proposait quant à lui la détection de carte électronique type Rasberry/Arduino a partir d'un petit dataset qu'il avait lui même réalisé [5]. Encore une fois la similarité avec notre projet était frappante mais malgré nos efforts les résultats étaient peu probant. Ce dernier utilisait un modèle différent de notre précédent essai et surtout il n'était pas basé sur Keras mais sur Tensorflow pur, avec le niveau de compréhension global que nous avions a ce stade du projet, nous n'avons pas réussi a adapté ce code a nos contraintes. [**1**]

Malgré nos succès relatif, nous étions perdu. Chaque projet que nous lisions n'avait ni la même structure de code, ni la même arborescence système et encore moins la même manière de lancer un entraînement, de charger un modèle ou bien encore d'évaluer ce dernier...

Nous décidons de suivre les tutoriaux suivants :

- Tensorflow : Train and serve a TensorFlow model with TensorFlow Serving [9]
- Tensorflow object detection api : Training Custom Object Detector [10]

provenant tous deux de la documentation officiel de Tensorflow afin de mieux comprendre le fonctionnement interne de la librairie.

Une fois les APIs (TF Keras) bien mieux comprises nous reprenons notre code basé sur le repository de Fizyr [4]. Nous réévaluons les modèles [8],

écartons les modèles SSD a cause de leur résultat trop faible en mAP, , ainsi que le resnet101 pour son temps d'inférence trop élevé qui ne conviendra pas à l'application vidéo final.

3 Résultat

3.1 Résultat de l'entrainement

3.1.1 with_masque

Cette catégorie présente les meilleurs résultats en terme de détection, assez évident étant donné sa représentation écrasante au sein de notre dataset. Notre réseau détermine avec une bonne précision les personnes portant un masque sur tout type de photo.

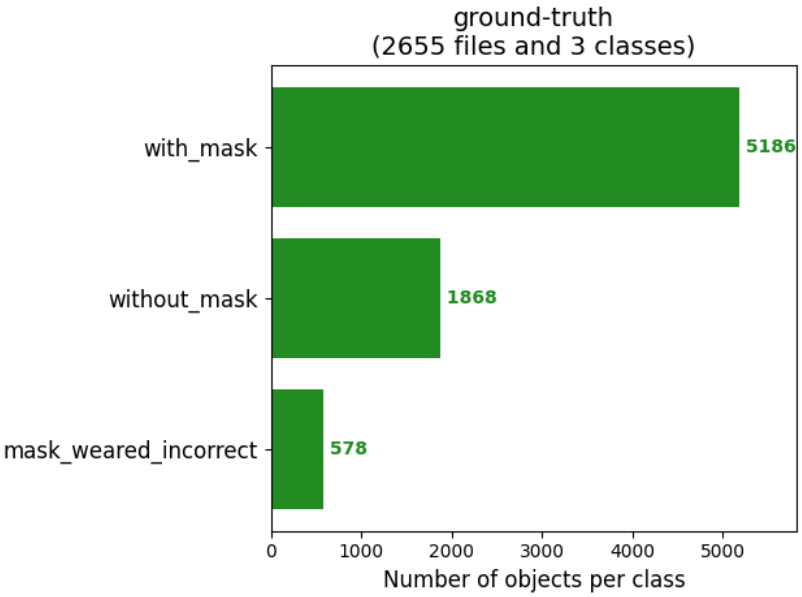

FIGURE 1 – Représentation des classes dans le dataset

3.1.2 without_masque

Opposé à la première cette catégorie, la catégorie without_ est elles aussi bien détecté mais avec nettement moins de précision. Cette classe est bien moins représenté que la précédente et les features à détecter son plus variable e difficile car il s'agit en réalité de l'absence de ces features qui sont masquées par le masque.

3.1.3 mask_weared_incorrect

Pour ceux qui est des masques mals portés, la précision des détections est faible. Ce fait est normal et directement lié au nombre de cas annoté ainsi dans notre dataset d'entraînement.

3.2 Résultats récolté au cours de l'entraînement

FIGURE 2 – Graphe d'entraînement

3.3 Evaluation du modèl, mAP, implémentation Cartucho

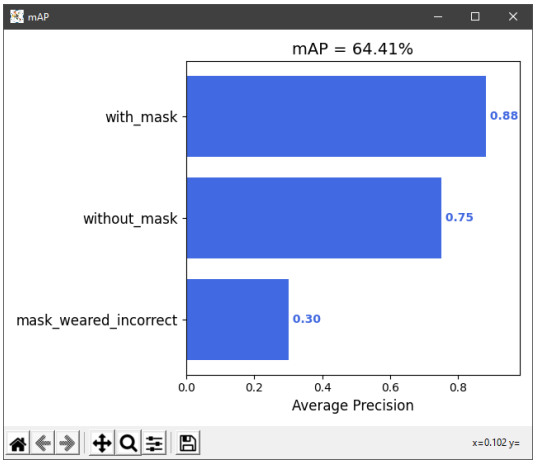

FIGURE 3 – mAP

Bibliographie

[1] Jason BROWNLEE. *How to Train an Object Detection Model with Keras*. 2019. URL : https://machinelearningmastery.com/how-to-train-an-object-detection-model-with-keras/.

[2] CARTUCHO. *mAP (mean Average Precision)*. 2020. URL : https://github.com/Cartucho/mAP.

[3] Christopher DOSSMAN. *Object Detection Accuracy (mAP) Cheat Sheet*. 2019. URL : https://towardsdatascience.com/object-detection-accuracy-map-cheat-sheet-8f710fd79011.

[4] FIZYR. *Keras RetinaNet*. 2020. URL : https://github.com/fizyr/keras-retinanet.

[5] Tanner GILBERT. *How to train a custom object detection model with the Tensorflow Object Detection API*. 2020. URL : https://github.com/TannerGilbert/Tensorflow-Object-Detection-API-Train-Model.

[6] Jonathan HUI. *mAP (mean Average Precision) for Object Detection*. 2018. URL : https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173.

[7] ROBOFLOW. *How to Train a Custom Faster R-CNN Model with Facebook AI's Detectron2 | Use Your Own Dataset*. 2020. URL : https://www.youtube.com/watch?v=4OXntFVfFio.

[8] TENSORFLOW. *Tensorflow Model*. 2020. URL : https://github.com/tensorflow/models.

[9] TENSORFLOW. *Train and serve a TensorFlow model with TensorFlow Serving*. 2020. URL : https://www.tensorflow.org/tfx/tutorials/serving/rest_simple.

[10] TENSORFLOW. *Training Custom Object Detector*. 2020. URL : https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html#.

[11] Venelin VALKOV. *Object Detection on Custom Dataset with TensorFlow 2 and Keras in Python*. 2019. URL : https://www.youtube.com/watch?v=LDQOC8WwzHY.