Ingénieur Machine Learning

Projet 7

Développez une preuve de concept

Pelletier Gaëtan

Table des matières

I. Présentation du projet

En partant d'un projet existant, nous tentons d'améliorer la solution développée et déployée. Ici, nous allons explorer un nouvelle fois notre jeu de données sur des races de chien, afin d'ajouter des fonctionnalités au produit final.

Dans le projet précédent, nous avions mis en place des CNN permettant de classifier les races de chiens. Ici, nous allons, d'une part tenter d'améliorer les performances de classification, mais aussi localiser le chien sur une photo. Pour ce faire nous allons utiliser un algorithme de détection d'objets (*illustration 1*).

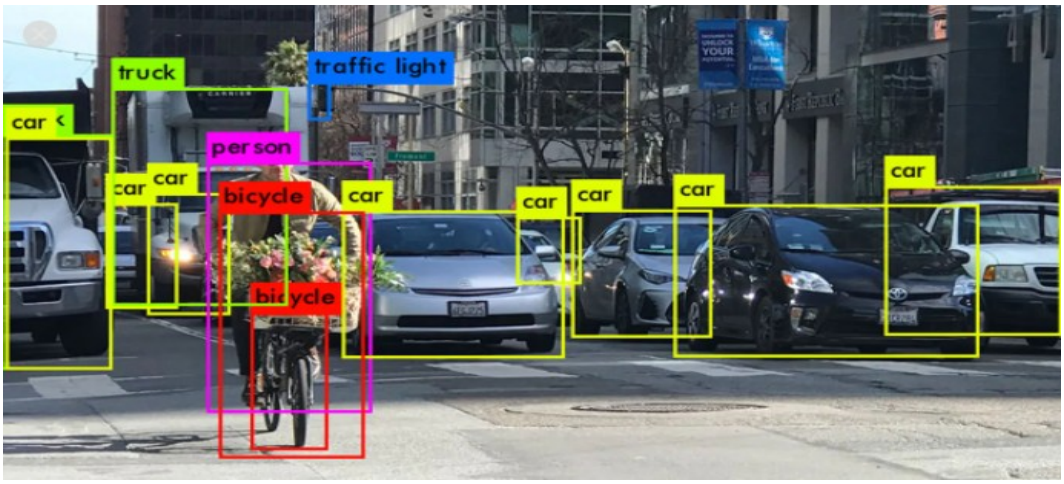

Illustration 1: Détection d'objet sur une image

II. État de l'art de la détection d'objets

La détection d'objet sur une image regroupe la classification de cet objet, ainsi que sa localisation. Cette dernière se visualise grâce à un rectangle d'ancrage (*anchor box*) encadrant la cible. Le jeu de données se compose souvent d'images et de fichiers textes regroupant la classe ainsi que les coordonnées du centre du rectangle d'ancrage, ainsi que sa longueur et sa hauteur.

Différentes architectures d'algorithme ont permis d'effectuer de la détection d'objet. Dans un premier temps, il y a eu l'utilisation de CNN de classification, auxquels se greffait une partie supplémentaire permettant de prédire les valeurs relatives à la localisation. Les algorithmes de type R-CNN, Fast R-CNN et Faster R-CNN ont permis d'obtenir des résultats de plus en plus impressionnants.

Puis, des réseaux de neurones entièrement convolutifs ont été développés. On parle alors de R-FCN. Ces réseaux de neurones ont surpassé les performances des meilleurs Faster R-CNN.

Enfin, une avancé majeure a été d'utiliser un R-FCN en appliquant une méthode dite *single shot*. Les algorithmes plus anciens faisaient une première passe afin de trouver des régions d'intérêt, puis les analysaient. Ici, l'algorithme effectue la détection en une seule passe. Les algorithmes utilisant cette méthodologie sont SSD ou YOLO.

Afin de mesurer les performances d'un algorithme de détection d'objet, nous pouvons utiliser les métriques classiques pour la classification et la métrique mAP (avec seuil IoU) pour la localisation.

III. Classification de races de chien

1. Jeu de données utilisé

Le jeu de données utilisé est celui du projet P6 du parcours IML : *Classez des images à l'aide d'algorithmes de Deep Learning*.

Pour chaque image de ce jeu de données, nous devons créer un fichier texte comprenant la race du chien (classe), ainsi que les informations du rectangle d'ancrage. Pour ce faire, nous utiliserons *Make Sense* (https://www.makesense.ai/). La création de tels fichiers étant très chronophage, nous nous limiterons à un jeu de données comprenant seulement 10 races de chien.

2. Modèles de référence

Nous allons utiliser deux modèles comme référence pour ce projet. Le premier est un CNN sans pré-entraînement. Le second est un CNN utilisant les poids d'un modèle Xception, grâce au *Transfer Learning*.

Une fois entraînés, nous pouvons mesurer les performances de ces modèles sur le jeu de test.

	CNN from scratch	**Xception**	**Xception (120 breeds)**
Loss	3,602	0,0115	0,3385
Accuracy	41,43 %	99,71 %	89,31 %

Maintenant que nous avons nos modèles de référence pour la partie classification, nous allons mettre en place un algorithme YOLOv4 afin de détecter des races de chiens dans des images.

IV. YOLOv4

1. Configurations de l'algorithme

Le projet YOLOv4 est récupérable à cette adresse : https://github.com/AlexeyAB/darknet.

Afin d'utiliser le *Transfer Learning*, nous téléchargeons des poids pré-entraînés via github : https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137

La configuration du projet permet de modifier le nombre de classes à détecter. Cependant, les paramètres optimaux impliquent un entraînement très long par rapport au temps requis pour ce projet. Nous faisons donc le choix de sous-dimensionner la valeur de certains de ces paramètres, afin d'accélérer la phase d'apprentissage. Il est quasiment certain que les performances de l'algorithme vont être impactées.

2. Métriques

Une fois l'entraînement terminé, nous pouvons mesurer les performances de l'algorithme sur le jeu de test (*illustration 2*).

```
 detections_count = 2206, unique_truth_count = 392
class_id = 0, name = Affenpinscher, ap = 90.61%          (TP = 13, FP = 0)
class_id = 1, name = Afghan_hound, ap = 94.40%           (TP = 50, FP = 14)
class_id = 2, name = African_hunting_dog, ap = 80.30%            (TP = 37, FP = 4)
class_id = 3, name = Airedale, ap = 93.93%       (TP = 36, FP = 7)
class_id = 4, name = American_Staffordshire_terrier, ap = 55.09%         (TP = 19, FP = 16)
class_id = 5, name = Chihuahua, ap = 54.53%      (TP = 18, FP = 21)
class_id = 6, name = Doberman, ap = 60.62%       (TP = 20, FP = 13)
class_id = 7, name = German_shepherd, ap = 68.28%        (TP = 23, FP = 17)
class_id = 8, name = Labrador_retriever, ap = 42.20%     (TP = 30, FP = 45)
class_id = 9, name = Yorkshire_terrier, ap = 78.17%      (TP = 32, FP = 21)

 for conf_thresh = 0.25, precision = 0.64, recall = 0.71, F1-score = 0.67
 for conf_thresh = 0.25, TP = 278, FP = 158, FN = 114, average IoU = 47.94 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.718125, or 71.81 %
```

Illustration 2: Performances de YOLOv4

On constate que YOLOv4 atteint une précision de 64 % sur le jeu de test (pour un rappel de 71 %). Pour la classification, ce modèle surclasse notre CNN sans pré-entraînement. Mais il se fait lui-même dépasser par le modèle Xception.

Pour la localisation d'objet, notre modèle atteint un score mAP@0.5 de 0,718.

3. Limites de cette méthode

Nous avons utilisé l'algorithme YOLOv4 afin de détecter des races de chien. Or, l'entraînement de YOLOv4 est assez chronophage avec 10 races. Cela sera encore plus compliqué avec les 120 races du jeu de données complet. De plus, l'utilisation de *darknet* n'est pas évidente. Enfin, l'inférence sur une image du jeu de test (*illustration 3*) n'indique pas l'indice de confiance de la prédiction.

Illustration 3: Inférence avec YOLOv4

Si nous regardons le Github de YOLOv4, nous pouvons trouver des projets intéressants permettant de contourner l'utilisation de *darknet*. L'un de ces projets utilise Pytorch et semble être plus rapide à entraîner : le projet YOLOv5.

V. YOLOv5

1. Configurations de l'algorithme

Le projet YOLOv5 est récupérable à cette adresse : https://github.com/ultralytics/yolov5.

Afin d'utiliser le *Transfer Learning*, nous téléchargeons des poids pré-entraînés grâce au fichier */yolov5/weights/download_weights.sh*

La configuration du projet permet de modifier le nombre de classes à détecter, mais aussi de paramétrer la data augmentation à utiliser (*illustration 4*). Nous nous inspirons du projet précédent afin de choisir des valeurs cohérentes par rapport à l'étude de photos de chiens (e.g. valeur peu extrême pour la rotation, car il y a peu de chance de rencontrer une image avec un chien à l'envers).

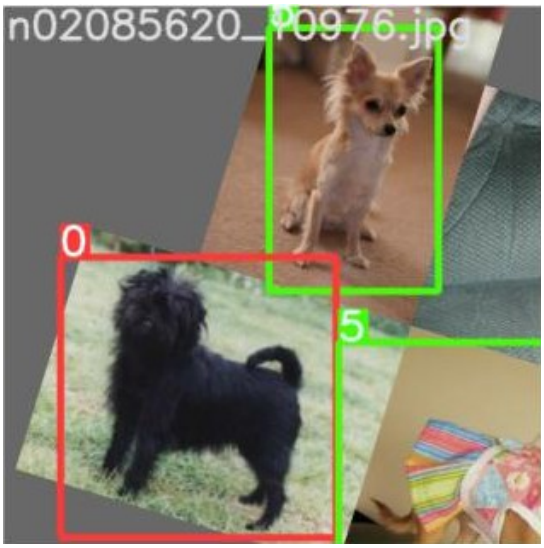

Illustration 4: Data augmentation avec YOLOv5

2. Métriques

Une fois l'algorithme entraîné, nous pouvons mesurer ses performances. Pour optimiser ses résultats, il est nécessaire de fixer un seuil de confiance. Pour cela, nous pouvons nous servir de la courbe F1-score produite automatiquement par YOLOv5 (*illustration 5*). Ainsi, nous pouvons mesurer les performances de l'algorithme pour la détection de races de chien (*illustration 6*).

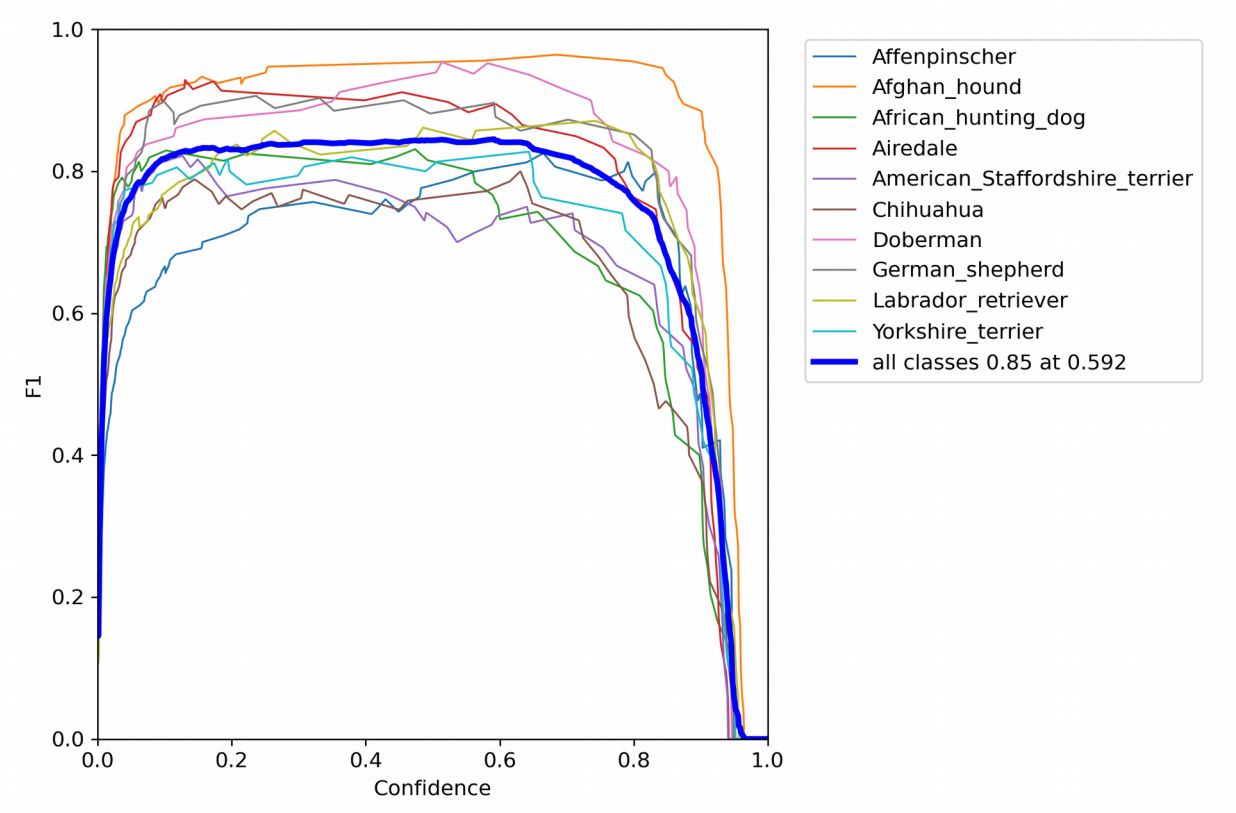

Illustration 5: Courbe F1-score de YOLOv5

Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95:
all	350	392	0.925	0.764	0.756	0.576
Affenpinscher	350	30	0.935	0.967	0.94	0.68
Afghan_hound	350	54	1	0.833	0.838	0.699
African_hunting_dog	350	49	1	0.673	0.678	0.519
Airedale	350	42	0.973	0.857	0.843	0.59
American_Staffordshire	350	37	0.893	0.676	0.678	0.524
Chihuahua	350	34	0.862	0.735	0.714	0.519
Doberman	350	31	0.96	0.774	0.769	0.593
German_shepherd	350	33	0.903	0.848	0.847	0.684
Labrador_retriever	350	45	0.84	0.467	0.457	0.335
Yorkshire_terrier	350	37	0.882	0.811	0.794	0.617

Illustration 6: Performances de YOLOv5

On constate que YOLOv5 atteint une précision de 92,5 % sur le jeu de test (pour un rappel de 76,4%). Pour la classification, ce modèle surclasse notre CNN sans pré-entraînement, ainsi que YOLOv4. Mais il se fait devancer par le modèle Xception.

Pour la localisation d'objet, notre modèle atteint un score mAP@0.5 de 0,756. Ce score dépasse celui obtenu avec l'algorithme YOLOv4. Rappelons qu'il a été plus difficile d'entraîner YOLOv4 à cause du temps requis. La réduction du temps d'entraînement a sûrement dû impacter les performances de ce dernier.

3. Déploiement du modèle

Après avoir effectué des inférences sur des images du jeu de test (*illustration 7*), nous pouvons tenter d'intégrer notre modèle au sein de *Gradio*. L'intérêt est de nous assurer que la solution mise en place puisse être facilement déployée pour une phase de démonstration.

Illustration 7: Inférence avec YOLOv5

En chargeant notre modèle grâce à la libraire Pytorch, nous obtenons une application capable de localiser des chiens sur une image, ainsi que de donner leur race (*illustration 8*).

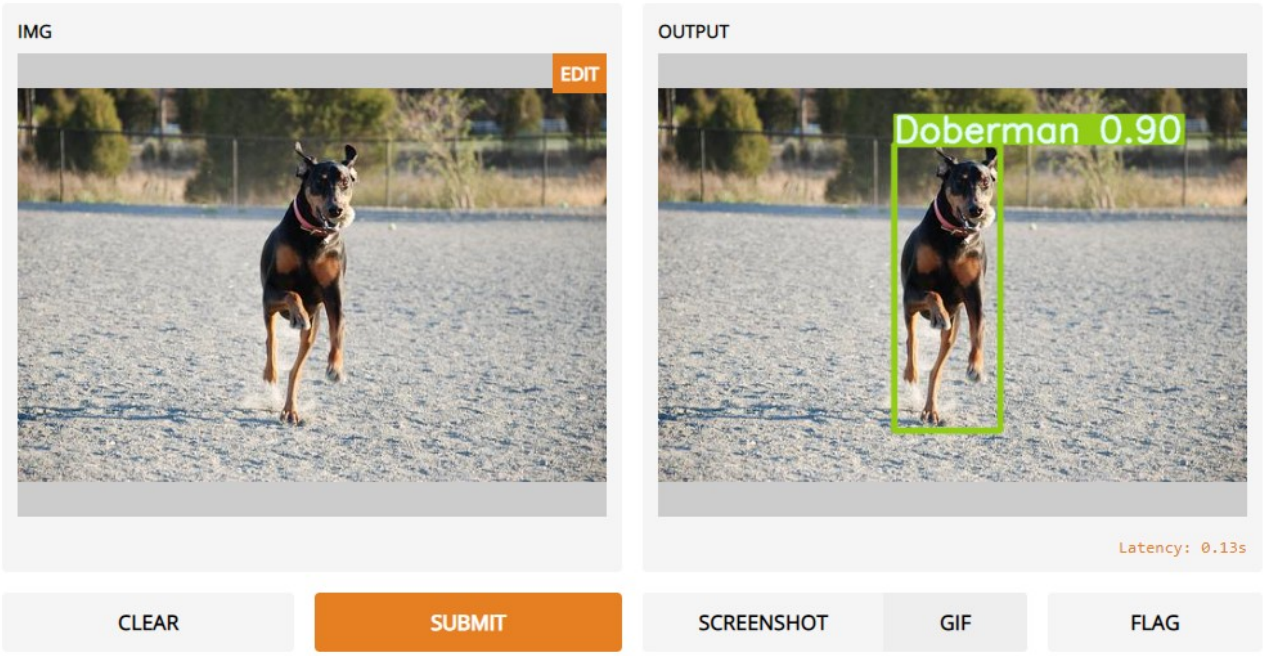

Illustration 8: Déploiement de YOLOv5 avec Gradio

Nous remarquons, pour certaines inférences, un soucis de police d'écriture au niveau du nom de la classe. Cela complique la lecture du nom de la race. De plus, l'indice de confiance de la prédiction est parfois hors cadre. Enfin, rappelons que ce modèle n'a été entraîné que sur 10 races de chien. Il n'est donc, actuellement, pas utilisable pour classifier des races de chiens, comparé à notre modèle déployé lors du projet précédent.

VI. Conclusion

L'utilisation de YOLOv4 nous a permis de mettre en place un détecteur de races de chiens présents sur des images. Face à des difficultés de temps d'entraînement, nous avons découvert un projet annexe : YOLOv5. Cet algorithme permet d'atteindre une précision largement supérieure à notre CNN entraîné de zéro pour la partie classification. Cependant, il ne permet pas de concurrencer le modèle basé sur Xception.

Nuançons toutefois les performances de ce dernier, car il a été pré-entraîné sur ImageNet. Or, notre jeu de données provient lui aussi d'ImageNet. Il est fort probable que ce CNN soit biaisé (e.g. il connaît déjà les photos du jeu de test). Les modèles YOLO sont pré-entraînés sur le jeu de donnée COCO. Leurs performances semblent alors plus fiables.

Si nous pouvions passer plus de temps sur ce projet, nous devrions labelliser le reste du jeu de données complet, afin d'indiquer les rectangles d'ancrage. Ensuite nous pourrions entraîner YOLOv5 et évaluer ses performances pour la détection des 120 races de chiens.

VII. Sources bibliographiques et autres

- Livre :

Deep Learning avec Keras et TensorFlow 2nd édition, *A. Géron* (2020)

- Papier de recherche :

YOLOv4 : Optimal Speed and Accuracy of Object Detection, *Bochkovskiy, Wang, Mark Liao* https://arxiv.org/pdf/2004.10934.pdf (2020)

- Blogs :

Deep Learning for Object Detection : A Comprehensive Review, *Joyce Xu* https://towardsdatascience.com/deep-learning-for-object-detection-a-comprehensive-review-73930816d8d9 (2017)

R-CNN, Fast R-CNN, Faster R-CNN, YOLO – Object Detection Algorithms, *R. Gandhi* https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e (2018)

YOLOv4 on Google Colab : Train your Custom Datasetwith ease, *Quang Nguyen* https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e (2020)

How to Train YOLOv5 On a Custom Dataset, *Jacob Solawetz, Joseph Nelson* https://blog.roboflow.com/how-to-train-yolov5-on-a-custom-dataset/ (2020)