# Travail effectué :

Dans cette partie nous allons procéder à l’implémentation d’un modèle R-CNN similaire à celui proposé par Ross Girshick et al [[1]](#footnote-1) que nous appellerons dans la suite « **modèle de base** » ou « **modèle de référence** » tout en apportant quelques modifications en l’adoptant à notre cas d’utilisation à savoir à des fins pédagogiques.

## Présentation des données :

Nous allons travailler sur la même base d’images utilisée par le modèle de référence à savoir Pascal VOC 2007 [[2]](#footnote-2) Cette base se constitue comme suit :



* 9 963 images entre l’ensemble d’apprentissage (training et validation) qui contient environ 5000 images et l’ensemble de test contient également environ 5000 images.
* Les images sont réparties en 20 classes (parmi les macro-catégories Person, Animal, Vehicle, Indoor), mais avec des étiquettes “multi-labels”. Par exemple, une image peut contenir à la fois une voiture et un vélo.
* 24 640 objets annotés dans les images avec la position des boîtes englobantes.
* Les images sont de tailles variables mais en moyenne de taille 500x300.

## Outils et environnement de travail :

Nous avons effectué l’implémentation du modèle sur un environnement Python 3.6 doté des principaux outils de traitement de données et de Machine Learning (Numpy, Scikit-Learn et Tenseflow).

Pour les outils de Deep Learning nous avons opté pour la librairie Keras [[3]](#footnote-3) 2 avec un Backend Tenserflow. Les calculs et visualisations des résultats se sont fait sur un document Jupyter Notebook détaillant toutes les étapes de la démarche.

## Implémentation du modèle

Nous allons à présent détailler la démarche suivie dans l’implémentation du modèle R-CNN tout en précisant les modifications apportées par rapport au modèle de base et les motivations de ces divergences.

## Région Proposal :

Pour l’algorithme de proposition de régions nous avons opté pour le même choix à savoir le « Selective Search », et ce principalement pour sa rapidité mais aussi son efficacité car il prend en compte les similarités de couleur, de taille et de texture des différentes parties de l’image. Nous avons utilisé une implémentation proposée par « AlpacaDB »[5] qui reprend l’algorithme de J.R.R. Uijlings et al *« Selective Search for Object Recognition »* [6] cité dans le modèle de référence.

Cet algorithme propose 3 différents paramètres :

* **Sigma :** qui est le paramètre écart-type du filtre gaussien utilisé pour atténuer les artéfacts de l’images sans pour autant altérer sa composition.
* **Min\_size :** qui correspond à la taille minimale retrouvé dans les régions proposées.
* **Scale :** un paramètre définissant une échelle sur laquelle la comparaison des histogrammes est faites. Plus ce paramètres et grand plus on favorise des objets de taille importante dans l’image.

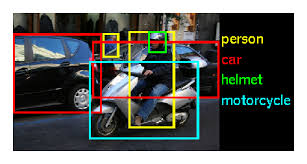
Les différents réglages de ces paramètres permettent de générer un nombre différent de régions pour chaque image. Nous rappelons que dans le modèle de base environs 2000 régions étaient générées par chaque images et ce dans un souci de scalabilités (s’adapter aux images de grande taille).  
Comme dans notre cas nous allons nous limiter aux données de la base Pacal VOC2007, dont la taille moyenne des images et de 300x500 pixels nous avons opté pour le nombre de 400 régions par images ou chaque région est caractérisée par un rectangle ou 4 coordonnées qui correspondent à son positionnement dans l’image d’origine.

Ce choix nous semble judicieux car d’une part il permet de générer un nombre suffisant d’entrées pour l’apprentissage (pour 300 images nous obtenons 60000 régions) et d’autre part le choix d’un nombre plus important engendrerai un temps de traitement extrêmement excessif notamment pour l’extraction des « Deep Features » que nous détaillerons dans la suite.

Ainsi, pour obtenir 200 régions par images nous avons effectué un « grid search » (recherche sur grille) des 3 paramètres définis précédemment. Nous avons donc gardé les paramètres suivants :

* **Sigma** : un écart type 0.2 qui représente la valeur par défaut préconisée par le concepteur.
* **Taille minimum** : 15x15 qui reste en adéquation avec les données traitées, car en fonctions des boîtes englobantes proposées par le dataset les plus petits objets sont à peu près de cette taille.
* **Scale** : en figeant les deux premiers paramètres une échelle fixée à 500 avait abouti aux résultats escomptés.

## Labélisation et région d’intérêt :

La deuxième étape pour préparer l’ensemble d’apprentissage consiste en la labélisation des régions, c’est-à-dire définir pour chaque régions la classe d’objet qu’elle contient (éventuellement) parmi les 20 macro-classes de notre base de données sinon l’étiqueter en tant que « background » ou arrière-plan ce qui reviendra donc à créer une 21eme classe « background ». 

Pour ce faire, nous disposons d’un fichier contenant pour chaque image les différents rectangles ou régions de l’image qui englobe un objet relatif à une classe.

Le but de cette étape est donc d’étiqueter les régions qui contiennent des objets, nous appellerons celles-ci « régions d’intérêt » ou « fenêtres positives », et par opposition les régions ne contenant aucun objet, donc arrière-plan, les fenêtres négatives.

Afin de définir les régions d’intérêt nous avons utilisé un critère de sélection défini comme IoU (intersection over union) ou l’intersection sur l’union qui consiste en la mesure de la superposition d’une région proposée sur une région « vraie » ou « groundtruth » c’est-à-dire qui est labelisée. Si la valeur de cette mesure est supérieure à un certain seuil (généralement 0.5) on peut considérer la région proposée comme étant une région d’intérêt.

Exemple :

## Extraction des « Deep Features » :

La troisième étape de la démarche consiste en l’extraction des caractéristiques profondes à partir des entrées obtenues par « region proposal » en utilisant un CNN pré-entrainé pour une tache de classification (pas de détection d’objet) sur une large base d’image.

### 3.1 Choix du CNN :

Nous avons utilisé le réseaux ResNet50 [7], vainqueur du concours ILSVRC [[4]](#footnote-4) en 2015, le réseau avait atteint un taux d’erreur top-5 de 3.57% pour la tâche de classification sur la base ImageNet qui contient plus de 1,3 millions d’images pour 1000 classes différentes. De plus le vecteur de caractéristique correspondant à la dernière couche avant la couche de sortie du réseau est de taille relativement réduite de 2048.

Pour rappel, le modèle R-CNN utilise le réseau VGG-16 [8] qui a atteint sur la même base d’images un taux d’erreur top-5 de 7.5% et se caractérise par un vecteur de caractéristique de dimension 4096.

Notre choix s’est donc porté donc sur le premier réseau de par sa performance et son efficience en mémoire pour notre cas d’utilisation.

### 3.2 Extraction du vecteur de caractéristiques :

Comme nous avons expliqué précédemment, nous avons opté pour la génération d’un nombre réduit de régions comparé au modèle de base et ce principalement pour des raisons de performance et coût de calcul.

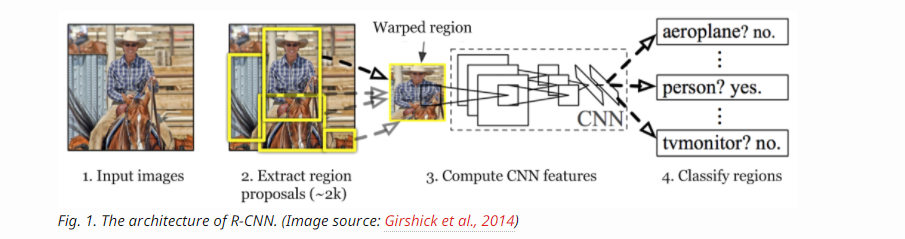
En effet, afin de d’obtenir le vecteur caractéristique d’une seule région le CNN calcule plus de 61 millions de paramètres, pour 300 images nous obtenons 60000 régions et le temps de calcul devient rapidement trop important pour des machines non dédiées (absence de GPU).

Donc, pour chaque image, à l’aide de l’algorithme de « Selective Search » nous générons environs 200 région puis nous effectuons l’étiquetage des régions d’intérêt, ensuite chaque région subit les transformations nécessaires pour correspondre aux entrées du CNN (redimensionnement et centrage) afin d’extraire le vecteur d’entrée.

Une fois toutes les images traitées nous enregistrons dans un fichiers tous les vecteurs caractéristiques obtenus avec les labels de classes correspondant.

En traitant les images d’apprentissage et de test nous obtenons donc deux fichiers qui correspondent à la base d’apprentissage et de test de notre classifieur.

Le schéma de cette étape se présente donc comme suit :



## Apprentissage du classifieur :

Cette dernière étape consiste à mettre en place un classifieur qui prend en entrée les vecteurs caractéristiques (correspondant à chaque régions) obtenus pendant l’étape précédente et en sortie les 20 classes de notre dataset en plus de la classe background.

### 4.1 Architecture du modèle :

Pour le choix du modèle nous avons utilisé un modèle de régression logistique pour apprendre toutes les classes en même temps. Par définition pour chaque région correspond un seul label, cependant nous avons opté pour un modèle avec des sorties en multi-label et ce pour se rapprocher du modèle de base ou on utilise un SVM binaire par classe. L’architecture se résume comme suit:

* Un réseau de neurones entièrement connectées sans couches cachées : un vecteur d’entrée de dimension 2048 et une couche de sortie de taille 21, en tout nous disposons de 43 029 paramètres à estimer.
* Pour la fonction de coût nous avons utilisé l’entropie croisée binaire avec un seuil de 0.5 adaptée aux contexte multi-label.
* Une fonction d’activation sigmoïde en adéquation à l’usage de l’entropie croisée binaire.

### 4.2 Hyperparamètres et validation :

Pour l’apprentissage nous disposons d’un peu plus de 93000 exemples, cependant la répartition de ces derniers est très déséquilibrée, en effet pour chaque image les fenêtres positives proposées sont extrêmement rares en comparaison aux fenêtres négatives d’arrière-plan. Nous disposions au final d’un ratio de 1/30 pour les exemples positifs et négatifs. Ce déséquilibre rend la convergence très lente voire impossible.

C’est pourquoi nous avons opté pour un apprentissage par mini-batch ou chaque mini batch est constitué pendant l’apprentissage en associant un vecteur de taille 32 comportant des exemples positifs et un autre de taille 96 comportant des exemples négatifs donnant ainsi lieu un batch de taille 128.

Cependant, cette technique risque d’engendrer un surapprentissage notamment des exemples positifs qui apparaissent plus souvent durant l’apprentissage, c’est pourquoi nous avons introduit un « Early stoping » calibré sur l’erreur d’un sous ensemble de validation (1/4 de l’ensemble d’apprentissage).

Enfin pour le pas d’apprentissage, nous avons utilisé « ADAM », une variante optimisées des SGD, en conservant les paramètres par défaut (0.01 pour le pas, et 0.8 pour le Momentum). Avec cette configuration nous avons remarqué une nette amélioration dans le temps de convergence.

### 4.3 Evaluation du modèle :

# Références

[1] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. in ICCV, 2015

[5] AlpacaDB: Selective Search Implementation for Python. <https://github.com/AlpacaDB/selectivesearch>

[6] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. IJCV, 2013.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition, ILSVRC, 2015

[8] Very Deep Convolutional Networks for Large-Scale Image Recognition. <https://arxiv.org/abs/1409.1556>

1. Rich feature hierarchies for accurate object detection and semantic segmentation [↑](#footnote-ref-1)
2. PASCAL VOC2007: http://host.robots.ox.ac.uk/pascal/VOC/voc2007/ [↑](#footnote-ref-2)
3. Keras [↑](#footnote-ref-3)
4. ImageNet Large Scale Visual Recognition Competition: [www.image-net.org/challenges/LSVRC/](http://www.image-net.org/challenges/LSVRC/) [↑](#footnote-ref-4)