|  |
| --- |
| 8INF912 - Sujet spécial en Machine Learning |
| Food Recognition |
| Université du Québec à Chicoutimi |

|  |
| --- |
| Georges Cosson  11/06/2019 |

# Présentation du projet

Le but de ce projet est de reconnaitre les aliments d’une assiette à partir d’une photographie de ceux-ci. Il s’inscrit dans un cadre plus large visant à fournir un outil aux diabétiques leur permettant d’avoir une estimation des glucides de leur repas à partir d’une simple photographie depuis leur téléphone.

L’intégralité du code est disponible [ici](https://github.com/stressGC/Food-Detection-Dataset).

# Travail réalisé

## Découverte de la détection d'objets

### Découverte

* Recherche de datasets de nourriture et d'aliments
* Recherches et compréhension des différents algorithmes
* Familiarisation avec le dataset [Food-101](https://www.vision.ee.ethz.ch/datasets_extra/food-101/)

### Première expérience

* Mise en place d'une instance IBM Cloud Storage
* Upload d'une partie du dataset sur celle-ci
* Utilisation de l'outil Cloud Annotation Tool pour annoter ces images
* Entrainement d'un modèle SSD MobileNet sur une instance IBM Cloud Computing
* Récupération du modèle
* Installation et configuration d'une application ReactJS pour tester le résultat sur ma webcam

### Bilan

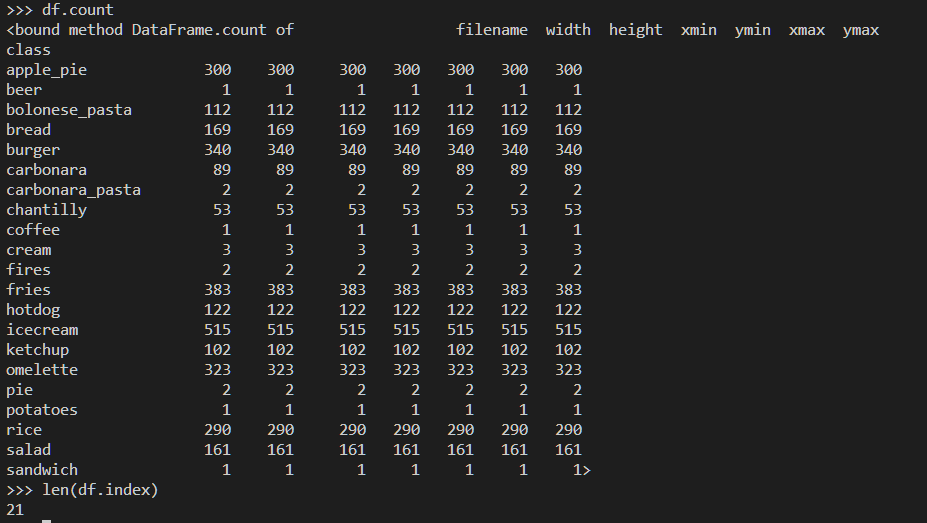
Le dataset était beaucoup trop petit pour avoir des résultats pertinents. De plus, le choix de pouvoir détecter les aliments à partir d'une vidéo (ou webcam) n'est pas pertinent, on n'a pas besoin de sacrifier la performance pour augmenter le temps de calcul.

## Création d'un dataset personnalisé

J'ai extrait environ 15 classes du dataset Food101, en choississant celles qui avaient beaucoup de chance de contenir d'autres classes dans leurs images, afin de gagner du temps de scrapping et de labelisation (ex: burger, ketchup et fries souvent ensemble).

De plus, j'ai scrappé Google Images pour obtenir des classes personnalisées comme "salad", "ketchup" ou "bread" à l'aide de l'outil de téléchargement [Bulk Image Downloader](http://www.talkapps.org/bulk-image-downloader).

J'ai donc obtenu environ 1900 images pour 19 classes, soit plus ou moins 100 images par classe. J'ai ensuite redimensionné celles-ci en format 250x250 pixels.



Annotation du dataset avec l'outil [LabelImg](https://github.com/tzutalin/labelImg) au format PascalVOC.

## Mise en place d'une VM AWS

Suite à de nombreux problèmes dans l'installation et configuration des outils de compilation de Tensorflow et de Python sous Windows, j'ai décidé de migrer sous Linux. J'ai donc mis en place une VM chez Amazon Web Services. VM de type g3.4xlarge, optimisée pour les calculs GPU (16 coeurs, 47 ECU et 122Gio de mémoire). Ensuite mise en place des dépendances et de Tensorflow.

## Formattage & Répartition des données

* Transfert du dataset vers l'instance AWS

scp -i <key\_path> -r <user>@<server> <local\_path>

* Convertion des annotations du format PascalVOC vers un format CSV

python voc\_to\_csv.py

* Mise en place d'un script de séparation des ensembles de test et d'entrainement

python labels\_test\_train\_split.py

* Compilation du dataset en un fichier tfrecord, à réaliser deux fois pour les subsets d'entrainement et de test.

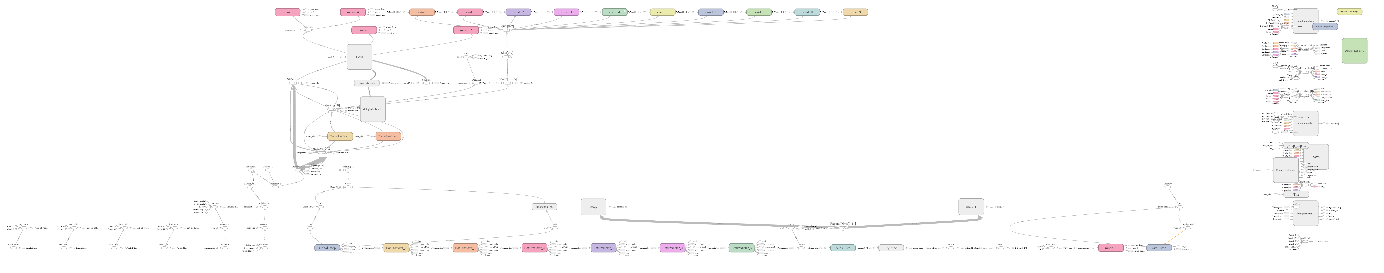
python generate\_tfrecord.py

* Création du fichier de mappage des classes

## Modèle

On utilise la méthode du transfer learning pour initialiser notre modèle à partir d'un modèle pré-entrainé sur des énormes quantités de données.

Nous avons le choix entre les modèles "légers" et plus "approximatifs", et des modèles plus lents et "précis". J'ai choisi la seconde catégorie suite à ma première expérience décrite plus haut. Le modèle est inception v2 entrainé sur le dataset Coco. On a fait le choix de reconnaitre le bon aliment en plus de temps de calcul.



### Installation

* On clone le projet à partir du GitHub de Tensorflow
* On l'intègre à notre projet et on configure le pipeline

### Entrainement

On lance l'entrainement

python ../object\_detection/model\_main.py \

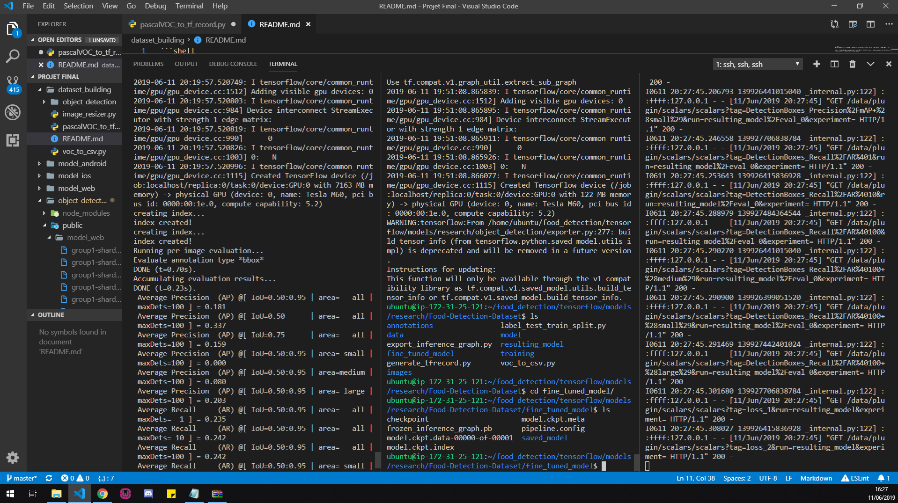
--pipeline\_config\_path=training/inception\_v2.config \

--model\_dir=resulting\_model/ \

--num\_train\_steps=50000 \

--sample\_1\_of\_n\_eval\_examples=1 \

--alsologtostderr



On supervise l'entrainement avec TensorBoard (évaluation jobs).

ssh -L 127.0.0.1:<remote\_port>:127.0.0.1:<local\_port> <user>@<server>

cd path/to/project

tensorboard log\_dir=.

Une image contenant capture d’écran

Description générée automatiquement

Enfin on sauvegarde le model en tant que fichier .pb.

python export\_inference\_graph.py \

--input\_type image\_tensor \

--pipeline\_config\_path training/inception\_v2.config \

--trained\_checkpoint\_prefix resulting\_model/model.ckpt-2406

--output\_directory ./fine\_tuned\_model

# Planification

* Script de conversion du modèle du format .pb à JSON pour l'utiliser sur le web en NodeJS (avec Tensorflow JS)
* Modifier l'application "démo" pour pouvoir tester sur des images et non sur la webcam le nouveau modèle.
* Réiterer la recherche du meilleur modèle sur lequel réentrainer
* Optimisation du modèle choisi
* Mise en place d'une API servant notre modèle
* Refactor l'architecture du projet pour respecter les bonnes pratiques
* Ajouter des classes à reconnaitre

# Bilan

J'ai rencontré beaucoup de difficultés à mettre en place un environnement de travail sous Windows, de plus la création d'un dataset est un processus très long et répétitif.

J'ai pris beaucoup de plaisir à mettre en place un dataset unique, et à le coupler à un modèle de Deep Learning. De plus, malgré beaucoup d'optimisations possibles, le modèle reconnait déjà ses premières images.

# Annexes

Une image contenant carte

Description générée automatiquement

Une image contenant capture d’écran

Description générée automatiquement

Figure 1: Capture d'écran de TensorBoard

Une image contenant capture d’écran

Description générée automatiquement

Figure 2: Graph d'Inception v2

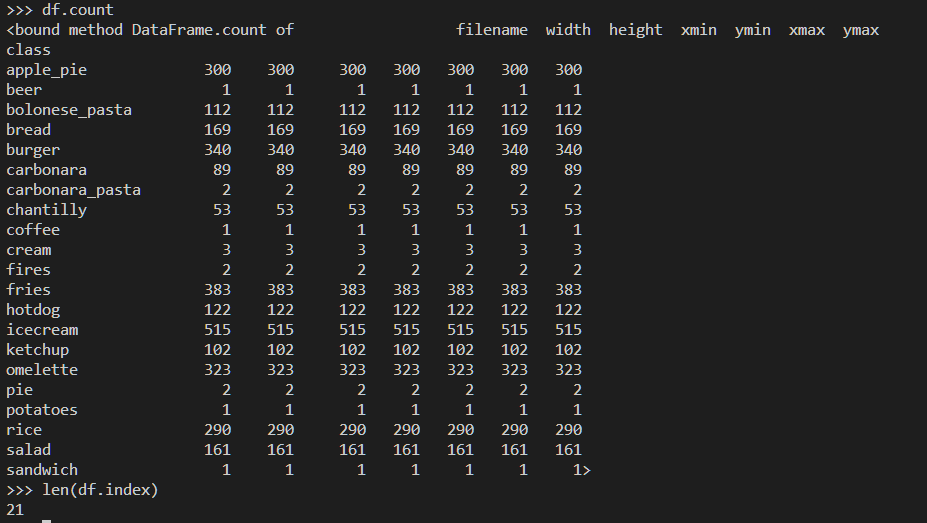


Figure 3: Capture d'écran des différentes classes

Une image contenant capture d’écran

Description générée automatiquement

Figure 4: Exemples de prédictions vs annotations

Une image contenant capture d’écran

Description générée automatiquement

Figure 5: Exemples de prédictions vs annotations

Une image contenant assis, ordinateur

Description générée automatiquement

Figure 6: différents terminaux servant à lancer et monitorer l'entrainement du modèle