

Une expérimentation autour de l'exploitation de données images annotées dans une architecture client-serveur.

Christophe Saint-Jean Julien Thérin



Journée Scientifique **Python**

14 Juin 2017 - Université de La Rochelle

Motivation applicative

Écrire un programme informatique capable de décrire une image en produisant des annotations.

Contexte du projet

- ➊ Plateforme de navigation autonome : système de pilotage autonome et/ou assisté (CPERs ECONAT et NUMERIC)



- ➋ Le temps réel n'est pas crucial
- ➌ Plateforme embarquée

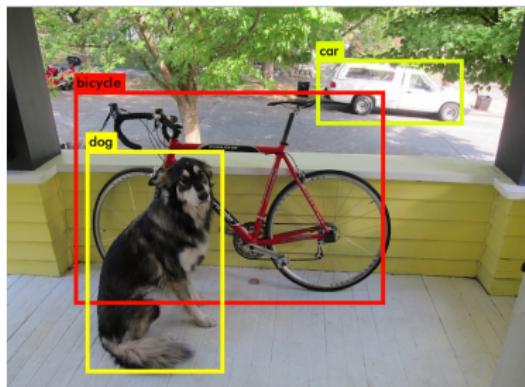
Contexte du projet

- ① Plateforme de navigation autonome : système de pilotage autonome et/ou assisté (CPERs ECONAT et NUMERIC)

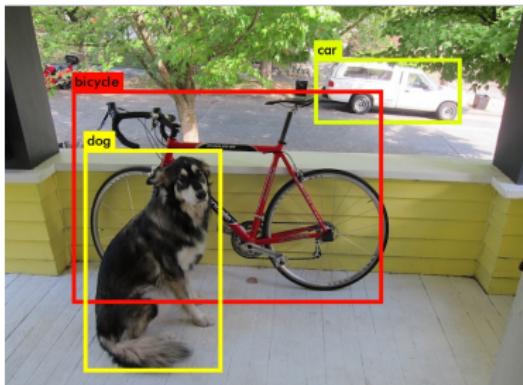


- ② Le temps réel n'est pas crucial
- ③ Plateforme embarquée

De nombreuses approches sont possibles, nous parlerons aujourd'hui d'**apprentissage profond** et du modèle **Tiny Yolo** ("You Only Look Once").



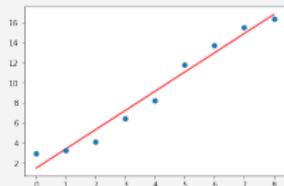
De nombreuses approches sont possibles, nous parlerons aujourd'hui d'**apprentissage profond** et du modèle **Tiny Yolo** ("You Only Look Once").



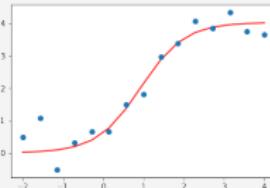
Le modèle Yolo formalise la prédiction d'annotations comme un **problème de régression** paramétrique:

$$f_{\theta}: x \mapsto y$$

Rappel:



linregress



curve_fit

Les outils de Scipy sont peu adaptés à l'optimisation séquentielle et aux données massives.

Le modèle Yolo formalise la prédiction d'annotations comme un **problème de régression** paramétrique:

$$f_{\theta}: x \mapsto y$$

Pour Tiny Yolo:

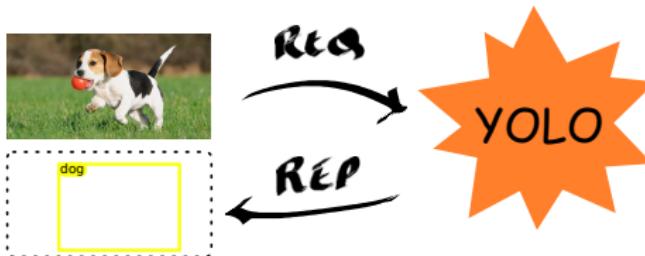
- la donnée d'entrée x est une image couleur 448×448
- la sortie y , l'annotation encodée sous la forme d'un vecteur numérique.

Probabilités / Classes

Confiance

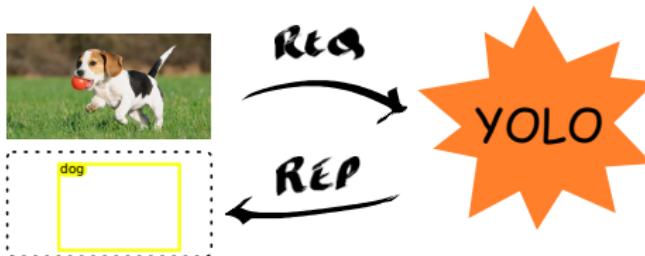
Coordonnées

Il s'agit de produire une annotation pour une image:



- Serveur : Tiny Yolo défini en Keras
- Client : N'importe quel client qui peut formuler une requête { $x = \text{image}$, $p_boxes = [\text{zone à prédire}]^*$ } au format JSON
 - Bases de données images / Fichiers
 - Interface graphique / Webcam
 - URL / Caméra en ligne
- Communication entre clients et serveur : Queue 0-MQ

Il s'agit de produire une annotation pour une image:

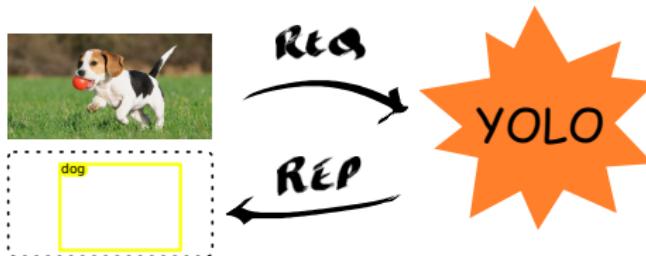


- Serveur : Tiny Yolo défini en Keras
- Client : N'importe quel client qui peut formuler une requête { $x = \text{image}$, $p_boxes = [\text{zone à prédire}]^*$ } au format JSON
 - Bases de données images / Fichiers
 - Interface graphique / Webcam
 - URL / Caméra en ligne
- Communication entre clients et serveur : Queue 0-MQ



Architecture du système : mode prédiction

Il s'agit de produire une annotation pour une image:

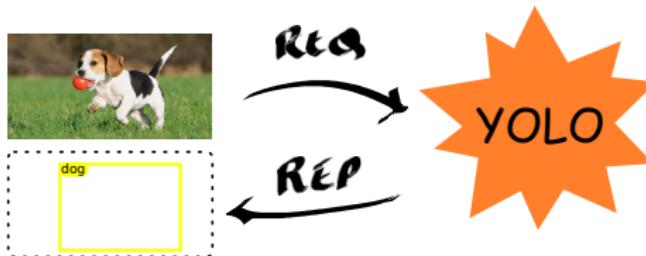


- Serveur : Tiny Yolo défini en Keras
- Client : N'importe quel client qui peut formuler une requête { $x = \text{image}$, $p_boxes = [\text{zone à prédire}]^*$ } au format JSON
 - Bases de données images / Fichiers
 - Interface graphique / Webcam
 - URL / Caméra en ligne
- Communication entre clients et serveur : Queue 0-MQ



Architecture du système : mode prédiction

Il s'agit de produire une annotation pour une image:

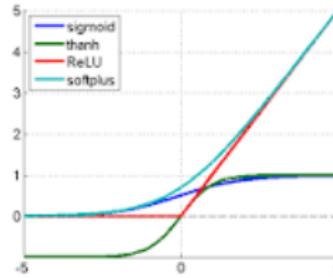
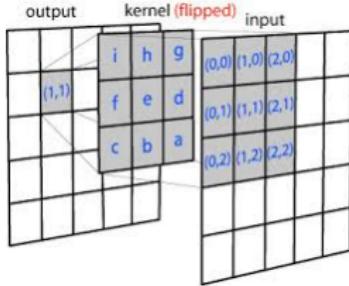


- Serveur : Tiny Yolo défini en Keras
- Client : N'importe quel client qui peut formuler une requête { $x = \text{image}$, $p_boxes = [\text{zone à prédire}]^*$ } au format JSON
 - Bases de données images / Fichiers
 - Interface graphique / Webcam
 - URL / Caméra en ligne
- Communication entre clients et serveur : Queue **0-MQ**

Pourquoi utiliser Keras K?

- Bibliothèque de haut niveau (TensorFlow, Theano, etc)
- Simple, bien documentée, code accessible \Rightarrow **populaire !**
- Peut être embarquée dans un contexte industriel.

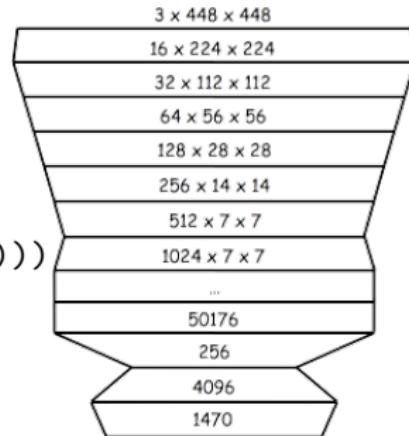
Dans les modèles d'apprentissage profond, le modèle est souvent une **composition de fonctions** linéaires (convolution, produit matriciel, etc) et non linéaires (activation ReLu, SoftMax, etc) **différentiables**.



Extrait de code :

```
model = Sequential()
model.add(Conv2D(filters=16,
                 kernel_size=(3, 3),
                 input_shape=(3, 448, 448)))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D(pool_size=(2,2)))
...
model.add(Dense(1470))

model.load('params.hdf5')
y_pred = model.predict(x)
```

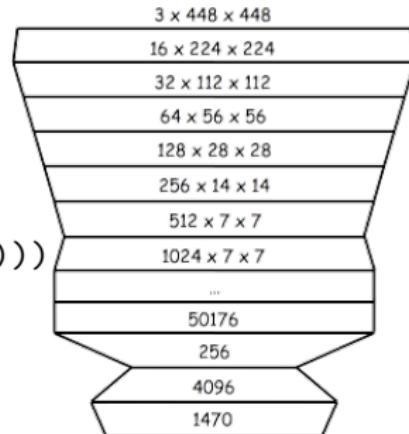


En résumé, Tiny Yolo c'est :

Extrait de code :

```
model = Sequential()
model.add(Conv2D(filters=16,
                 kernel_size=(3, 3),
                 input_shape=(3, 448, 448)))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D(pool_size=(2,2)))
...
model.add(Dense(1470))

model.load('params.hdf5')
y_pred = model.predict(x)
```



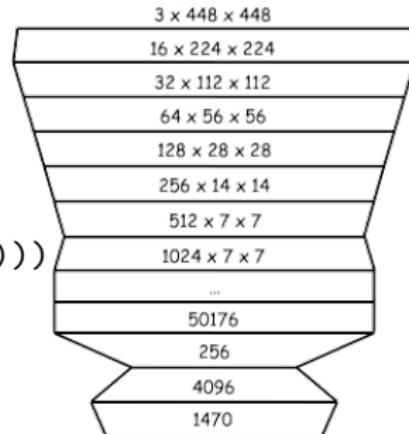
En résumé, Tiny Yolo c'est :

- $x : 448 \times 448 \times 3 = 602.112$ prédicteurs

Extrait de code :

```
model = Sequential()
model.add(Conv2D(filters=16,
                 kernel_size=(3, 3),
                 input_shape=(3, 448, 448)))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D(pool_size=(2,2)))
...
model.add(Dense(1470))

model.load('params.hdf5')
y_pred = model.predict(x)
```



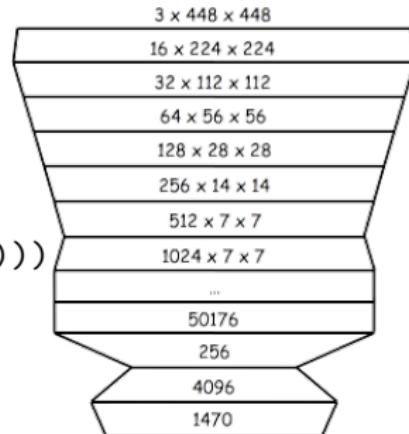
En résumé, Tiny Yolo c'est :

- $x : 448 \times 448 \times 3 = 602.112$ prédicteurs
- $y : 1470$ variables à prédire (suivant nombre de classes)

Extrait de code :

```
model = Sequential()
model.add(Conv2D(filters=16,
                 kernel_size=(3, 3),
                 input_shape=(3, 448, 448)))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D(pool_size=(2,2)))
...
model.add(Dense(1470))

model.load('params.hdf5')
y_pred = model.predict(x)
```



En résumé, Tiny Yolo c'est :

- $x : 448 \times 448 \times 3 = 602.112$ prédicteurs
- $y : 1470$ variables à prédire (suivant nombre de classes)
- et 45.089.374 paramètres !!!



Exemple minimal de client (Prédiction)

Nous avons choisi le format d'échange JSON:

Avantages

- Interopérable à l'extérieur de Python
- Réprésentation par une chaîne de caractères

Inconvénients

- Objets simples
- Pas de compression

Dans le module `json`, on a seulement besoin de:

`dumps` : encodage

`loads` : décodage

→ Démo



Exemple minimal de client (Prédiction)

Nous avons choisi le format d'échange JSON:

Avantages

- Interopérable à l'extérieur de Python
- Réprésentation par une chaîne de caractères

Inconvénients

- Objets simples
- Pas de compression

Dans le module **json**, on a seulement besoin de:

`dumps` : encodage

`loads` : décodage

▶ Démo !

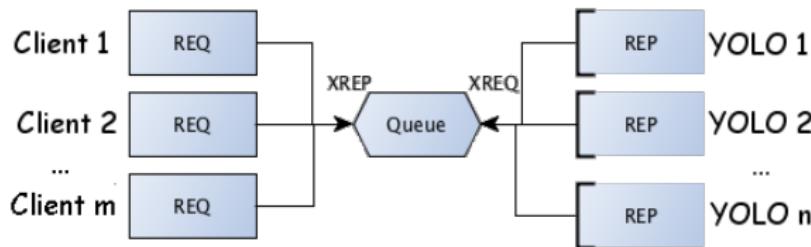


Communication Client-Serveur (Prédiction)

Nous avons choisi la bibliothèque 0-MQ (**pymq**)

- En C++, mais interfacée avec tous les langages.
- Supporte la communication intra- et inter-machines.
- Rapide (entrées-sorties asynchrones).
- Supporte le JSON !
- Implémente des motifs de communication haut niveau !

Nous avons seulement besoin du device Queue:

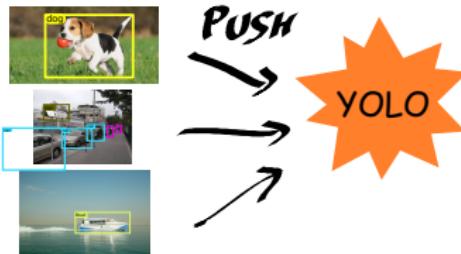


▶ Démo !



Architecture du système : mode entraînement

On a besoin d'un **MAXIMUM** d'images annotées:

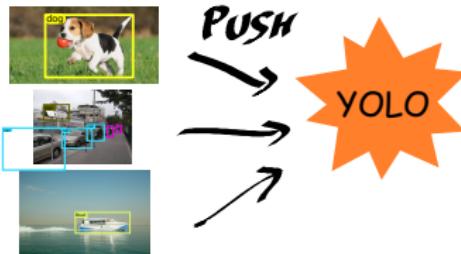


- Serveur : Tiny Yolo défini en **Keras** +
- Producteurs d'images annotées:
 - Bases d'images (ex: VOC img+xml via **Beautiful Soup**).
 - Annotations utilisateurs spécifiques à la tâche visée.
 $(x = \text{image}, b_boxes = \{(x_1, y_1, x_2, y_2)\}^+, \text{labels} = \{0..K\}^+)$
- Communication entre clients et serveur : Streamer **0-MQ**



Architecture du système : mode entraînement

On a besoin d'un **MAXIMUM** d'images annotées:

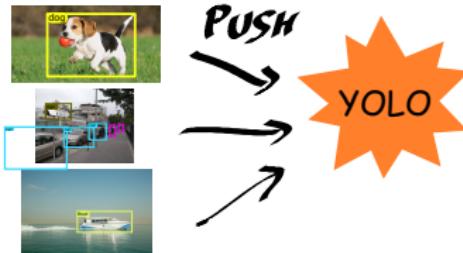


- Serveur : Tiny Yolo défini en **Keras** +
- Producteurs d'images annotées:
 - Bases d'images (ex: VOC img+xml via **Beautiful Soup**).
 - Annotations utilisateurs spécifiques à la tâche visée.
 $(x = \text{image}, b_boxes = \{(x_1, y_1, x_2, y_2)\}^+, \text{labels} = \{0..K\}^+)$
- Communication entre clients et serveur : Streamer **0-MQ**



Architecture du système : mode entraînement

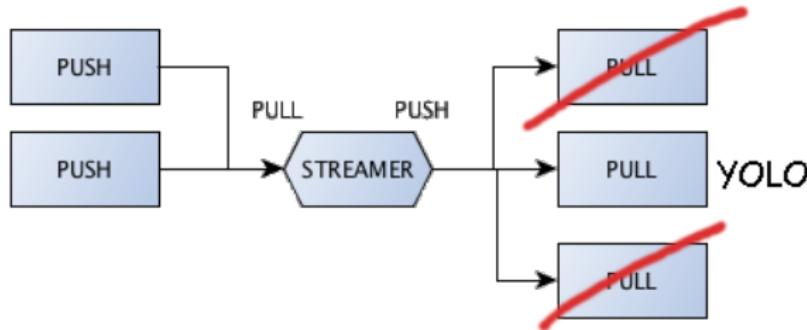
On a besoin d'un **MAXIMUM** d'images annotées:



- Serveur : Tiny Yolo défini en **Keras** +
- Producteurs d'images annotées:
 - Bases d'images (ex: VOC img+xml via **Beautiful Soup**).
 - Annotations utilisateurs spécifiques à la tâche visée.
 $(x = \text{image}, b_boxes = \{(x_1, y_1, x_2, y_2)\}^+, \text{labels} = \{0..K\}^+)$
- Communication entre clients et serveur : Streamer **0-MQ**

Nous avons seulement besoin du device Streamer:

- Communication unidirectionnelle.
- Protocole JSON.
- Un seul serveur Yolo.



Lien vers l'exemple Streamer **pyzmq**

Remarque: *C'est un motif pour la répartition de tâches...*



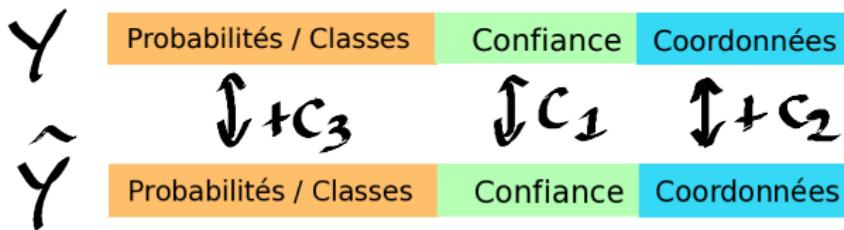
Quelques tâches du serveur:

- Encodage de l'annotation JSON vers un vecteur y .
- Calcul de coût de l'erreur:
- Mettre à jour les 45.089.374 paramètres pour minimiser l'erreur.
- Lots, sauvegarde périodique des paramètres, monitoring, etc



Quelques tâches du serveur:

- Encodage de l'annotation JSON vers un vecteur y .
- Calcul de coût de l'erreur:

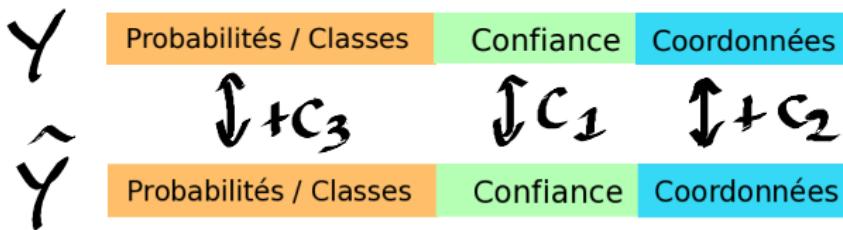


- Mettre à jour les 45.089.374 paramètres pour minimiser l'erreur.
- Lots, sauvegarde périodique des paramètres, monitoring, etc



Quelques tâches du serveur:

- Encodage de l'annotation JSON vers un vecteur y .
- Calcul de coût de l'erreur:

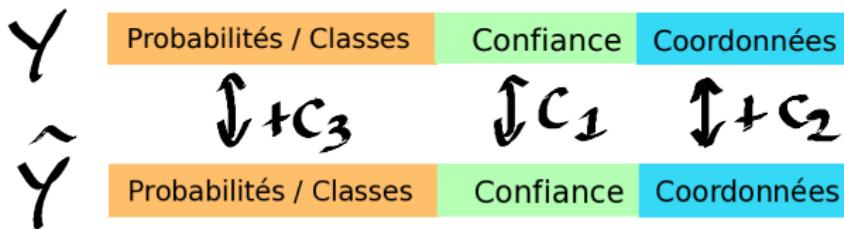


- Mettre à jour les 45.089.374 paramètres pour minimiser l'erreur.
- Lots, sauvegarde périodique des paramètres, monitoring, etc



Quelques tâches du serveur:

- Encodage de l'annotation JSON vers un vecteur y .
- Calcul de coût de l'erreur:



- Mettre à jour les 45.089.374 paramètres pour minimiser l'erreur.
- Lots, sauvegarde périodique des paramètres, monitoring, etc



Mais, on va également confier une autre tâche au serveur :

Augmenter le nombre d'images annotées !!

Mais, on va également confier une autre tâche au serveur :

Augmenter le nombre d'images annotées !!



Plutôt facile à reproduire avec **OpenCV**



Mais, on va également confier une autre tâche au serveur :

Augmenter le nombre d'images annotées !!



Beaucoup plus compliqué.

Mais, on va également confier une autre tâche au serveur :

Augmenter le nombre d'images annotées !!



Plutôt facile à reproduire avec **OpenCV**

▶ Démo !

- Présentation de la méthodologie YOLO pour l'annotation d'images.
- Architecture Client/Serveur:
 - Simple : 30 lignes de code, 3 ports ouverts (0-MQ)
 - Puissante : Tout client/fournisseur qui supporte JSON.
 - Adaptable pour d'autres fonctions de serveur
(ex. : Collecte de données)
 - Versatile (YOLO est invisible du client)
- Cadre de développement Recherche:
 - D'autres modèles de régression (architecture, type couches) ...
 - D'autres fonctions de coût ...
 - ... d'autres méthodes d'optimisation.
 - Intégrer des images partielles annotées (ex.: Flickr)



- Présentation de la méthodologie YOLO pour l'annotation d'images.
- Architecture Client/Serveur:
 - Simple : 30 lignes de code, 3 ports ouverts (0-MQ)
 - Puissante : Tout client/fournisseur qui supporte JSON.
 - Adaptable pour d'autres fonctions de serveur
(ex. : Collecte de données)
 - Versatile (YOLO est invisible du client)
- Cadre de développement Recherche:
 - D'autres modèles de régression (architecture, type couches) ...
 - D'autres fonctions de coût ...
 - ... d'autres méthodes d'optimisation.
 - Intégrer des images partielles annotées (ex.: Flickr)



- Présentation de la méthodologie YOLO pour l'annotation d'images.
- Architecture Client/Serveur:
 - Simple : 30 lignes de code, 3 ports ouverts (0-MQ)
 - Puissante : Tout client/fournisseur qui supporte JSON.
 - Adaptable pour d'autres fonctions de serveur
(ex. : Collecte de données)
 - Versatile (YOLO est invisible du client)
- Cadre de développement Recherche:
 - D'autres modèles de régression (architecture, type couches) ...
 - D'autres fonctions de coût ...
 - ... d'autres méthodes d'optimisation.
 - Intégrer des images partielles annotées (ex.: Flickr)

- Finaliser un entraînement de Tiny YOLO des bases existantes.
- Raffiner les paramètres par rapport au projet.
- Embarquer le tout dans un prototype (Robot / Drone).

▶ Démo !