Haute École d'Ingénierie et de Gestion du Canton de Vaud

Gestion et valorisation de projet de Machine Learning

GML

Crapauduc - 2022

Authors:
Schaller Joris
D'Ancona Olivier
Logan Victoria
Akoumba Erica Ludivine
Wichoud Nicolas

January 5, 2023

Contents

Chapter 1

Introduction

Ce projet de machine learning nous a été proposé dans le cadre du cours de Gestion et valorisation de projet en Machine Learning (GML), donné en 5ème du cursus de Bachelor en Informatique spécialisé en ingénierie des données à la HEIG-VD. Le but de ce travail est de nous faire découvrir la gestion et organisation impliquée par un travail de machine learning, autant au niveau de la recherche technologique qu'au niveau de l'organisation d'équipe, notamment au niveau de la distribution de tâches, gestion d'équipe et de délais.

Le projet étudié ici est un projet existant ayant déjà été réalisé plusieurs fois par le professeur et ses étudiant.e.s : l'étude de crapauducs. Un crapauduc est un "petit conduit sous une route, permettant le passage protégé des batraciens" - selon Le Robert. En 2017, dix-huit crapauducs (*Figure 1.1*) ont étés construits le long de la route d'Aubonne à Gimel - canton de Vaud - afin de permettre aux grenouilles, crapauds et tritons de traverser la route des bois à l'étang en toute sécurité.

À l'intérieur de ces crapauducs ont été installée une caméra équipée d'un capteur, qui implique la prise d'une petite série de photos (*Figure 1.2*) lors de la détection de mouvement, ainsi qu'une planche afin de faciliter la distinction des objets du sol. Comptant moins de caméras que de crapauducs, les caméras n'étaient pas rattachée à un crapauduc et comptent donc des images prises depuis différents crapauducs.

Figure 1.1: Un des 18 crapauducs installés pour l'étude

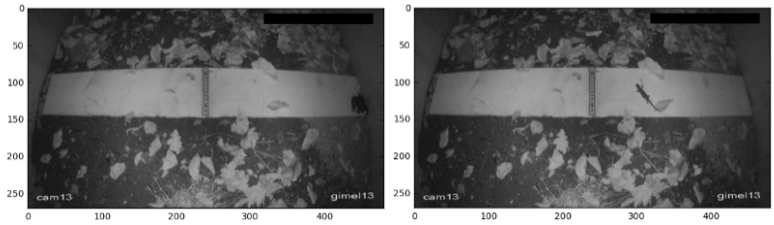

Figure 1.2: Exemples des images d'amphibiens prises par les cameras

Au terme de la première année d'utilisation, ces voies ont été empruntées par plus de 6'000 crapauds, grenouilles et tritons confondus. Ce comptage a été effectué par des chercheurs, qui ont du regarder les images prises par les caméras et compter les animaux "à la main". Le projet Crapauduc vise ainsi à utiliser l'apprentissage automatique (Machine Learning) pour automatiser le comptage des batraciens.

Notre objectif pour ce projet est donc de détecter la présence ou non de grenouille/crapaud et tritons (en considérant les grenouilles et crapauds comme une seule et même catégorie) en utilisant l'apprentissage automatique, ce afin de déterminer si la constructions de ces crapauducs est efficace. Pour se faire, nous avons les prises des caméras du 23 février 2017 au 20 avril de la même année, totalisant près de 1 million d'images, dont on connaît pour chacune la caméra dont elle provient ainsi que la date (YYYY-MM-DD) et l'heure et minute à laquelle elle a été prise.

Enfin, le professeur nous a également mis à disposition sa labelisation de 2020 de ces images, des bounding box pour certaines images, ainsi que les données météos enregistrées durant cette période (notamment la température, le vent, la précipitation, l'humidité).

Chapter 2

outils

But Nous avons constitué un repository github contenant des scripts permettant de transformer les données brutes en données utilisables pour l'analyse. Ces scripts sont disponibles dans le repository utils sur github.

2.1 Colabeler

Afin de réaliser les bounding box et les labels, nous avons utilisé le logiciel Colabeler permettant d'annoter les images pour l'object detection. Ainsi nous pouvons ajouter des bounding box facilement et rapidement. Il a été utilisé dans le cadre de la création du filtre et dans la constitution du dataset de test.

2.2 Conversion de format

Nous avons écrit un petit script python permettant de convertir les labels en différents formats. Il existe plusieurs manières de définir les bounding box. Elle peuvent être définies comme un point d'ancrage et une taille plus une hauteur ou simplement être 2 points. De plus, il existe différente nomenclatures pour stocker ces images telles que le format coco stocké dans un fichier .json qui est associé au dataset coco. Il se peut que les données soient encore stockées sous forme d'un csv ou d'un fichier manifest qui peut être utile pour des services comme amazon sagemaker.

2.3 Comptage des labels

Un petit script a été mis sur pied afin de compter les labels déjà effectués. Ce qui permet d'avoir une liste des images déjà traitées et de constituer un subset rapidement pour entraîner des algorithmes.

2.4 Folder Shortener

Ce script bash permet de simplifier le chemin d'accès aux images pour une question de clarté et d'entretient du projet.

2.5 Fusion des dataframes

Chapter 3

Gestion du projet

3.1 Organisation du projet

Dès le départ, nous avons décider de travailler avec git, plus précisement sur github. Nous avons donc créé une organisation afin de séparer les différents dépots. Nous en avons définis 2, mais les membres de l'organisation étaient libres d'en ajouté d'autres.

- crapauduc. Ce dépôt est le dépot principal où les notebooks des modèles sont déposés, nous y avons aussi placé les rapports des anciens étudiants afin d'y avoir un accès rapide. Nous y avons aussi déposé unun subset d'image d'environ 0.5 Gib permettant le fine tuning.
- utils. Ce dépôt contient des scripts faisant des transformormations ou des analyses sur les données. Nous y avons par exemple un script qui permet de convertir les annotations de csv à COCO.

De plus, nous avons créer un compte google ayant le doux nom de `student GML` afin d'avoir un espace google drive de 15 GiB pour stocker les données ainsi qu'une intégration facilitée dans le service colab de Google. Nous croyions être prêts.

3.2 Gestion du temps de travail

Dès le départ, nous avons décider de travailler à distance afin de dédier la totalité de la journée à ce projet sans perdre de temps dans les transports publiques. En effet, le mardi où tombe le cours de GML, nous n'avons pas d'autre cours que ce dernier. Ainsi, un mardi typique se déroule comme suit:

- 8h00 - 13h15: Libre, mais souvent on prépare la séance de l'après-midi.
- 13h15 - 15h: Appel Teams, où nous expliquons notre avancement, normalement les différents problèmes rencontré doivent être réglé avant la réunion. Planification des tâches pour la prochaine semaine, et répartition des tâches. Durant chaque réunion un membre du groupe prend des notes afin d'avoir un historique des discussions, ce procès verbale des réunions est stocké sur le google drive de `student GML`.

La séance du mardi se résume donc essentiellement à un partage d'information entre les différents groupes de travail composé de 1 à 3 étudiant.e.s. Le travail proprement dit est pour la plupart effectué en dehors des réunions, soit le mardi après la réunion soit à un autre moment choisis par les membres du groupe.

3.3 Gestion des tâches et répartition

Nous avons poussé notre utilisation de github, en gérant nos tâches à l'aide de l'outil de gestion de projet kanban directement intégré dans github. Ainsi, nous pouvons savoir à n'importe quel moment quel membre de l'équipe travail sur quelle partie du projet. De plus, nous pouvons voir les tâches en cours, les tâches terminées, les tâches en attentes, etc.

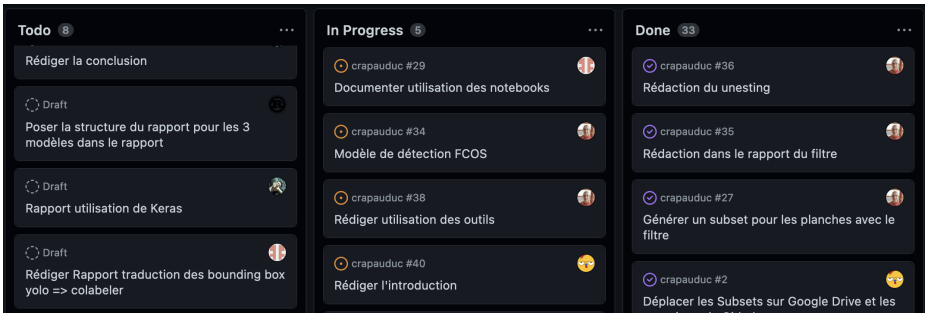

Figure 3.1: Nos tâches du Kanban réparties en 3 catégories : à faire, en cours et terminées.

Chapter 4

Filtrage

Idée générale Le but de ce chapitre est de décrire les différentes méthodes de filtrage investiguées, dans le but d'améliorer la qualité des données.

Problème Le dataset original est composé de 18 caméras regroupant environ 1 million d'images. Une bonne partie de ces images sont des faux positifs. Il est donc nécessaire de filtrer les images afin de ne garder que les images qui nous intéressent. Une première observation nous fait remarquer que les images uniquement constituées de feuilles n'ont jamais d'animaux. Ensuite, une deuxième lecture nous fait remarquer que les animaux se déplacent plus facilement par temps humide. Et finalement, nous constatons que les animaux sont nombreux certains jours. À partir de ces observations, nous avons élaboré 3 méthodes pour filtrer les images et ainsi augmenter notre probabilité de trouver des animaux pour constituer de nouveaux labels ou constituer un dataset de validation. Ces méthodes sont décrites dans les sections suivantes.

4.1 Analyse des données labellisées

Comme dit en introduction, notre professeur monsieur Satizabal Mejia Hector Fabio nous a fourni

un fichier csv contenant les labelisations de 2020 images –¿ NON –¿ investiguer !!!!!! (check avec Joris d'où il a pris les données path_and_bounding_box.csv

4.1.1 Analyse temporelle

4.1.2 Analyse météorologique

4.2 Détecteur de planches

Nous avons développé un réseau de neurones convolutif à l'aide de la libraire PyTorch. Ce classificateur binaire, prédit ou non la présence de planche.

Dataset d'entraînement Nous avons extrait 600 images d'une même caméra et labellisé 359 non planches et 241 planches. Ensuite, nous avons développé un dataloader permettant d'intégrer nos labels et de charger des batchs de données directement dans la libraire PyTorch. Celui ci, utilise un pipeline d'entrée qui applique plusieurs transformations à l'image avant de pouvoir l'utiliser comme un tenseur.

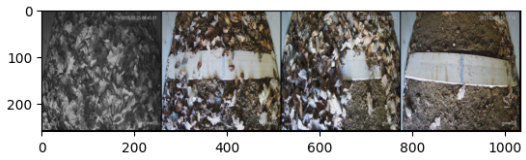

Figure 4.1: Exemple de données d'entraînement

Architecture du Détecteur Le détecteur est simplement constitué de 3 couches convolutives suivi de 2 couches entièrement connectées. Les channels d'entrée et de sortie des couches convolutives est de : 3 - 32, 32 - 64, 64 - 128. Le nombre de neurones des couches fully connected sont de 128 et 1 pour le neurone de sortie. La fonction de coût utilisé est la BCELoss et l'optimiseur est Adam. Le réseau est entraîné pendant 10 epochs avec un learning rate de 0.001 et un momentum de 0.9 sur 3 epochs.

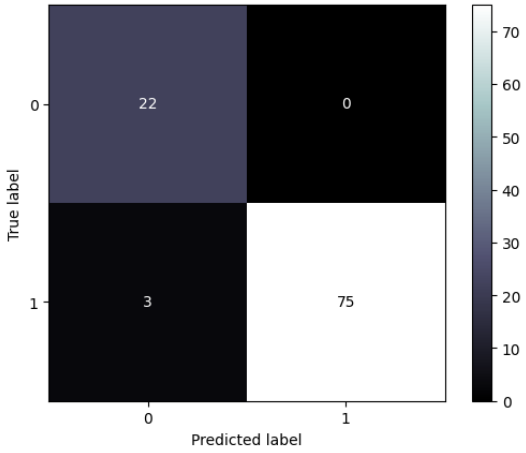

Figure 4.2: Matrice de confusion du détecteur de planche

Résultats Le détecteur de planche a une précision de 1 et un recall de 0.98 sur la détection de planche. En revanche, la précision sur la détection de non planche est de 0.88 et un recall de 1. Ce qui veut dire que notre filtre est un peu trop efficace et a tendance à se tromper pour détecter les images sans planche. Comme les résultats sont satisfaisant pour dégrossir le travail, nous n'avons pas passé de temps supplémentaire à optimiser le réseau afin qu'il sépare mieux les images dotés d'une planche ou non. Comme, nous traitons une grande quantité de données, l'erreur est acceptable. Lancé sur la quasi intégralité du dataset, le filtre a tourné pendant plus de 10h sur un ordinateur de bureau doté d'un processeur Ryzen9500X. Au final, le filtre a détecté 48910 images de non planches sur les 754543 images analysées.

Chapter 5

Modèles

5.1 Choix des modèles

Le choix des modèles à été un choix rapide. Nous avons commencé par regarder les tutoriaux sur les sites web des frameworks que nous utilisons. Nous avons donc regardé les tutoriaux de pytorch car nous voulions utiliser ce framework mais nous avons aussi regardé les githubs de modèles dont nous avions entendu parler, comme YOLOv5 . Il faut noter que notre compréhension du problème et du jargon utilisé dans le domaine s'est enrichi au fur et à mesure de nos recherches. Ainsi nos premiers choix peuvent sembler mauvais, mais lors de la prise de décision nous étions persuadé de faire le bon choix. Notre approche de départ se basait essentiellement sur un facteur: nous voulions des modèles pour lesquels il existe beaucoup de ressources en ligne. Cette méthodologie nous a amené à explorer une solution (YOLOv5) qui n'était pas adaptée à notre problème. Après beaucoup d'essaie infructueux nous avons changé de méthodologie, nous nous sommes laissé la liberté d'utiliser des outils plus haut niveau, comme Detectron2 et de ne pas se restraindre à uniquement PyTorch. Une fois que nous avons pris en main ce framework nous avons pu nous concentrer sur la performance du modèle, c'est aussi à ce stade que nous avons compris que le score sur le benchmark COCO2017 indiqué sur beaucoup de documentation de modèles était justement indiqué pour pouvoir comparer les modèles entre eux.

5.2 les échecs

YOLOv5

YOLO est l'acronyme de You Only Look Once, et c'est aussi le nom du premier modèle que nous avons essayé. Nous avons décidé de commencer avec ce modèle pour plusieurs raisons parmi lesquelles : une abondance de tutoriel sur le net et une solution qui nous semblait clé en main pour résoudre notre problème. Malgré ces signes positifs, il n'a pas été une solution adaptée. En effet YOLO, a été publié il y a plus de 2 ans et n'est plus forcément l'état de l'art actuel, de plus ce modèle est adapté à du traitement en temps réel ce qui ne fait pas partie de notre problème. Cependant le réel souci qui nous a fait abandonner cette solution est la structure spéciale du dataset qui ne correspondait pas à notre structure. Durant une phase de réflexion visant à résoudre le souci de structure, nous avons ainsi réalisé l'incapacité du modèle à gérer des images n'étant pas de 640x640 pixels. Nous aurions pu effectuer un réajustement de la taille des images mais ces derniers éléments nous ont fait réaliser que YOLO n'était pas la solution clé en main à laquelle nous nous attendions et après quelques

discussions nous avons décidé de passer à un modèle plus adapté à notre problème initial et nous nous sommes lancé sur faster RCNN, un modèle qui supporte des images de tailles arbitraires et qui est plus récent que YOLOv5.

SSD

Faster RCNN avec Keras

5.3 FASTER R-CNN

source1 : https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

source2 : https://stackoverflow.com/questions/48318448/understanding-basic-difference-between-a-cnn-and-rnn

+ check videos en cherchant "rcnn, fast rcnn, faster rcnn" sur google

Description

- Histoire (evolution de RCNN)
- Date du papier l'expliquant
- architecture
- Score benchmark coco

Adapter les exemples sur un problèmes légèrement différent : bullshiter sur assembler différent tutoriaux nécessite de comprendre, perte de temps sur manque de connaissance du jargon technique : mAP, etc les benchmarks coco, comprendre que coco (Common Object in COntext) est beaucoup de chose (dataset, benchmark baseline, dataset d'entrainement)

5.4 RETINA net

Description

Results

5.5 DEtection Transformer (DE-TR)

DETR est un modèle sorti en 2020 qui est extrêmement simple à implémenter et fournit des scores (mAP, IoU etc) sur le dataset COCO qui surpassent ceux des modèles existants de quelques points. C'est pourquoi, en plus de son architecture innovante, nous avons voulu l'essayer. L'architecture, visible en figure 5.1, se base sur un transformer, un modèle de deep learning parut dans le fameux papier de 2017 de Google, *Attention is all you need*. On peut observer que le modèle commence par génèrer des features à partir de l'image d'entrée en utilisant une backbone c'est à dire un modèle pré-entrainé visant justement à extraire ces features à l'aide d'un réseau convolutionnel. Ensuite, le modèle utilise la partie encodeur du transformer suivit du décodeur et finalement d'un `prediction feed-forward networks`

(FFN). C'est le réseau FFN qui génère les possibles boundings boxes et les classes associées.

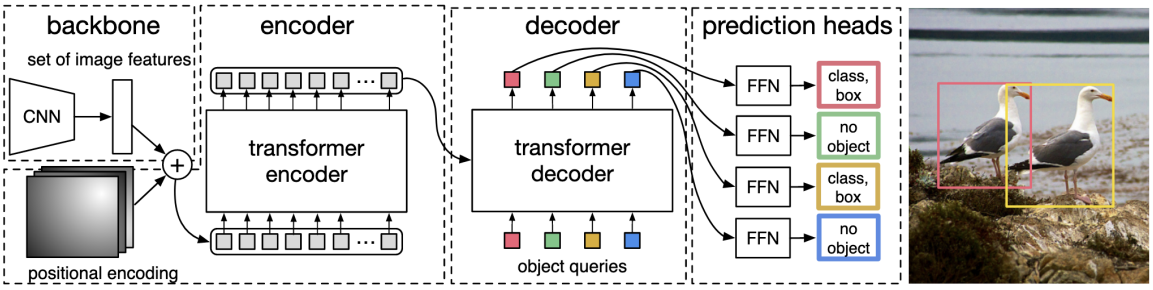

Figure 5.1: Architecture de DETR

Chapter 6

Evaluation

6.1 Section1

paragraph1

6.1.1 Features

Dans une tâche d'object detection, nous utilisons le score IoU qui signifie Intersection over Union. C'est un score qui compare les bounding boxes prédites par le modèle avec les bounding boxes réels (ground-truth). Une tâche d'object detection comprend deux sous-problèmes: la classification et la localisation. Ainsi nous avons plusieurs métriques pour analyser la performance d'un modèle. IoU se concentre sur l a localisation des bounding box prédites tandis que la métrique mAP (mean Average Precision) se concentre sur la classification. Ceci est un acronyme k-nearest neighbor

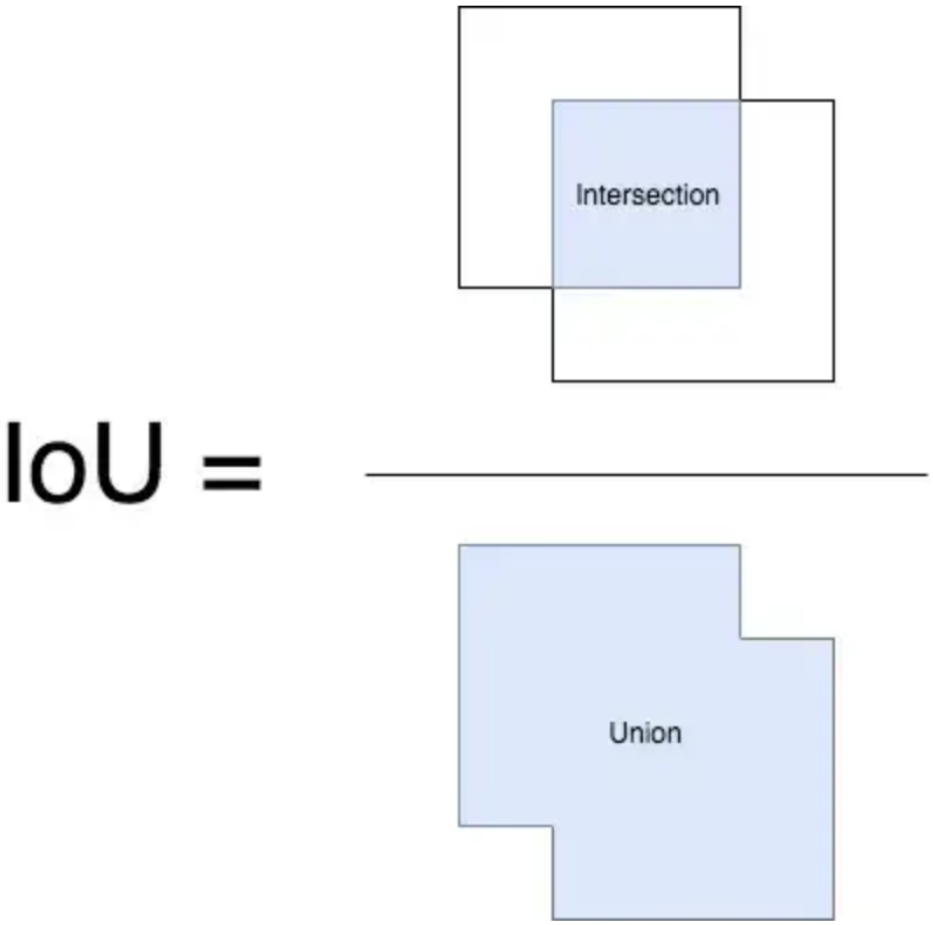

Figure 6.1: Intersection over Union

Chapter 7

Deployment

7.1 Section1

paragraph1

7.1.1 Features

Chapter 8

Conclusion

Ceci est un acronyme k-nearest neighbor