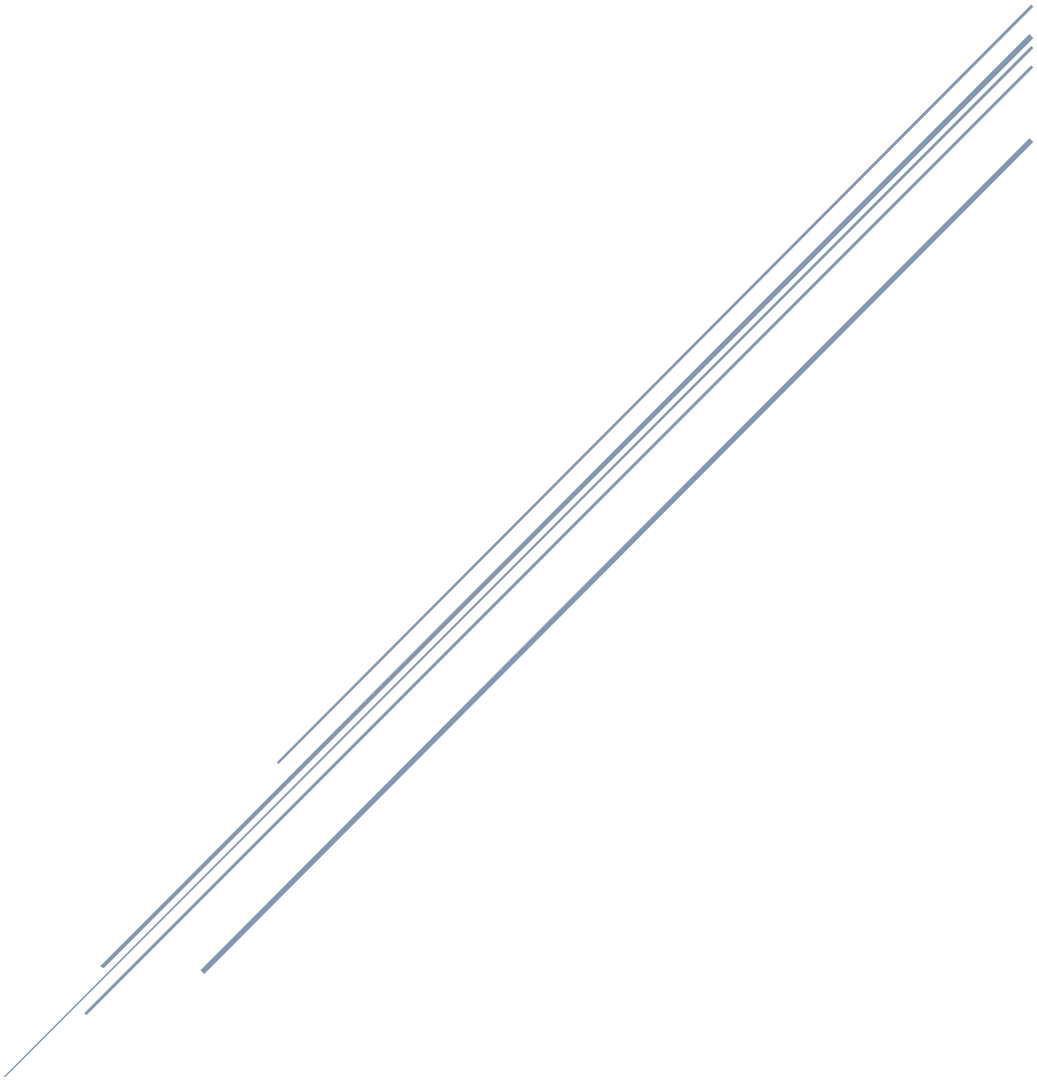
RAPPORT D'EXPERIENCE

Détection de braconniers sur vidéo de drones



BORDEAUX YNOV CAMPUS

Deep Learning

Pierre LEROY

P. COLINMAIRE / L. CONSTANTIN / N. LAUGA / M. TILHET

Table des Matières

Table des Figures

Rapport d'expérience

I) Contexte

1.1) Rappel du besoin

Un organisme de protection des animaux souhaite mettre en place un système de détection de braconniers. L'organisme souhaite identifier le danger, représenté par les braconniers, afin de le mettre hors d'état de nuire. Ce système de détection a pour vocation d'être embarqué sur des drones, et aura pour source d'information les images captées par la ou les caméras du drone.

1.2) Enjeux

Suite à l'analyse du cahier des charges, l'équipe projet comprend que l'enjeu majeur de ce projet est de fournir un système de détection à la fois fiable et performant, tout en réduisant les coûts de déploiement et assurer la scalabilité de la surveillance.

En effet, la tâche de détection de personnes sur des images de drones est une tâche pénible et répétitive, nécessitant une forte attention de la part de l'opérateur. Or, la fatigue entrainée par des périodes de travaux prolongées a un impact direct sur l'attention humaine, et donc, augmente le risque d'échouer dans cette tâche de la détection. L'utilisation d'une machine, combiné à la puissance de l'intelligence artificielle, vient palier à ce problème.

De plus, au cours des années, les braconniers ont mis en place des techniques toujours plus poussées pour se dissimuler. Cette faculté des braconniers à se fondre dans la nature rend la tâche d'autant plus difficile pour une personne n'ayant aucune qualification ou formation spécifique dans le domaine de la surveillance. Pour cette raison, l'organisme se doit de recruter des opérateurs experts dans ce domaine, ce qui complexifie la tâche du recrutement. L'utilisation d'une solution informatique vient également palier à ce problème.

Pour finir, les avancées technologiques de la robotique ces dernières années a entrainé une baisse significative des prix des drones, rendant ces appareils bon marchés et accessibles. Le choix de déployer la solution sur des drones rend cette dernière à la fois facile et engageant des coûts acceptables.

Bien évidemment, outre ces enjeux économiques, l'équipe projet a conscience de la parfaite adaptation d'un drone dans un environnement tel que la savane. On peut notamment penser à la capacité du drone d'accéder à des endroits difficiles d'accès, ou encore sa capacité à surveiller un périmètre global via les airs. Ces deux raisons semblent parfaitement justifier l'utilisation d'un drone, en comparaison à d'autres robots de surveillance tels que des caméras fixes ou des appareils roulants.

1.3) Qui sommes-nous ?

CPF est une start-up informatique spécialisée dans la création de solutions informatiques basées sur des intelligences artificielles.

1.4) Éléments Facilitateurs

- Visibilité du projet : L'organisme fournit avec son cahier des charges deux schémas présentant le fonctionnement attendu de la solution.
- Fourniture d'un jeu de données labelisées : Le projet nous est fourni avec les jeux de données « Okutama » et « Aiskeye ». Ces jeux de données contiennent des vidéos et des images, captées par des drones, et labelisées.
- Détection simplifiée : L'organisme accepte pour le POC de recevoir une détection binaire du danger. En ce sens, l'équipe projet doit démontrer sa capacité à détecter la présence ou l'absence d'un danger. Le nombre de braconniers et leur localisation précise sur l'image fera l'objet d'une phase projet ultérieure.
- Analyses de fichiers : L'organisme accepte pour le POC d'effectuer des détections sur des fichiers vidéo enregistrés, et non un flux vidéo continue (=streaming)
- Proximité client : Un employé de l'organisme s'est rendu disponible pour porter la maitrise d'ouvrage sur ce projet, et se présenter comme un interlocuteur privilégié avec l'équipe. (=Single Point Of Contact)

1.5) Challenges

- Rareté des données : L'équipe projet ne dispose d'aucune image de drone, captées en conditions réelles, et labelisées.
- Temps de projet réduit : Le projet doit être réalisé sur une période de trois jours. Or, selon le triangle des Coûts / Délais / Qualité, l'équipe dispose de Coûts et de Délais fixes. L'équipe projet doit donc démontrer sa capacité maximiser la qualité de son projet, compte tenu des préoccupations fixes précédemment évoquées.
- Exigence de qualité : Pour le POC, l'organisme souhaite disposer d'un taux de fausses alertes inférieur à 5%, et souhaite également louper le moins possible de braconniers.

II) Gestion de projet

2.1) Découpage du projet

Pour optimiser et paralléliser les temps de travaux des membres de l'équipe tout en limitant les situations d'interblocages, le projet a été découpé en deux macro-parties :

- Le modèle : Il s'agit du cœur de fonctionnement de la solution. C'est l'intelligence artificielle qui aura pour but de détecter la présence de braconniers. Les exemples de tâches sur cette partie sont : L'implémentation du modèle et la détection d'objets, le déploiement d'un système de monitoring, la formulation d'une fonction de coût, etc etc…
- Le pipeline projet : Il s'agit de toute l'enveloppe du modèle, permettant de préparer les données d'entrées pour les injecter dans le modèle, mais également de capter les données en sortie du modèle pour les rendre exploitable. Les exemples de tâches sur cette partie sont : Les conversions d'images en vidéo et vice-versa, la détection de danger, le dessin des bounding boxes, l'évaluation des metrics, etc etc…

La volonté de l'équipe est de créer un pipeline projet modulable, au sein duquel il serait possible de placer un modèle entrainé. Une telle dynamique permettra à l'équipe de déployer en parallèle un laboratoire indépendant, où le modèle pourra être amélioré, pendant que le pipeline projet continue de s'affiner.

2.2) Organisation et Planification

Ce projet a été réalisé sur la période du printemps 2020, qui, nous tenons à le préciser, a été marquée par un confinement généralisé dû à la pandémie de Covid-19. Aussi, pour cette raison, l'équipe projet a dû mettre en place une organisation lui étant inédite, basée sur les contraintes d'un projet réalisé intégralement à distance.

Pour organiser à bien le projet, l'équipe s'est inspirée du process framework « Scrum ». Cependant, en vue de la courte durée du projet, et compte-tenue des attentes, l'équipe a accepté de réduire le temps dédié à l'entretien et l'actualisation des outils formels de gestion de projet, afin de dégager davantage de ressources pour les réalisations techniques, et l'amélioration de la qualité du livrable.

Initialement, un tableau virtuel de post-it a été déployé pour découper les tâches et assigner ces dernières aux différents membres. Cependant, après quelques heures de projet, l'équipe s'est aperçue que la réalisation de chacune des tâches était de trop courte durée pour bénéficier des avantages de cet outil. De plus, chacune des tâches permettait d'obtenir une sortie fonctionnelle, mais la sortie technique était peu visible avant de l'avoir disponible. En ce sens, de nombreux changements prenaient lieu à une fréquence élevée, ce qui complexifiait davantage l'entretien du tableau de post-it, et réduisait la pertinence de ce dernier.

Aussi, c'est pourquoi l'équipe a rapidement basculé sur une organisation de projet basée intégralement sur une mise en communication permanente de tous les membres, et intégration fréquente des travaux de chacun via la plate-forme GitHub. De plus, assurer un environnement de travail commun, un conteneur docker a été créé et déployé sur le poste de chacun des membres.

Afin d'assurer la coordination des travaux, et la tenue des délais, un lead projet a été désigné pour superviser l'avancée des travaux, ainsi que prioriser et distribuer les tâches à effectuer.

2.3) Répartition des responsabilités

En vue des domaines de compétences des membres, ainsi que de leur volonté d'approfondir leurs connaissances dans un secteur particulier, l'équipe s'est réunie pour mettre au point, ensemble, une répartition de responsabilités sur les quatre pôles principaux du projet :

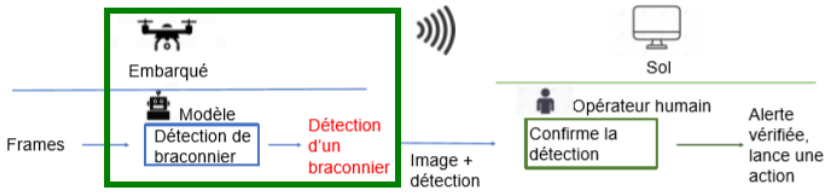
- Lead Projet : Son rôle principal est de superviser le projet dans son ensemble. Il doit mener les membres tout au long du projet en communicant des informations basiques sur le projet, tout en étant à l'écoute des besoins de chacun des membres.
- Analyste fonctionnel : Son rôle principal est d'étudier les différentes manières de mettre en œuvre le besoin du client au sein de la solution. C'est le membre de l'équipe le plus au courant des exigences clients.
- Expert Pipeline : Son rôle principal est d'implémenter les fonctionnalités de la solution, hors détection des braconniers. Il est en charge d'optimiser l'utilisation de la solution, et de simplifier l'exploitation du modèle, notamment à travers la mise en place de diverses transformations des informations d'une couche à l'autre.
- Ingénieur R&D : Son rôle principal est d'implémenter et optimiser les performances du modèle de l'intelligence artificielle. Il est en charge d'assurer la détection des braconniers sur des images.

Ainsi, pour ce projet, ces rôles ont été assignés aux personnes suivantes :

- Lead Projet : Paul COLINMAIRE
- Analyste fonctionnel : Maxime TILHET
- Expert Pipeline : Laetitia CONSTANTIN
- Ingénieur R&D : Nathan LAUGA

III) Analyses

Compte tenu du process cible, décrit par le client dans le schéma ci-dessous, l'équipe projet a concentré ses efforts de travail sur la partie du process encadrée en vert.



3.1) Analyse Fonctionnelle

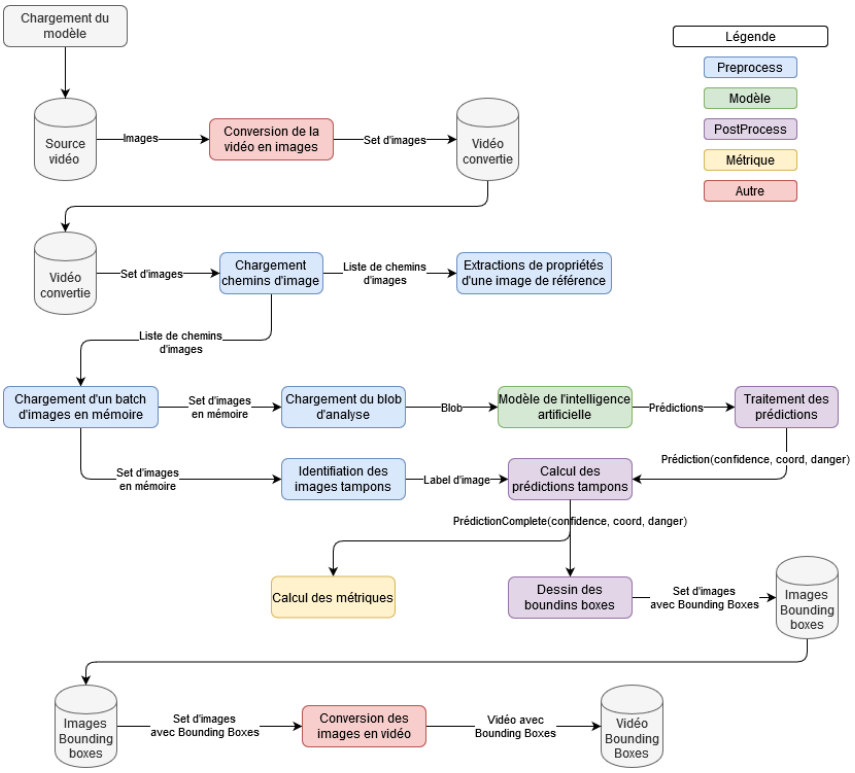


Figure 1 : Schéma du pipeline projet

Tôt dans le projet, l'équipe s'est réunie afin de définir ensemble ce qui serait appelé le « pipeline projet ». Ce pipeline projet consiste en l'assemblement des différentes briques fonctionnelles, périphériques au système de détections d'objets, nécessaires au bon fonctionnement du projet. Concrètement, ces différentes briques fonctionnelles se matérialiseraient sous la forme de code informatique python, exécutant des tâches simples telles que :

- Convertir une vidéo en image, et vice versa
- Charger des images dans le modèle
- Dessiner les prédictions
- Calculer des métriques
- Etc etc…

3.2) Analyse Technique

3.2.1. Choix du modèle YoloV3

L'objectif principal de ce projet étant d'implémenter une intelligence artificielle réalisant de la détection d'objet, la première des étapes de l'analyse technique fût d'effectuer un ensemble de recherche sur les architectures de « Convolutional Neural Network », afin d'étudier quelle architecture serait la plus à même de répondre à notre besoin, tout en respectant les contraintes associées.

Ainsi, le papier de recherche « Détection d'objets en milieu naturel : application à l'arboriculture » rédigé par Alexandre DORE et al., disponible sur archives-ouvertes.fr a retenu notre attention pour des raisons évidentes.

Ce papier compare notamment les performances des modèles « Faster-CNN » & « Yolo », en termes de rapidité d'exécution et de qualité des prédictions.

> *« L'une des principales différences entre YOLO et Faster R-CNN est le temps de calcul, YOLO permet d'avoir une détection de 37 images par seconde pour une image de445x445x3 alors que Faster R-CNN permet d'avoir seulement 5 images par seconde (sur les images prises dans des vergers). »*
> *Alexandre DORE et al., « Détection d'objets en milieu naturel: application à l'arboriculture »*

TABLE 1 – Comparaison entre méthodes de détection des pommes.

méthode de détection	taux de fausse détection	taux de non détection
Faster R-CNN	0.120	0.203
Ondelettes de Haar et réseaux de neurones (J.P.Wachs et al.)	0.172	0.488
YOLO	$\simeq 0.3$	$\simeq 0.6$

Alexandre DORE et al., « Détection d'objets en milieu naturel : application à l'arboriculture »

La lecture de ce papier nous a donc incité à se renseigner davantage sur l'architecture YoloV3. Notamment sur son site officiel : https://pjreddie.com/darknet/yolo/, où les Benchmarks réalisés nous apparaissaient comme satisfaisant.
Cependant, nous sommes restés conscients, lors de l'analyse de ce benchmark, que nous étions sur le site dédié à l'architecture, et qu'il était donc normal que les performances de YoloV3 soient mises en avant. Ainsi, l'analyse de ce benchmark a essentiellement permis de nous confirmer que son utilisation dans notre contexte était une solution envisageable.

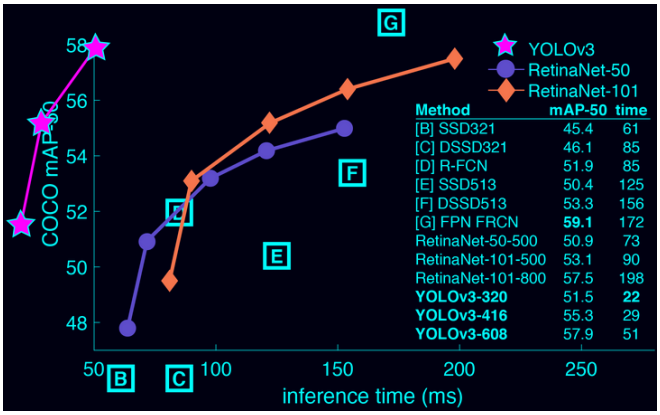


Figure 2 : Becnhmark des performances de YoloV3, en comparaison à d'autres CNN (https://pjreddie.com/darknet/yolo/)

Une dernière étape d'analyse a été réalisée avant de lancer l'équipe dans l'utilisation de cette architecture, il s'agissait de l'étude de sa complexité. En effet, bien que les performances du modèle soient séduisantes, nous voulions éviter de nous embourber dans une implémentation technique complexe, qui nous limiterait le champ de nos possibilités.

Aussi, nos recherches plus avancées sur l'architecture YoloV3 nous ont permis de découvrir l'image ci-dessous, issu d'un article du site medium.com, et présentant les différentes couches de neurones, et leurs imbrications

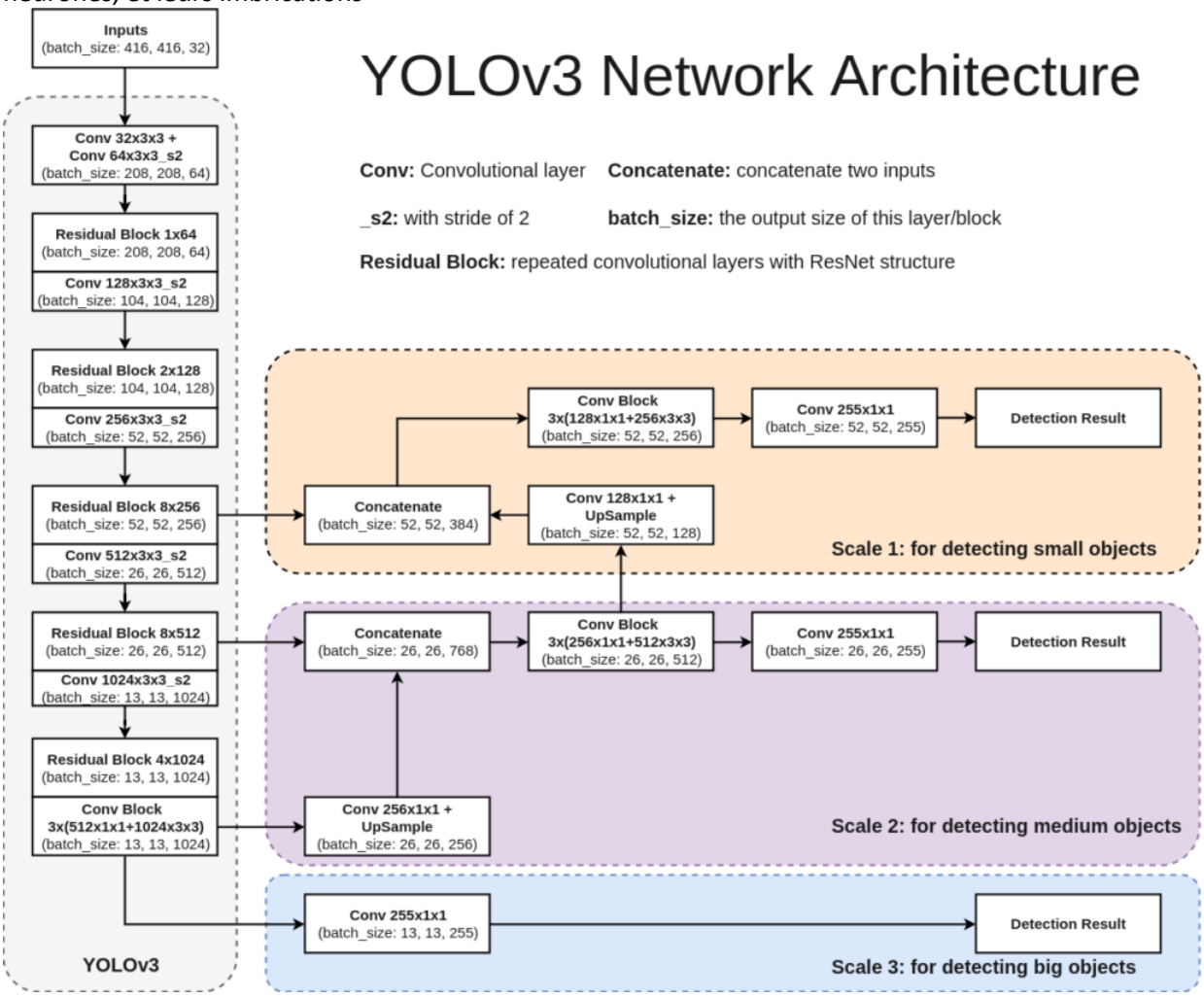


Figure 3 : Schéma de l'architecture YoloV3, disponible sur l'article « All about YOLOs — Part4 — YOLOv3, an Incremental Improvement », medium.com

Notre étude de cette architecture nous a permis de valider notre capacité à l'exploiter. Voici comme nous en sommes arrivés à choisir ce modèle pour ce projet.

3.2.2. Déploiement de pipeline via OpenCV

Comme précisé précédemment, nous avons choisi de séparer l'organisation du projet en deux macro-parties, à savoir

1) Création du pipeline projet
2) Optimisation du modèle.

Ainsi, à l'aide d'un article posté sur le blog d'Arun PONNUSAMY, nous avons pu implémenté un pipeline projet complet, en intégrant directement un modèle YoloV3 déjà entrainé.

Cette implémentation nous a été fortement utile afin de mieux comprendre les outputs du modèle Yolo, mais également, comment les traiter, afin de réaliser toutes les briques de post-process.

De plus, l'implémentation par OpenCV nous a permis de valider par la pratique le fait que le modèle YoloV3 était une architecture pouvant répondre à la fois à notre besoin et aux exigences techniques attendues.

Pour finir, une fois le pipeline implémenté, nous avons pu apprécier la facilité à être en mesure d'interchanger les architectures de Yolo. Ainsi, ce pipeline projet sous Open CV nous a également permis de tester les performances des architectures tiny-yolo et yolo-drone, afin de les comparer aux performances de YoloV3, et ainsi, choisir l'architecture la plus pertinente à implémenter en Keras, en vue de son optimisation.

L'implémentation du pipeline projet OpenCV est disponible au lien ci-dessous :
https://github.com/Nathanlauga/deep-learning-poacher-detection/blob/master/notebooks/poacher_detection.ipynb

3.2.3. Récupération YoloV3 via Keras

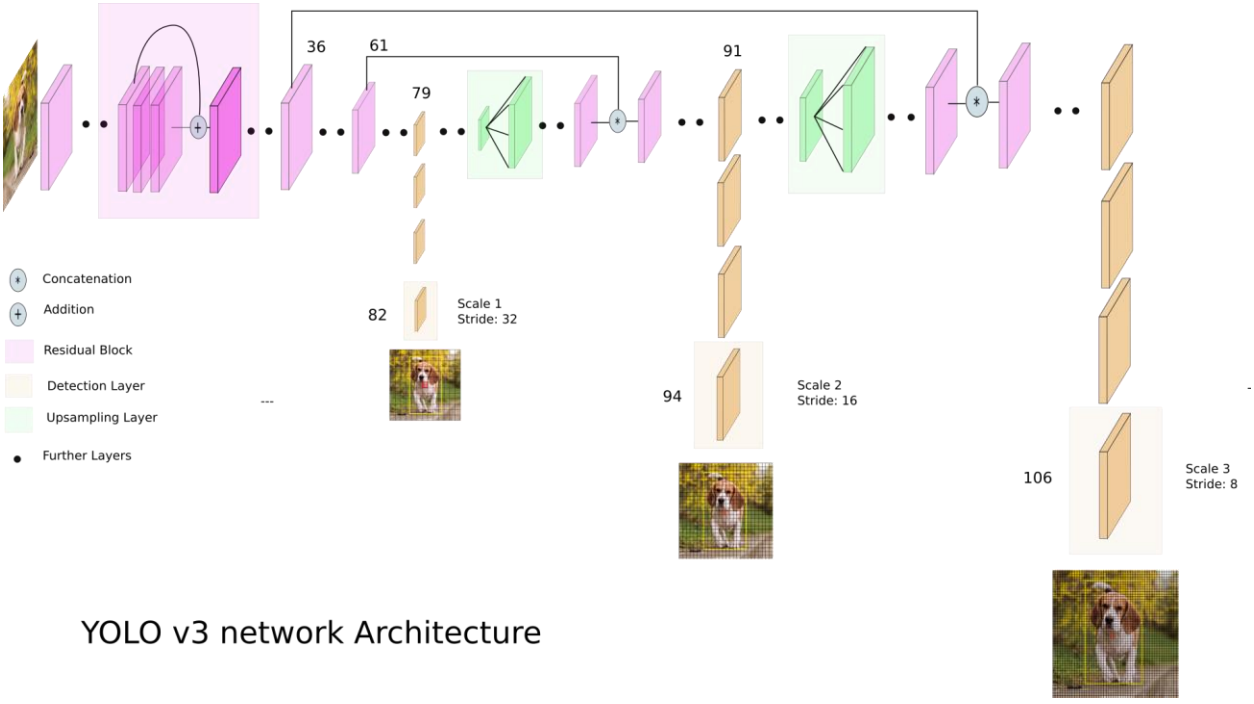


Figure 4 . Source : https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b

Afin de plus facilement modeler le modèle de YoloV3, nous avons choisi d'utiliser le framework de Keras sous la version 2.2.0 de Tensorflow. Pour parvenir à convertir le modèle et ses poids dans leur forme Darknet (un autre framework) nous avons effectués une première tentative avec le code suivant : https://github.com/qqwweee/keras-yolo3.

```
python convert.py yolov3.cfg yolov3.weights model_data/yolo.h5
```

La sortie nous donne un fichier au format h5 permettant de charger un model Keras grâce à la commande suivante :

```
model = tf.keras.models.load_model(model_path)
```

Cette conversion n'a pas fonctionné : les résultats donnaient des chiffres faux et par conséquent nous ne pouvions pas partir dessus.

Pour la suite nous avons choisi un nouveau GitHub pour récupérer le modèle YoloV3 en Keras : https://github.com/YunYang1994/TensorFlow2.0-Examples/tree/master/4-Object_Detection/YOLOV3

La bonne nouvelle a été le fait qu'il s'agissait d'une implémentation en Tensorflow v2 et par conséquent la facilité de récupération du modèle YoloV3. Pour rappel le modèle a l'architecture de base suivante :

Nous y voyons 3 sorties, la première correspondant à une grille de 13x13 cellule permet une analyse des objets prenant de la place (en « gros plan »), alors que la dernière sortie avec une grille de 52x52 est spécialisée dans les petits objets.

De plus, en somme, chaque sortie retourne nous retourne pour chaque cellule les informations suivantes :

X milieu	Y milieu	Largeur	Hauteur	Probabilité d'un objet	P(C0)	P(C1)	…	P(C79)

- Ce tableau est donc de taille 85
- les quatre premiers attributs indiquent la position de l'objet
- le cinquième donne la probabilité de la présence ou non d'un objet centré sur la cellule
  - Si la probabilité est faible : il ne faut surtout pas prendre en compte les autres chiffres.
- Le reste correspond aux probabilités des classes de 0 à 79 (0 étant la classe « personne »)

Pour revenir au code récupéré, le créateur du GitHub a fait le choix de rajouter des graphes de post-process après les différentes sorties, un exemple ci-dessous :

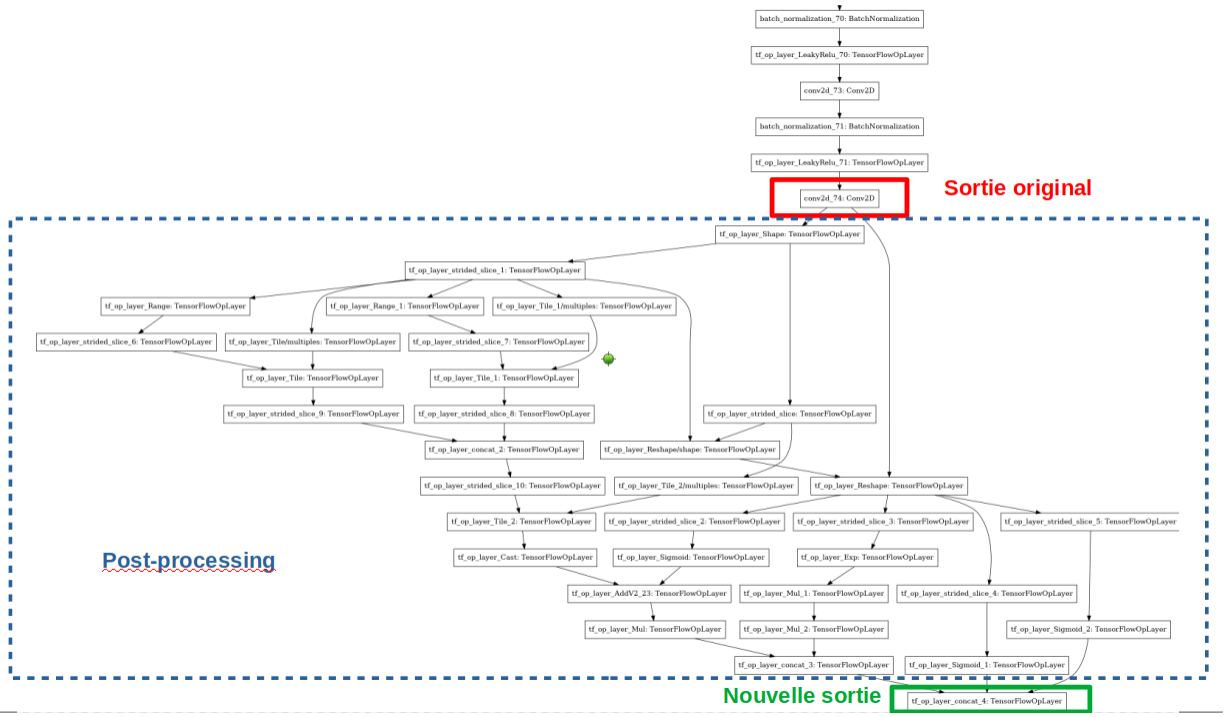


Figure 5 : Couches finales du modèle YoloV3 implémenté, ajoutant, en transfer learning, des couches de post-process d'images.

Parmi les opérations de post-processing nous pouvons retrouver : le « reshape » des sorties, la mise à jour des valeurs pour qu'elles soient en pixel et enfin l'utilisation de la fonction sigmoid pour les probabilités.

3.2.4. Transfer Learning via Keras

Comme nous avons pu le voir dans la partie précédente le modèle de départ correspond au YoloV3 légèrement modifié en sortie en Keras. Pour mieux travailler la problématique de la détection de braconniers nous avons testé les entraînements suivants :

- Modification des sorties de 85 dimensions à 5 puisque le nombre de classe à détecter est un (correspondant à « personne »)

Après différentes tentatives dans ce sens, toutes ont aboutis à la conclusion suivante : réduire la dernière dimension du modèle de YoloV3 ne fonctionne pas.

Même en réinitialisant les poids le modèle donnait par moment des sorties « -infini » et impossible pour nous de savoir pourquoi.

- Réduction du nombre de dimensions en sortie 6 ou 7 à la place de 85.

La conclusion a été la même que la précédente. Une piste nous a conduit sur les graphes de post-processing, mais leur pertinence dans le réseau nous a obligé a laissé tombé cette possibilité.

- Conservation de la même architecture en adaptant nos données d'entraînements.

Nous avons fait ce choix pour les raisons suivantes :

1. Les échecs des tentatives précédentes nous montrent bien que l'architecture du réseau est sensible en sortie ⇒ il est donc nécessaire d'éviter de la toucher
2. Les poids à disposition maîtrisaient déjà la reconnaissance de personnes avec la classe 0 et donc nous avions une bonne base de départ avant même de débuter l'entraînement.

De plus, afin de permettre un entraînement plus rapide, nous avons rendu uniquement les poids de la couche de convolution 73 et 74 (la dernière sortie 52x52 et la couche précédente) puisque la grande majorité de nos labels étaient répertoriés pour cette couche (pour information un objet est labellisé sur la couche lui correspondant le mieux ou s'il est entre deux couches, il peut être sur les deux, mais en aucun cas il ne sera labellisé sur les trois sorties).

En somme, notre modèle :

1. Reprends la même architecture que YoloV3.
2. Conserve les graphes de post-processing ajouté par le code de YunYang1994
3. Ne garde d'entrainable que les deux dernières couches de convolution au niveau de la sortie des petits objets (52x52).

Le code pour l'entrainement du modèle est accessible ici :
https://github.com/Nathanlauga/deep-learning-poacher-detection/blob/master/notebooks/Poacher%20detection%20-%20Train%20Yolov3.ipynb

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

Figure 6 : Formule mathématique de la loss function implémentée

3.2.5. Déploiement de pipeline via Keras

Le pipeline de traitement d'une vidéo via Keras est accessible sur le notebook suivant : https://github.com/Nathanlauga/deep-learning-poacher-detection/blob/master/notebooks/VIDEO%20PIPELINE%20-%20detect%20poacher%20on%20an%20inputed%20video%20.ipynb

3.2.6. Environnement de travail

Nous avons mis en place un GitHub pour le projet accessible à l'url : https://github.com/Nathanlauga/deep-learning-poacher-detection

De plus afin de faciliter l'accès à un environnement de travail mutualisé nous avons utilisé Docker et Docker-compose (GPU ou CPU selon l'ordinateur).

3.3) Définition des métriques

3.3.1. Calcul de la fonction de coût

La fonction de coût a été défini à l'aide cet article https://blog.emmanuelcaradec.com/humble-yolo-implementation-in-keras/.

Nous sommes donc parties sur la même fonction que les développeurs de Yolo ont utilisées (en la modifiant légèrement). La formule est la suivante :

Pour mieux comprendre nous allons l'étudier point par point :

- **Le coût de X et Y :**
  Le coût est calculé si et seulement si un objet est bel et bien présent dans la cellule. Contrairement à Yolo nous avons fait le choix de convertir X et Y avec une fonction sigmoid puis de calculter leur coût avec une binary crossentropy :

```
xy_loss = ( object_mask * K.binary_crossentropy( tf.sigmoid(y_true_xy),
tf.sigmoid(y_pred_xy), from_logits=True) )
xy_loss = K.sum(xy_loss) / m
```

- **Le coût de W et H (largeur et hauteur) :**
  Le coût est calculé si et seulement si un objet est bel et bien présent dans la cellule. Ensuite nous calculons la différence des racines carrées afin de réduire les erreurs quand les boîtes sont trop grandes.

```
wh_loss = ( object_mask * K.square(K.sqrt(y_true[..., 2:4]) - K.sqrt(y_pred[..., 2:4])) )
wh_loss = K.sum(wh_loss) / m
```

- **Le coût des classes :**
  Le coût est calculé si et seulement si un objet est bel et bien présent dans la cellule. Ensuite nous faisons tout simplement la racine de la différence des carrées.

```
class_loss = ( object_mask * K.square(y_true[..., 5:] - y_pred[..., 5:]) )
class_loss = K.sum(class_loss) / m
```

- **Le coût de la confience (IOU : intersection over union)**
  Le coût est calculé si et seulement si un objet est bel et bien présent dans la cellule. L'erreur ici calculera si la boîte est bien positionnée par rapport à l'objet réel.

```
confidence_loss = ( K.square(y_true_conf*iou - y_pred_conf)*y_true_conf )
confidence_loss = K.sum(confidence_loss) / m
```

- **Le coût de l'objet :**
  Il s'agit là du coût qui restera en cas d'absence d'objet : sans cette partie là, le modèle commencera à apprendre qu'il y a des objets partout puisque le coût sera toujours à 0 sans objet : elle est donc essentielle.

```
obj_loss = ( K.binary_crossentropy(y_true_conf, y_pred_conf, from_logits=True) )
class_loss = K.sum(class_loss) / m
```

3.3.2. Calcul de l'accuracy

- **Définition métier** : L'accuracy permet de comprendre la tendance générale du modèle puisqu'elle prend en compte les bonnes prédictions par rapport à toutes les images fournies.
- **Définition technique** : L'accuracy permet d'obtenir la proportion des prédictions correctes parmi l'ensemble des prédictions possibles.
- **Formule** : `Precision = (true_positive + true_negative) / nb_images_analysées`
- **Variables** :
  - true_positive signifie que la prédiction est correcte et qu'elle a détecté un danger
  - true_negative signifie que la prédiction est correcte et qu'elle n'a pas détecté un danger

3.3.3. Calcul de la précision

- **Définition métier** : La précision est utilisée pour savoir le pourcentage de notification "utile" à un opérateur pour les cas où il y a un braconnier sur l'image.
  Par exemple, une précision de 90% signifie que pour 10% des notifications envoyés aux opérateurs seront sans braconnier sur l'image.
- **Définition technique** : La precision est une mesure sur la qualité des prédictions. Elle permet d'obtenir la proportion des dangers correctement prédits parmi l'ensemble des dangers détectés.
- **Formule** : `Precision = true_positive / (true_positive + false_positive)`
- **Variables** :
  - true_positive signifie que la prédiction est correcte et qu'elle a détecté un danger
  - false_positive signifie que la prédiction est incorrecte et qu'elle a détecté un danger

3.3.4. Calcul du recall

- **Définition métier** : Le recall (*ou rappel*) correspond à la mesure la plus importante : en effet, plus le rappel est faible, plus de braconniers ne seront pas identifiés et par conséquent des animaux risquent d'être tués.
  Par exemple, un recall de 90% signifie que pour 10% des cas où il y a réellement un braconnier il ne sera pas identifié.
- **Définition technique** : Le recall est une mesure sur la quantité des prédictions. Elle permet d'obtenir la proportion des dangers correctement prédits parmi l'ensemble des dangers réels (les dangers correctement prédits et les dangers manquants).
- **Formule** : `Recall = true_positive / (true_positive + false_negative)`
- **Variables** :
  - true_positive signifie que la prédiction est correcte et qu'elle a détecté un danger
  - false_negative signifie que la prédiction est incorrecte et qu'elle n'a pas détecté un danger

IV) Résultat des expériences

4.1) Entraînement du modèle

Nous avons réalisé 2 entraînements qui ont abouti, le deuxième repartant du premier. De plus, nous avons implémenté l'outil Tensorboard, nous ayant ainsi permis de monitorer les entraînements pour lesquels nous allons voir les résultats ci-dessous :

Utilisation de l'optimizer Adam afin de ne pas sur-apprendre sur les données :

- Taux d'apprentissage : 0,001
- Beta 1 = 0,9
- Beta 2 = 0,999
- Epsilon = 1e-8

Paramètres :

- 6666 images pour l'entraînement dont 5 % de validation.
- Batch size de 4 (un batch size au-dessus pouvait échouer à cause de problème de mémoire)
- 5 epochs

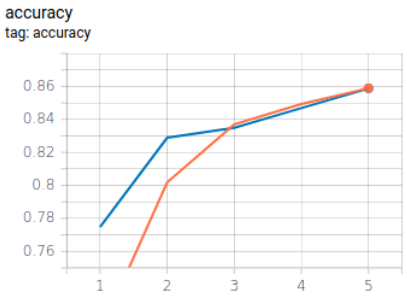


Figure 7 : Progression de l'accuracy au cours de l'entrainement N°1

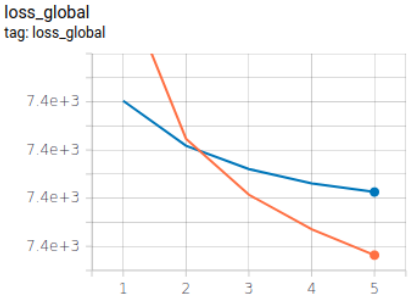


Figure 8 : Progression de la loss ou cours de l'entrainement N°1

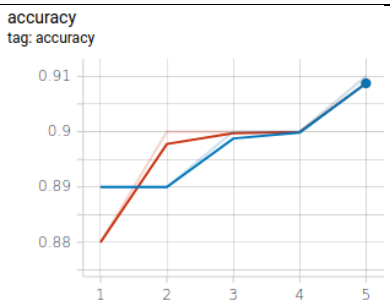


Figure 9 : Progression de l'accuracy au cours de l'entrainement N°2

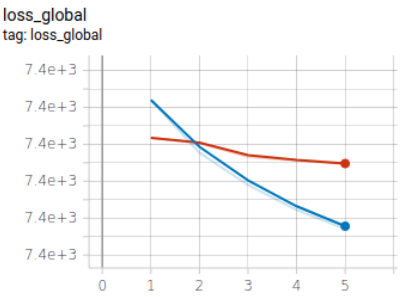


Figure 10 : Progression de la loss ou cours de l'entrainement N°2

Mêmes paramètres pour le second entraînement (les courbes reprennent à peu près où elles se sont arrêtées au-dessus).

4.2) Évaluation des modèles

Pour savoir la performance de notre modèle après entraînement nous l'avons comparé avec le modèle YoloV3 original. La comparaison s'est faite sur 1000 images tirées de manière aléatoires avec environ 600 images contenant des humains dedans et 400 sans humain. Les mesures sont affichées pour seulement la sortie 52x52 puisque la majorité des échantillons sont des avec des petits objets (seulement 18 images ont une sortie dans la sortie moyenne 26x26).

Modèle	Implémentation	Input size	Temps inférence	Accuracy	Precision	Recall
YoloV3	Keras	416x416	~0,37sec	77 %	94 %	68 %
Poacher 5 epochs	Keras	416x416	~0,37sec	91 %	94%	90%
Poacher 10 epochs	Keras	416x416	~0,37sec	92 %	95 %	90 %

Figure 11 : Tableau d'évaluation des performances des modèles

Pour calculer le temps d'inférence moyen, sur les implémentations Keras, nous avons exécuté le modèle avec Tensorflow CPU et calculer le temps pour chaque étape de l'analyse d'une image.
Code disponible ci-dessous :
https://github.com/Nathanlauga/deep-learning-poacher-detection/blob/master/notebooks/VIDEO%20PIPELINE%20-%20detect%20poacher%20on%20an%20inputed%20video%20.ipynb)

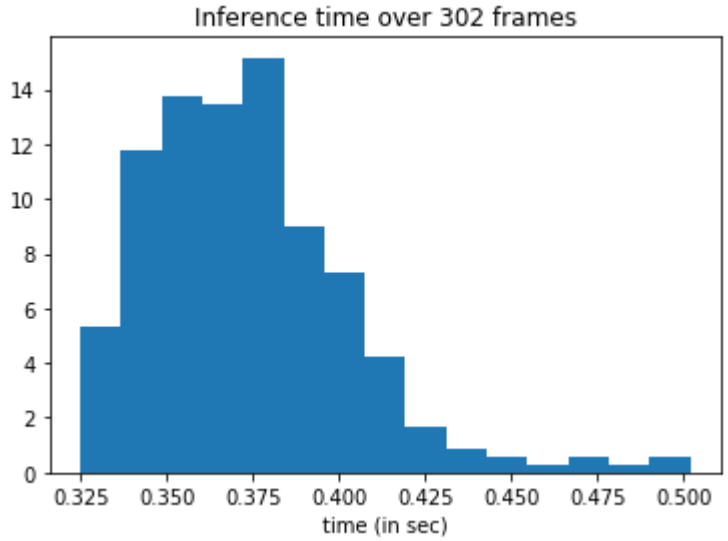


Figure 12 : Distribution des temps d'inférences pour les implémentations Keras

```
Script ended in 218.72s
Average time of writing into out video (604 frames) : 0.083s (std 0.0086)
Average time of getting next frame (604 frames) : 0.021s (std 0.0121)
Average time of preprocessing a frame (302 frames) : 0.009s (std 0.0016)
Average time of prediction on a frame (302 frames) : 0.374s (std 0.0290)
Average time of postprocess bboxes (302 frames) : 0.007s (std 0.0009)
Average time of drawing bboxes on a frame (604 frames) : 0.001s (std 0.0008)
```

Figure 13 : Sortie du script pour l'analyse de 302 images (pour 604 images dans la vidéo ~ 20sec) :

Nous pouvons constater que le temps d'inférence moyen tend vers 0,374 seconde pour une prédiction prenant une grande partie du pipeline de prédiction.

V) Bilan de projet

5.1) Difficultés rencontrées

5.1.1. Prise en main du fonctionnement du modèle YoloV3

5.1.2. Remaniement du modèle YoloV3

Résumé : Difficulté pour faire fonctionner un réseau de neuronne Keras retouché

5.1.3. Calcul de la fonction de cout

La difficulté pour implémenter la fonction de coût c'est qu'il s'agissait d'une fonction très personnalisée. De plus, le code du modèle YoloV3 de YunYang1994 n'avait pas une fonction de coût très propre.

La recherche et bien comprendre comment créer une fonction custom en Keras a été difficile, mais nous avons réussi à l'implémenter grâce au « compile » suivant :

```
model.compile(optimizer=optimizer,
                      loss={
                          'tf_op_layer_concat_4': lambda y_true, y_pred: yolo_loss(y_true,
y_pred), # 52x52
                          'tf_op_layer_concat_7': lambda y_true, y_pred: yolo_loss(y_true,
y_pred), # 26x26
                          'tf_op_layer_concat_10': lambda y_true, y_pred: yolo_loss(y_true,
y_pred),# 13x13
                      },
                      metrics=metrics_poacher
                     )
```

5.1.4. Utilisation d'un modèle Keras dans un pipeline OpenCV

Lors de la tentative d'intégration du modèle Keras sur le pipeline développé à la base avec OpenCV, nous avons rencontré différents problèmes qui nous on poussait à utiliser un pipeline distinct pour le modèle Keras :

1. Les sorties des modèles n'étaient pas exactement les mêmes : X, Y, W et H était en ratio pour opencv alors qu'ils étaient en pixel pour Keras
2. La probabilité sur laquelle se basait le modèle opencv était celle de la classe « personne » alors que pour Keras c'était d'abord celle de l'objet puis celle d'une « personne »

3. Enfin, les images en entrée n'étaient pas pre-processaient de la même façon, les deux étaient redimensionnés en 416x416, mais pour opencv il s'agissait de déformer l'image, alors que pour Keras un padding automatique était ajouté (voir ci-dessous)

Figure 14 : Exemple de pré-process par OpenCV

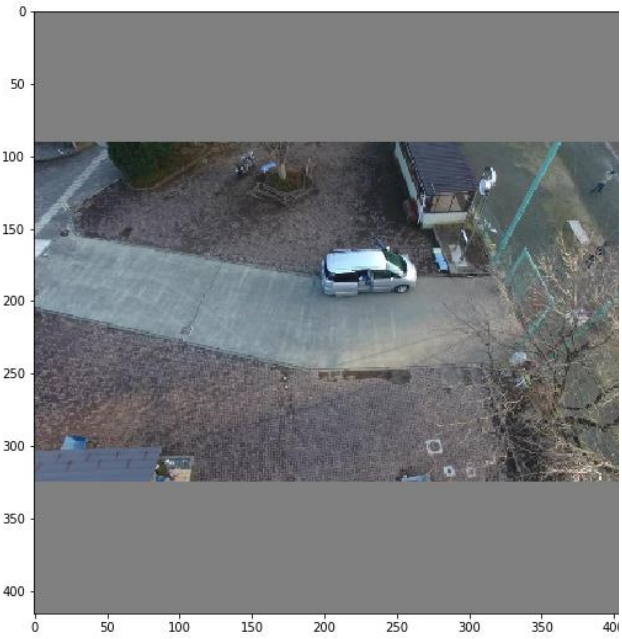


Figure 15 : Exemple de pré-process par Keras

5.1.5. Hétérogénéité des compétences

5.2) Évolutions envisagées

Nous avons commencé à implémenter un MLFlow pour automatiser la gestion d'expérience avec Keras le problème étant que nous ne pouvions pas utiliser les logs automatiques puisque nous ne pouvions pas faire appel à la fonction "fit" (avec cette fonction toutes les données doivent être chargées et donc cela fait planter pour des problèmes de mémoire).

Nous envisageons donc de personnaliser l'utilisation de MLFlow pour suivre les expériences.

De plus, nous allons mettre en place le modèle sous une application Flask pour construire une API intégrable directement sur la Raspberry PI des drones.

VI) Conclusion