Connexionisme

Deeplab

FELIX Jérémy
MOUKIN Kévin
REGALADE Davy

2018 - 2019

Sommaire

I. Objectifs et points du projet

Le but de ce projet était de comprendre le deep learning, à travers un sujet choisi ou imposé. Dans notre cas, cela s'est fait sur le DeepLab. Le DeepLab est une technique de segmentation d'images avec une reconnaissance d'objets dans un plan grâce à son apprentissage préalable par tensorflow.

Pour ce projet, nous disposons d'un jeu de données fournies par DeepLab, disponible sur leur dépôt Github.

Le jeu de données peut avoir deux axes très différents dans l'apprentissage :

- Entraîner le modèle à partir de zéro : liberté d'avoir un nombre illimité de classes d'objets (nombre d'étiquettes) pour la segmentation. Cela nécessite un temps de formation très long.
- Le modèle préformé : liberté d'avoir un nombre quelconque de classes d'objets pour la segmentation. Mettre à jour que les poids du classificateur avec l'apprentissage par transfert. Cela prendra beaucoup moins de temps pour l'entraînement que pour le scénario précédent.

II. Explications

a. TensorFlow

TensorFlow est un Framework Open Source de programmation pour le calcul numérique développé Google.

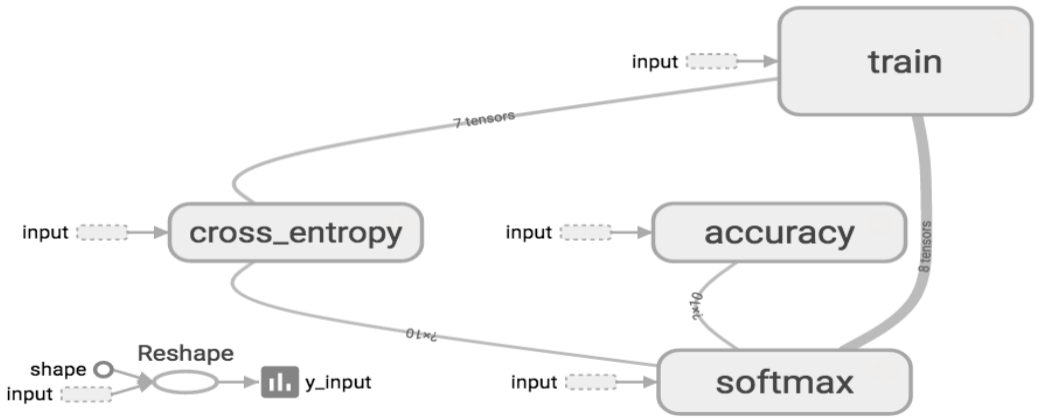
TensorFlow représente les calculs sous la forme d'un graphe d'exécution: les différents nœuds représentent les « *Operations* » à réaliser, et les liens représentent les « *Tensors* ». Une « *Operation* » peut aller d'une simple addition à une fonction complexe de différenciation matricielle.

Chaque « *Operation* » prend en entrée zéro, un ou plusieurs « *Tensor* », effectue un calcul, et retourne zéro, un ou plusieurs « *Tensor* ». Un exemple typique de « *Tensor* » est un batch d'images. Un batch d'images est représenté par un « *Tensor* » à 4 dimensions: taille du batch (nombre d'images dans le batch), hauteur, largeur et nombre de canaux de représentation (3 pour une image en couleurs représentée en RGB).

La création du graphe est automatiquement gérée par TensorFlow une fois les « *Tensors* » et « *Operation* » implémentés et instanciés. Cela permet une optimisation et parallélisation du code et de l'exécution lors du lancement.

TensorFlow possède de plus un support très vaste pour la création d'opérations spécifiques au Deep Learning, et il devient donc facile de construire un réseau de neurones et d'utiliser les opérations mathématiques couramment associées pour l'entraîner avec les bons optimiseurs.

Voici, un exemple de graphe d'exécution :



Exemple de graphe d'exécution 1

b. DeepLab

DeepLab est l'une des techniques les plus prometteuses pour **segmentation d'image sémantique** avec Deep Learning. La segmentation sémantique consiste à comprendre une image au niveau du pixel, puis à attribuer une étiquette à chaque pixel d'une image, de sorte que les pixels portant la même étiquette partagent certaines caractéristiques.

Qu'est-ce que la segmentation sémantique d'image ?

La **segmentation sémantique d'image** est une opération de traitement d'image qui a pour but de rassembler des pixels entre eux suivant des critères prédéfinis. Les pixels sont ainsi regroupés en région, qui constituent un pavage ou une partition de l'image. Il peut s'agir par exemple de séparer les objets du fond.

c. DCNN

Le **DCNN** est une architecture se rapprochant du **CNN** mais avec cette notion de multiples couches cachées faisant office de réservoir d'où son nom de DEEP(Profond).

Par ailleurs, cette architecture fonctionne de manière quasi similaire qu'un CNN car implicitement un CNN est un DCNN particulier n'ayant qu'une seule couche cachée.

Qu'est-ce qu'un CNN ?

Ici, nous n'allons pas expliciter cette architecture de réseau de neurones car beaucoup trop dense et complexe à expliquer.

Le CNN constitue en grande partie à une avancée dans le Deep Learning. De par cette technique d'apprentissage via l'utilisation d'un CNN, ils en sont même capables d'apprendre à trier des images par catégorie avec, dans certains cas, d'où le projet deeplab.

Afin de comprendre le fonctionnement d'un CNN, nous allons juste nous intéressés à son fonctionnement global de par ses différentes taches/actions auxquels il soumet son image, dans le but d'avoir des informations pertinentes en sortie, comme tout réseau de neurones, lui permettant de déterminer différentes classes de manière générale.

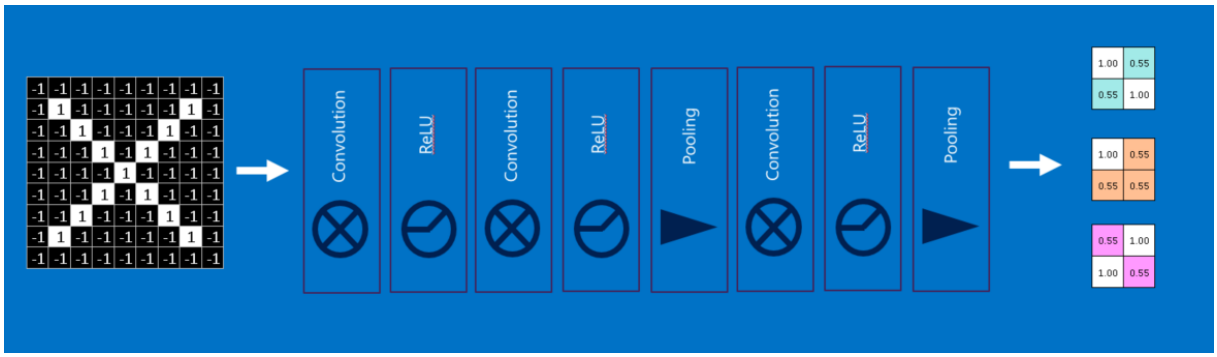
Le CNN compare les images fragment par fragments. Les fragments qu'il recherche sont appelés les caractéristiques. En trouvant des caractéristiques approximatives qui se ressemblent à peu près dans 2 images différentes, le CNN est bien meilleur à détecter des similitudes que par une comparaison entière image à image. Chaque caractéristique, qui rassemble les données communes des images, est représentée par une imagette en 2D, fonctionnant comme le principe des moments.

Le CNN, comporte différentes couches:

- **Convolution** :
- **Pooling** : cherche le maximum, du filtre et réduit l'image d'entrée par une taille inférieure de chaque bord dû à la convolution. On place dans le pixel central, kernel, la valeur maximale du filtre.
- **ReLu** : remplace les valeurs négatives par 0, afin d'avoir un intervalle positif pour les pixels de notre image et non des valeurs aberrantes.

Les CNN ont un autre aspect. En effet, les couches entièrement connectées prennent les images filtrées de haut niveau et les traduisent en votes. Ces couches entièrement connectées sont les principaux blocs de construction des réseaux de neurones traditionnels. Au lieu de traiter les entrées comme des tableaux de 2D, ils sont traités en tant que liste unique et tous traités de manière identique. Chaque valeur a son propre vote, ce qui déterminera la classe en sortie.

Exemple de traitement CNN :



Exemple de traitement du CNN

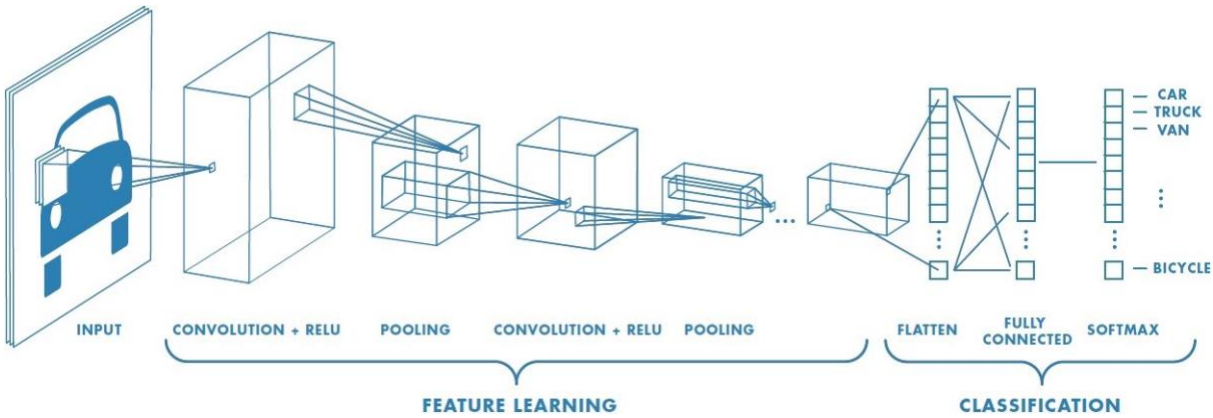
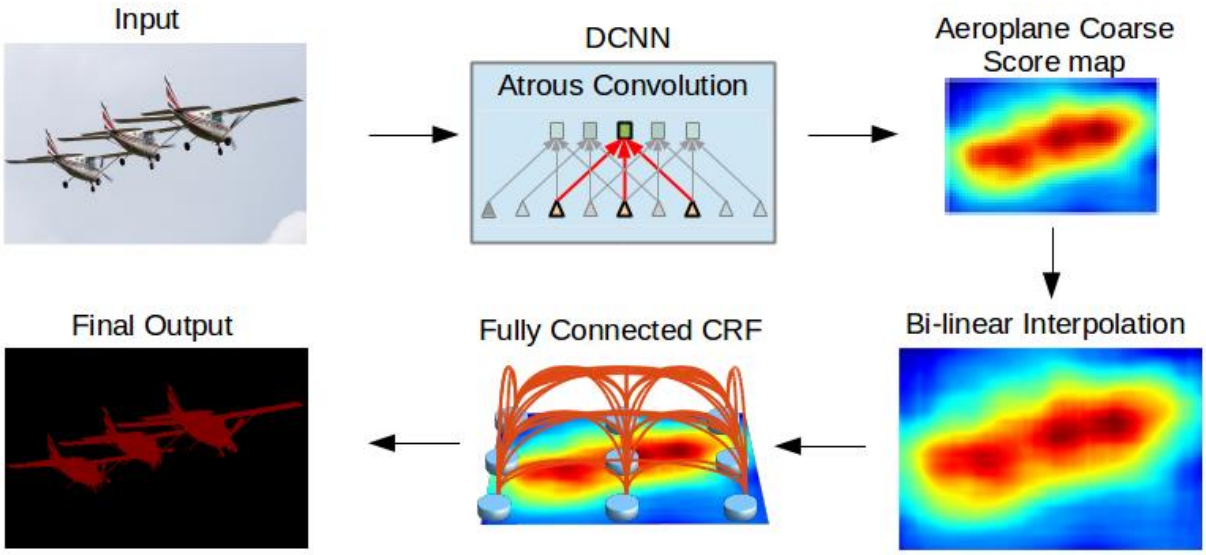


Schéma du CNN

C'est quoi le Deep Convolutional Neuronal Networks(DCNN)?

Passons, maintenant, au DCNN.

Comme nous avons pu le souligner précédemment, le Deep Convolutional Neuronal Network ou encore DCNN, est une architecture de réseaux de neurones qui a comme apprentissage, le deep learning. Le DCNN reprend le même principe que le CNN, sauf que dans son cas il comporte plusieurs couches cachées au lieu d'une seule.

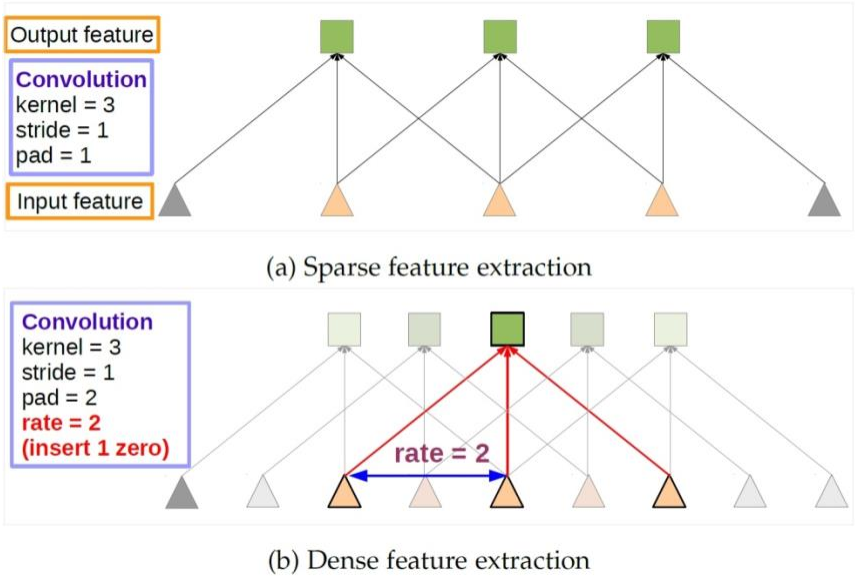


Trois contributions principales ont démontré expérimentalement qu'elles ont un grand mérite pratique pour la segmentation d'image avec le deep Learning.

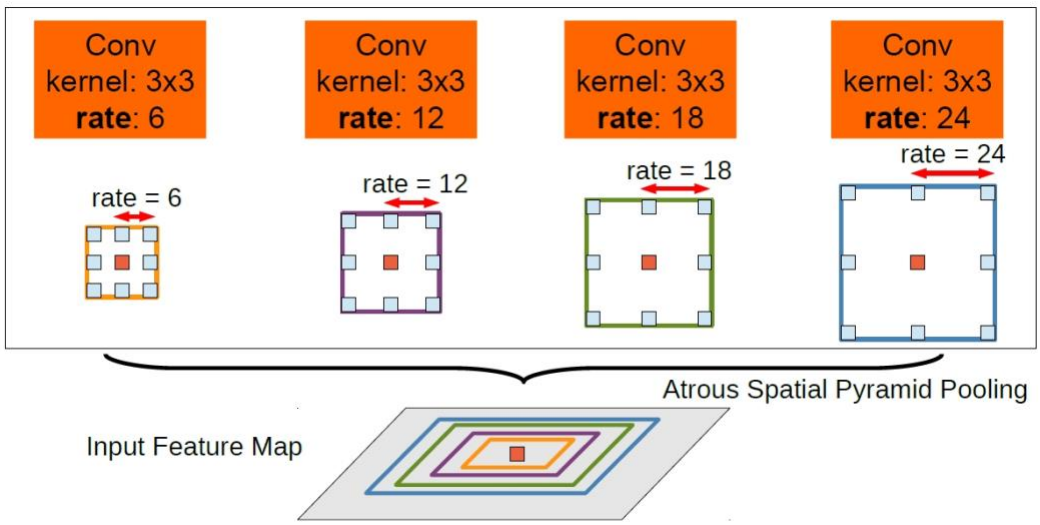
- La **convolution Atrous**, nous permet de contrôler explicitement la résolution dont les réponses des caractéristiques sont calculées dans les réseaux neuronaux convolutionnels profonds.

- La mise en commun de l'ASPP pour segmenter de manière robuste des objets à des échelles multiples. L'ASPP sonde une couche de caractéristiques convolutionnelles entrante avec des filtres à des fréquences d'échantillonnage multiples et des champs de vues efficaces, capturant ainsi des objets ainsi que le contexte de l'image à plusieurs échelles

L'amélioration de la localisation des limites des objets en combinant les méthodes des DCNNs et des modèles graphiques probabilistes. La combinaison couramment utilisée du max-pooling et du downsampling dans les DCNNs permet d'obtenir une invariance mais a un impact sur la précision de localisation. Nous surmontons ce problème, en combinant, les réponses à la couche DCNN finale avec un champ aléatoire conditionnel (CRF) entièrement connecté, ce qui est démontré à la fois qualitativement et quantitativement pour améliorer les performances de localisation. De plus, un traitement supplémentaire avec les champs aléatoires de Markov permet d'augmenter qualitativement les performances du réseau.



Atrous Convolution - DCNN



Classification des pixels - ASPP

Pour classer le pixel central (orange), Atrous Spatial Pyramid Pooling(ASPP) exploite des caractéristiques multiéchelles en utilisant plusieurs filtres parallèles avec des taux différents. Les vues de champ effectives sont affichées en différentes couleurs.

Pour la détermination des classes, l'idée est qu'à la sortie de chaque pixel après le traitement du DCNN, cela renvoie une valeur et une couleur propre à chaque classe. Là où, la question peut se poser, c'est clairement comme l'application fait pour déterminer le fond des classes et ainsi lui attribuer une certaine couleur.

L'une des pistes, est que le background est considéré comme étant une classe propre et non comme étant une classe non reconnue et donc on lui attribue une valeur et sa couleur. De plus,

je pense que cette attribution est due au mask présenté par défaut au RNA. En ce qui concerne, l'application mobile, nous ne sommes pas exactement dans ce cas car il ne donne des valeurs qu'aux différentes classes reconnues et ne change pas le reste, du moins ne superpose pas la couche de reconnaissance.

Exemple de Mask 1

Exemple, de jeu de couleurs pour les différentes classes :

B-ground	Aero plane	Bicycle	Bird	Boat	Bottle	Bus
Car	Cat	Chair	Cow	Dining-Table	Dog	Horse
Motorbike	Person	Potted-Plant	Sheep	Sofa	Train	TV/Monitor

Les réseaux neuronaux convolutionnels profonds(DCNN) formés sur un grand nombre d'images on fait évoluer la segmentation sémantique des images de façon significative. Des méthodes de Maximisation des Attentes pour la formation des modèles de segmentation d'images sémantiques dans ces environnements faiblement supervisés et semi-supervisés sont développé.

III. Installation de l'application

Pour ce projet, nous nous sommes basé sur une application déjà existante, utilisant le Tensorflow et provenant d'un projet existant de reconnaissances d'objets dans une image/vidéo, DeepLab sur Android. (*https://github.com/pillarpond/image-segmenter-android*)

Fonctionnement de l'application

Un objet « **TensorFlowInferenceInterface** » est créé avec le graphe de Tensorflow qui possède la base d'apprentissage, ce graphe est situé dans le dossier « Asset ».

L'appareil photo numérique du Smartphone capture les images en permanence, ces images sont transmis à une fonction de l'objet « **TensorFlowInferenceInterfac**e », puis, il commence à effectuer la reconnaissance des différents objets, si l'un des objets est identifié, alors un filtre est ajouté directement, en fonction de la classe de cet objet identifié, et une lui attribue une couleur.

Les différentes étapes du fonctionnement de l'application :

1. Chargement de la base pré-entrainée,
2. Sauvegarde de la base de données dans une instance de classe **TensorflowInferenceInterface**,
3. Création d'une variable graphe à partir de **TensorflowInferenceInterface** permettant de stocker la structure du graphe et ses collections.
4. Envoi des données vers Tensorflow et génération d'une matrice de données (sous format 1 D)
5. Application d'un mask (filtre de couleur) pour chaque pixel pertinent de la matrice,
6. Appel de la fonction permettant la reconnaissance,
7. Application du mask sur chaque pixel reconnu sur le bitmap (sur l'écran),
8. L'objet reconnu est ajouté à une liste.

Afin de l'utiliser sur Android, il va falloir installer quelques prérequis :

1. Installer Android Studio.
2. Installer Git sur votre machine
3. Configurer git dans Android studio
    - Sur Android Studio, aller dans File -> Settings -> Git
    - Mettre le chemin de git.exe existant dans votre Git au préalable installé,
    - Tester pour vérifier le path et Valider
    - Ensuite, cliquer sur VCS -> Check out from…. -> Git
    - Mettre l'url suivante : https://github.com/lebenef/image-segmenter-android
4. Git clone le projet.
5. Activer le mode développeur et le débogage de son téléphone.
6. Run l'application sur son téléphone.

IV. Modifications apportés dans le code

Pour l'instant, nous n'avons pas effectué de modifications majeures sur l'application. Nous pouvons juste afficher les labels (noms) de chaque classe reconnue.

Par ailleurs, nous avons réfléchi à des améliorations que l'on pourrait apporter à cette application, dans un premier temps, augmenter la reconnaissance puis par exemple l'apprentissage de nouvelles classes d'objets ou encore un historique des objets reconnus et leur occurrence durant un laps de temps. L'ajout d'un mode vidéo ou photo pour la reconnaissance, c'est-à-dire, qu'on lui passe une vidéo et l'application arrive à détecter les objets de la vidéo avec leur temps d'apparitions ? De même pour une image qu'on lui passe en entrée.

Si l'on veut pousser le concept un peu plus loin et partir dans une idée assez folle. On pourrait s'inspirer de « l'œil de dieu » de Fast and Furious 7, et faire en sorte que le deeplab détecte des comportements dangereux ou d'objets grâce aux caméras. Intégrer le sujet de thèse de M.Nagau pour la détection de certaines plantes ?

V. Conclusion

VI. Webographie

DeepLab

- https://ai.googleblog.com/2018/03/semantic-image-segmentation-with.html
- http://liangchiehchen.com/projects/DeepLab.html
- https://arxiv.org/pdf/1511.03339.pdf
- http://www.stat.ucla.edu/~xianjie.chen/paper/Chen14cvpr.pdf
- https://medium.freecodecamp.org/how-to-use-deeplab-in-tensorflow-for-object-segmentation-using-deep-learning-a5777290ab6b

CNN

- https://fr.wikipedia.org/wiki/R%C3%A9seau_neuronal_convolutif#cite_note-mcdns-6

- https://medium.com/@CharlesCrouspeyre/comment-les-r%C3%A9seaux-de-neurones-%C3%A0-convolution-fonctionnent-b288519dbcf8
- https://openclassrooms.com/fr/courses/4470531-classez-et-segmentez-des-donnees-visuelles/5082166-quest-ce-quun-reseau-de-neurones-convolutif-ou-cnn
- https://www.researchgate.net/figure/A-general-DCNN-architecture_fig1_318332170

TensorFlow

- https://blog.xebia.fr/2017/03/01/tensorflow-deep-learning-episode-1-introduction/