Data challenge

Land cover predictive modeling from satellite images

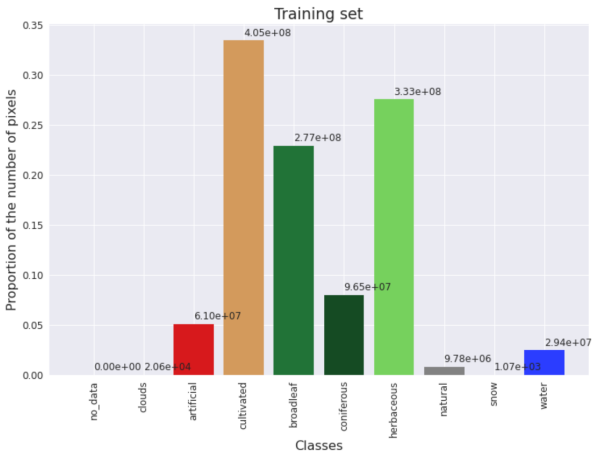
Mohammad Afandi

Objectif du challenge

Ce challenge porte sur un problème de modélisation prédictive de la couverture du sol à partir d'images satellites. L'objectif est de prédire, pour une image satellite donnée dans laquelle chaque pixel a été asssigné à une classe de couverture terrestre, la proportion de chaque classe de couverture terrestre dans l'image.

Données

Les données contiennent des images de 256×256 pixels multispectrales à 4 bandes (R-G-B-NIR). Chaque pixel recouvre une surface au sol de 10m^2.
Les 10 classes (avec l'index du label) présentes sont : no_ data (0), clouds (1), artificial surfaces and construction (2), cultivated areas (3), broadleaf tree cover (4), coniferous tree cover (5), herbaceous vegetation (6), natural material surfaces (7), permanent snow-covered surfaces (8), water bodies (9).
Deux points sont à noter : d'une part, on ne tient en fait pas compte des deux premières classes "no_ data" (0) et "clouds" (1). En effet, la classe 0 (correspondant à des annotations manquantes) n'est pas présente dans l'ensemble de données, tandis que la classe 1 (nuages) est rare et apporte très peu d'informations. D'autre part, il y a un fort déséquilibre entre les classes, comme illustré sur cette figure :



Nous disposons donc de deux ensembles de données :

- Les données d'entraînement : 18941 fichiers TIFF 16 bits pour les images et autant de fichiers TIFF 8 bits pour les masques (labels) associés ;
- Les données de test : 5043 fichiers images TIFF 16 bits.

Vu le format particulier des images, j'ai dû utiliser le module Python `TiffFile` pour charger les données issues de `Train` d'une part et de `Test` d'autre part. J'ai ensuite transformé les images en tableaux numpy. Pour l'entraînement, j'ai brassé les données issues de `Train` avant de créer un ensemble d'apprentissage et un ensemble de validation (répartition 90%/10%).
Enfin, j'ai normalisé les valeurs des pixels des images de chaque ensemble de données entre 0 et 1 (`entraînement`, `validation` et `test`).

Choix du modèle

Ayant la chance de disposer d'une carte graphique Nvidia récente (compatible avec CUDA), j'ai décidé d'opter pour un modèle qui est très utilisé dans le domaine du traitement d'image, plus particulièrement en segmentation d'images. Ne voulant pas m'orienter vers des algorithmes plus exotiques et compliqués, je me suis inspiré du benchmark fourni sur le site du challenge pour implémenter un réseau U-Net (Ronneberger et al 2015) sous Tensorflow. Le modèle est entraîné à prédire les probabilités d'appartenance à une classe pour chaque pixel d'une image. Après avoir été entraîné sur les masques de segmentation, le réseau prédit pour chaque pixel la classe la plus probable.

Commentaires et résultats

La première grosse difficulté a été de charger les données correctement dans le réseau. Il s'agissait de bien faire attention aux dimensions des vecteurs transformés et utilisés.

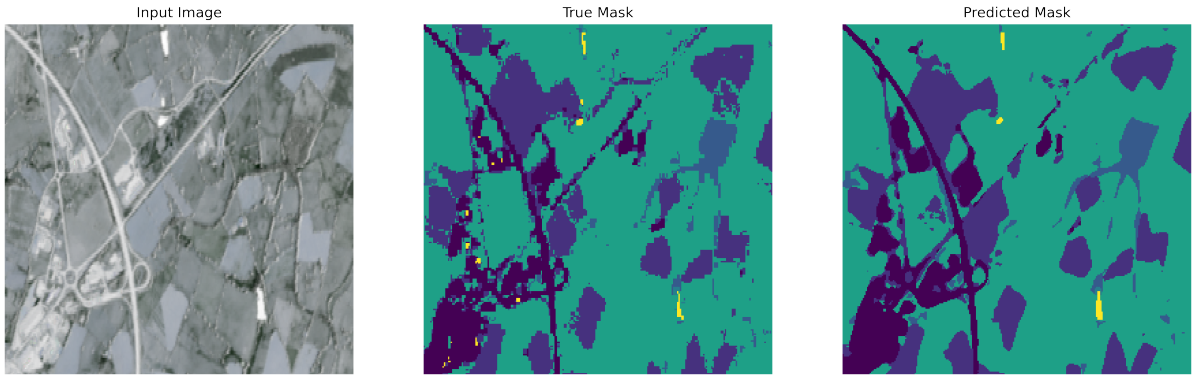
Une autre difficulté à laquelle j'ai été confronté est inhérente à la nature du modèle que j'ai choisi. En effet, comme le temps d'entraînement est long (5 minutes pour une époque avec $batch\_size = 8$, le maximum permis par ma carte graphique disposant de 8Go de VRAM), il est fastidieux d'optimiser les divers paramètres.

J'ai opté pour une fonction de perte simple (entropie croisée), mais peut-être qu'une fonction différente aurait été plus adaptée au problème de classification avec classes déséquilibrées. Un exemple de telle fonction est la dice loss. L'inconvénient est qu'elle rend plus compliqués les calculs de rétropropagation du gradient lors de la phase d'entraînement.

En inspectant le code fourni dans le benchmark du challenge, je me suis aperçu que le déséquilibre des classes n'était en fait pas pris en compte par le modèle fourni. Après avoir corrigé cette erreur (non sans mal), mon modèle a énormément surappris et par conséquent a eu du mal à généraliser. J'ai essayé de modifier les paramètres, réduire le nombre de couches, changer les fonctions d'activation, en vain. C'est pourquoi mon modèle final ne prend pas en compte le poids des classes. J'aurais peut-être dû procéder autrement, par exemple au moment du traitement des données en ré-échantillonnant ou alors en créant des échantillons synthétiques afin d'équilibrer la répartition des classes.

Une autre piste d'amélioration serait peut-être d'utiliser des méthodes plus fines pour entraîner le modèle sur chacune des différentes classes. En effet, chaque classe ayant des propriétés différentes, il serait peut-être judicieux d'adapter les techniques de classification selon les cas.

J'ai finalement obtenu un score de 0.0481 sur le classement public (`epochs=95`, `batch_size=8`).
Un exemple de prédiction :



Pour conclure, même s'il est toujours possible d'améliorer le modèle, je suis satisfait de mon challenge. Il aura été l'occasion pour moi de mettre en oeuvre des techniques vues en cours et de découvrir de nouvelles fonctions Python.