# Rapport de projet 8 :

# Développez une preuve de concept

**Réalisé par:  Taher Haggui**

# Introduction

Ce projet a pour objectif de mettre en oeuvre une méthode plus récente , capable de fournir des performances meilleures qu'une méthode utilisée pour un jeu de données donné.
Nous avons choisi de revenir au problématique du projet 7, qui consiste à réaliser un algorithme de détection de race du chien sur une photo. Nous avons utilisé dans celui-ci une approche classique et une autre, basée sur les réseaux de neurones de convolution. Cette dernière a débouché sur les meilleures performances qui frôlent 60 % de précision . Ainsi en guise de convnet appliqué sur le jeu de données du projet 7, nous allons mettre en oeuvre une nouvelle méthode , permettant de livrer des résultats meilleurs que celle de convnet.
Nous allons se pencher vers le domaine de détection d'objets pour utiliser le state of the art de celui-ci dans notre travail. Plus précisément, nous allons utiliser l'algorithme le plus récent de détection d'objets, qui est Mask RCNN. Cet algorithme, nous permettra dans un premier temps de localiser les chiens dans nos photos tout en fournissant les coordonnées de rectangle qui les entoures . Celui-ci, nous permettra dans un second temps d'extraire la zone la plus intéressante , qui correspond au rectangle qui entoure le chien . Par la suite, nous utilisons l'architecture de convnet du projet 7, pour l'entraîner sur les nouvelles photos affinées et mesurer la performance de cette nouvelle approche sur les données de validation.

# I. Introduction au domaine de détection d'objets:

Combien de temps avez-vous passé à chercher les clés de votre chambre perdue dans une maison désordonnée et en désordre? Cela arrive aux meilleurs d'entre nous et jusqu'à ce que la date reste une expérience incroyablement frustrante. Mais que se passerait-il si un simple algorithme informatique pouvait localiser vos clés en quelques millisecondes?

C'est la puissance des algorithmes de détection d'objets. Bien qu'il s'agisse d'un exemple simple, les applications de la détection d'objets s'appliquent à de nombreux secteurs, allant de la surveillance 24 heures sur 24 à la détection de véhicules en temps réel dans les villes intelligentes. En bref, ce sont de puissants algorithmes d'apprentissage en profondeur.

Dans cet partie, nous allons présenter une illustration de comment fonctionne les algorithmes de détection d'objets ainsi que le state of the art de celui-ci.

## 1. Illustration de fonctionnement des algorithmes de détection d'objets:

L'image ci-dessous est un exemple populaire d'illustration du fonctionnement d'un algorithme de détection d'objet. Chaque objet de l'image, d'une personne à un cerf-volant, a été localisé et identifié avec un certain niveau de précision.

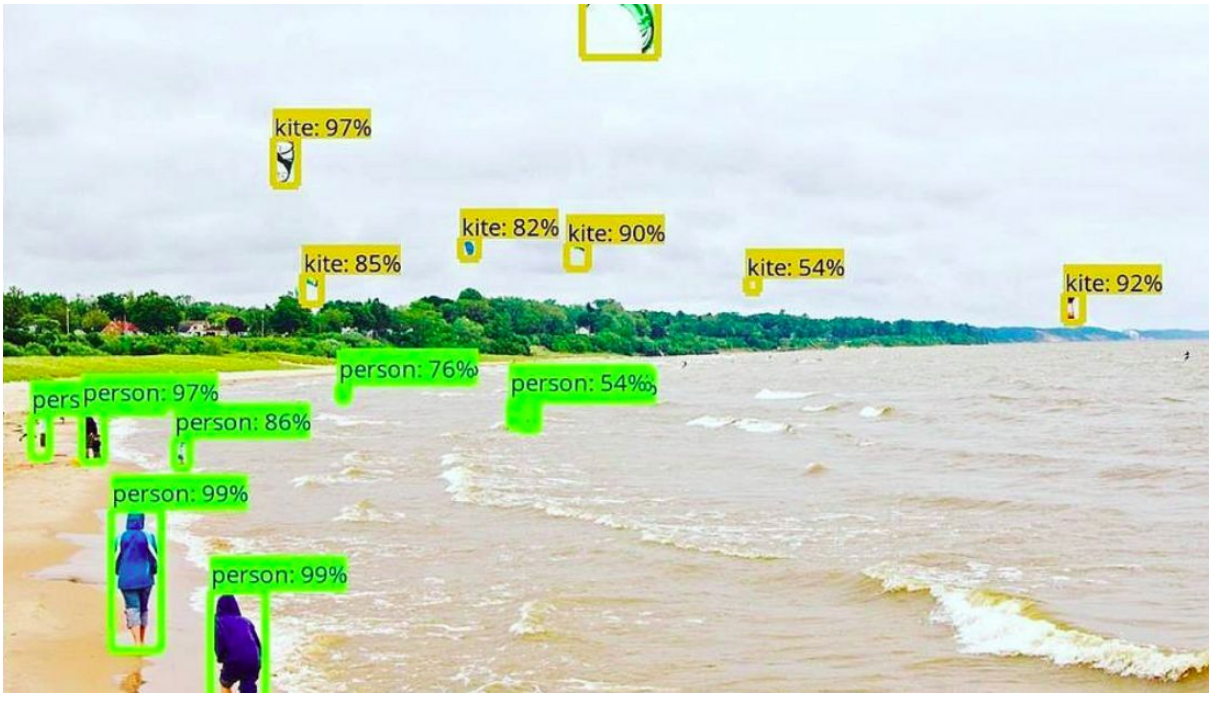



illustration du fonctionnement d'un algorithme de détection d'objets.

### a. CNN pour détection d'objets:

Commençons par la méthode d'apprentissage en profondeur la plus simple et la plus largement utilisée pour détecter des objets dans des images qui sont les réseaux de neurones convolutionnels ou CNN.

Dans un CNN ,nous transmettons une image au réseau, qui la transmet ensuite à travers diverses convolutions et couches de pooling. Enfin, nous obtenons la sortie sous la forme de la classe de l'objet. Assez simple, n'est-ce pas?

Pour chaque image d'entrée, nous obtenons une classe correspondante en sortie. Peut-on utiliser cette technique pour détecter divers objets dans une image? Oui nous pouvons! Voyons comment résoudre un problème général de détection d'objet à l'aide d'un CNN.

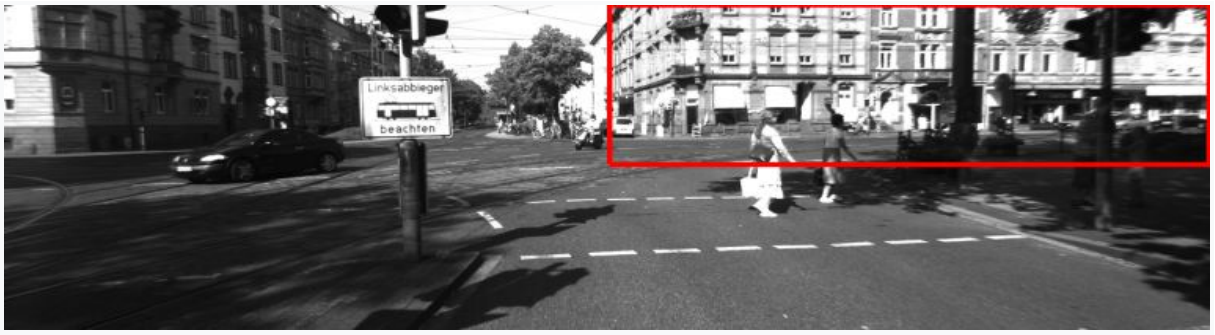
1. Premièrement, nous prenons une image en entrée:


2. Ensuite, nous divisons l'image en différentes régions:


3. Nous considérerons ensuite chaque région comme une image distincte.

4. Transmettez toutes ces régions (images) au CNN et les classer dans différentes classes.

5. Une fois que nous avons affecté chaque région en sa classe correspondante, nous pouvons combiner toutes ces régions pour obtenir l'image d'origine avec les objets détectés:




Le problème avec cette approche est que les objets dans l'image peuvent avoir différents formats et différents emplacements . Par exemple, dans certains cas, l'objet peut couvrir la plus grande partie de l'image, alors que dans d'autres, il se peut qu'il ne recouvre qu'un faible pourcentage de l'image. Les formes des objets peuvent également être différentes (cela se produit souvent dans des cas d'utilisation réels).

En raison de ces facteurs, nous aurions besoin d'un très grand nombre de régions, ce qui entraînerait un temps de calcul considérable. Donc, pour résoudre ce problème et réduire le nombre de régions, nous pouvons utiliser CNN basé sur les régions (RCNN), qui sélectionne les régions en utilisant une méthode de proposition. Voyons ce que RCNN peut faire pour nous.

### b. RCNN pour détection d'objets:

Au lieu de travailler sur un grand nombre de régions, l'algorithme RCNN propose un ensemble de cases dans l'image et vérifie si l'une d'elles contient un objet. RCNN utilise la recherche sélective pour extraire ces zones d'une image (ces zones sont appelées régions).

Voyons d'abord ce qu'est la recherche sélective et comment elle identifie les différentes régions. Il existe essentiellement quatre régions qui forment un objet: échelles, couleurs, textures et enceintes variables. La recherche sélective identifie ces motifs dans l'image et, à partir de cela, propose diverses régions. Voici un bref aperçu du fonctionnement de la recherche sélective:

- Il prend d'abord une image en entrée:

- Ensuite, il génère des sous-segmentations initiales afin que nous ayons plusieurs régions à partir de cette image:

- La technique combine ensuite les régions similaires pour former une région plus grande (basée sur la similarité de couleur, la similarité de texture, la similarité de taille et la compatibilité de forme):

- Enfin, ces régions produisent ensuite les emplacements d'objets finaux (région d'intérêt).

Vous trouverez ci-dessous un résumé succinct des étapes suivies dans RCNN pour détecter des objets:

➔ Nous prenons d'abord un réseau de neurones convolutionnels pré-entraînés.

➔ Ensuite, ce modèle est recyclé. Nous formons la dernière couche du réseau en fonction du nombre de classes à détecter.

➔ La troisième étape consiste à obtenir la région d'intérêt pour chaque image. Nous modelons ensuite toutes ces régions afin qu'elles puissent correspondre à la taille d'entrée CNN.

➔ Après avoir obtenu les régions, nous entraînons SVM à la classification des objets et de l'arrière-plan. Pour chaque classe, nous entraînons un SVM binaire.

➔ Enfin, nous formons un modèle de régression linéaire pour générer des cadres de délimitation plus étroits pour chaque objet identifié dans l'image.

Jusqu'à présent, nous avons vu comment le RCNN peut être utile pour la détection d'objets. Mais cette technique a ses propres limites. L'entraînement d'un modèle RCNN est coûteuse et lente à cause des étapes ci-dessous:

- Extraction de 2 000 régions pour chaque image en utilisant la méthode "selective search "
- Extraction de features à l'aide de CNN pour chaque région d'image. Supposons que nous ayons N images, le nombre de CNN features sera alors de N * 2 000.
- L'ensemble du processus de détection d'objet à l'aide de RCNN repose sur trois modèles:
  - ➢ CNN pour l'extraction de features.
  - ➢ Classificateur SVM linéaire pour l'identification d'objets.
  - ➢ Modèle de régression pour le resserrement des cadres de sélection.

Tous ces processus s'associent pour rendre le RCNN très lent. Il faut environ 40 à 50 secondes pour faire des prévisions pour chaque nouvelle image, ce qui rend le modèle encombrant et pratiquement impossible à construire face à un gigantesque ensemble de données.

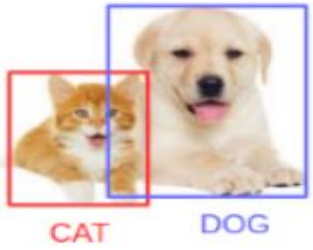
Voici la bonne nouvelle: nous disposons d'une autre technique de détection d'objets qui corrige la plupart des limitations observées dans RCNN.

## 2. State of the art détection d'objets: Mask RCNN

Plusieurs algorithmes se sont succédé pour pallier aux limites de la méthode RCNN basique. Parmi ces algorithmes , nous trouvons Fast RCNN, Faster RCNN et le state of the art Mask RCNN.

Mask R-CNN est essentiellement une extension de Faster R-CNN. Faster R-CNN est largement utilisé pour les tâches de détection d'objets. Pour une image donnée, elle renvoie les coordonnées de l'étiquette de classe et du cadre de sélection pour chaque objet de l'image. Alors, disons que vous transmettez l'image suivante:

Le modèle Fast R-CNN renverra quelque chose comme ceci:




Le framework Mask R-CNN est construit sur Faster R-CNN. Ainsi, pour une image donnée, le masque R-CNN, en plus des coordonnées de l'étiquette de classe et du cadre de sélection pour chaque objet, renverra également le masque d'objet.

# II. jeu de données et méthode de baseline:

## 1. Jeu de données :

Nous choisissons le problématique du projet 7 , qui consiste à entraîner un algorithme sur le jeu de données **Stanford Dogs Dataset** et que celui-ci doit être capable de détecter la race du chien sur une photo.

## 2. Méthode de baseline:

Nous avons choisi comme baseline de réentraîner le réseau pré-entraîné VGG16 sur notre jeu de données **Stanford Dogs Dataset .** Nous avons décidé par la suite de garder les poids de cinq premières couches de réseau VGG16 et réentrainer toutes les autres couches de convolutions et de pooling sur notre jeu de données, qui correspondent à l'extraction des couches les plus complexes . Par la suite, nous avons ajoutés 3 couches fully connected au top de notre réseau. Les paramètres de ces trois couches fully connected sont :

- 1ére couche : output =210 et une fonction de Relu comme activation.
- 2éme couche: output=200 et une fonction de Relu comme activation.
- 3éme couche : output=120, correspond au nombre de classe à prédire et une fonction softmax comme activation.

Pour la compilation de ce réseau , nous avons utilisé les paramètres suivants:

- La fonction de loss : nous avons choisi la fonction "cross entropy".
- Optimiseur: Nous avons utilisé la fonction stochastic gradient descent (SGD) avec les paramètres suivants:
  - ➢ taux d'apprentissage: 0.001
  - ➢ momentum: 0.9

Ainsi nous avons décidé d'entraîner notre convnet sur 70% de notre jeu de données , pendant 600 epochs et de tester ses performances sur les 30% restants. Celui-ci a débouché sur les résultats suivant:

- taux de précision (Accuracy) : 57%
- valeur d'erreur (Loss): 1.6

# III. Nouvelle approche:

Le but de ce projet est de trouver une nouvelle méthode , permettant d'améliorer les performances de la méthode baseline . Ainsi pour ce faire, nous nous sommes inspiré par le domaine de détection d'objets pour définir une nouvelle approche , faisant appel à la state of the art de ce domaine . Dans la suite, nous présentons la mise en oeuvre de cette nouvelle approche , basée essentiellement sur la méthode Mask RCNN.

## 1. Mise en oeuvre :

L'algorithme Mask RCNN ,permet de retourner la classe de chaque objet ainsi que les coordonnées du rectangle qui entoure intimement l'objet. Celui-ci nous a amené à utiliser une nouvelle approche qui consiste à combiner un algorithme Mask RCNN et un convnet classique dont celui de notre baseline pour résoudre notre problématique. En fait, les images contiennent plus d'objets et d'éléments que celui de l'objet intéressant , qui correspond à l'objet à prédire son classe . Ainsi tous ces éléments peuvent induire de biais au cours de la phase d'apprentissage et de définition des éléments clés qui distinguent nos classes . Pour éviter ce problème et laisser le convnet se concentre sur les objets intéressant , correspondent à ceux à classer . Cette nouvelle approche consiste à utiliser tout d'abords Mask RCNN , pour extraire la partie intéressante de chaque image, correspond à la partie qui entoure l'objet à classer. Par la suite , nous entraînons notre convnet classique sur les nouvelles images affinées.

## 2. Mise en pratique:

Pour la mise en pratique de notre approche, nous procédons tout d'abords pour préparer le réseau Mask RCNN comme suit :

**étape 1: cloner le Mask RCNN repistory**

Celui-ci contient l'architecture de l'algorithme Mask RCNN. Pour réaliser cette étape, nous devons utiliser la commande suivante:

git clone https://github.com/matterport/Mask_RCNN.git

**étape 2: Télécharger les poids de réseau pré-entraîné sur MS COCO**

Ensuite, nous devons télécharger les poids pré-entraînés. Nous avons utilisé ce lien pour télécharger les poids pré-entraînés. Ces poids sont obtenus à partir d'un modèle entraîné sur l'ensemble de données MS COCO. Une fois que le fichier des poids, soit téléchargé . Nous collons le dans le dossier Mask_RCNN que nous avons cloné à l'étape 1 .

**étape 3 : Configurer un modèle et faire une prédiction**

Tout d'abord, le modèle doit être défini via une classe MaskRCNN.

Cette classe nécessite un objet de configuration en tant que paramètre. L'objet de configuration définit comment le modèle peut être utilisé lors de l'entraînement ou de prédiction.

Dans ce cas, la configuration ne spécifie que le nombre d'images par lot, ce qui sera une, et le nombre de classes à prédire.

```
# define the test configuration
class TestConfig(Config):
    NAME = "test"
    GPU_COUNT = 1
    IMAGES_PER_GPU = 1
    NUM_CLASSES = 1 + 80
```

Nous pouvons maintenant définir l'instance MaskRCNN.

Nous définirons le modèle comme type «inférence» indiquant que nous sommes intéressés à faire des prédictions et non à faire d'entraînement . Nous devons également spécifier un répertoire dans lequel tous les messages d'erreures peuvent être écrits, qui dans ce cas sera le répertoire de travail actuel.

```
# define the model
rcnn = MaskRCNN(mode='inference', model_dir='./', config=TestConfig())
```

La prochaine étape consiste à charger les poids que nous avons téléchargés.

```
# load coco model weights
rcnn.load_weights('mask_rcnn_coco.h5', by_name=True)
```

Nous pouvons désormais faire des prédictions pour nos images. Nous devons tout d'abords chargé nos images et les transformer sous la forme de **NumPy array**.

Au lieu d'appeler predict () comme nous le ferions sur un modèle de Keras normal, nous allons appeler la fonction detect () et lui transmettre l'image unique.

```
# make prediction
results = rcnn.detect([img], verbose=0)
```

Le résultat contient un dictionnaire pour chaque image transmise à la fonction detect (), dans ce cas, une liste d'un dictionnaire unique pour une image.

Le dictionnaire comporte des clés pour les cadres de sélection, les masques, etc., et chaque clé pointe vers une liste de plusieurs objets possibles détectés dans l'image.

Les clés du dictionnaire sont les suivantes:

'Rois': les boîtes ou les régions d'intérêt (ROI) liées aux objets détectés.
'Masques': masques des objets détectés.
'Class_ids': Les entiers de classe pour les objets détectés.
'Scores': probabilité ou confiance pour chaque classe prédite.

Après avoir préparé le réseau Mask RCNN, nous avons crée une fonction permettant d'affiner les images d'origine moyennant le Mask RCNN. En fait, cette fonction consiste à :

- Alimenter l'algorithme préparé Mask RCNN par l'image introduite comme paramètre
- tester s'il y aurait parmi les objets détecté, un chien.
- S'il y a un chien , la fonction extraire une nouvelle image , correspond aux pixels qui constituent la zone intéressante (ROI), fournie par Mask RCNN.
- Si non, la fonction garde la même image d'origine avec tous ses pixels.

Maintenant que nous avons transformé nos images de notre jeu de données à des images plus affinées . Nous revenons à notre convnet de la méthode baseline avec la même architecture , la même démarche d'entraînement et les mêmes paramètres de compilation pour le réentrainer sur ces nouvelles images.

# IV. Comparaison des résultats:

Le tableau 1 , récapitule les différents résultats obtenus à travers la méthode baseline et celle de la nouvelle approche.

Tableau 1: Performances de la méthode baseline et celle de la nouvelle approche

| | Accuracy | Loss |
|---|---|---|
| Baseline | 57.7 | 1.63 |
| Nouvelle approche | 53.77 | 1.96 |

Malgré que notre nouvelle approche, permet d'extraire les zones intéressantes qui concernent les chiens dans nos images et de faire le focus sur ces objets pour extraire leur features les plus distinctives , les performances de cette nouvelle approche n'ont pas été meilleure que celles de baseline.

# Conclusion

Dans ce projet , nous avons mis en oeuvre une  nouvelle approche permettant d'extraire les régions intéressante de l'objet à classer . Cependant celle -ci n'a pas pu amélioré les performances de la méthode baseline . Ceci pourrait être expliqué par :

- Le pourcentage des images ou notre algorithme Mask RCNN, n'a pas pu détecter l'objet chien .
- La deuxième raison , c'est le fait que nous sommes face à un problème de classification de race de chien , donc il serait très difficile pour notre algorithme d'extraire les features les plus distinctives de chaque race à cause de la similarité entre les races.

Ainsi nous recommandons pour pallier à ces deux problèmes de :

- ❖ Améliorer notre algorithme pour qu'il puisse détecter des objets connexes à l'objet chien tels que les objets chat , cheval , mouton dans le cas où il ne pouvait pas détecter l'objet chien .
- ❖ Faire des expérimentations de cette approche sur d'autres problèmes de classification lié à des catégories différentes.