Apprentissage et reconnaissance (GIF-4101 / GIF-7005)
Département de génie électrique et de génie informatique
Automne 2018

UNIVERSITÉ LAVAL

Devoir 4

Instructions :

- Formation des équipes :
  - GIF-4101 : le devoir est réalisé en équipe de deux à trois étudiants
  - GIF-7005 : le devoir est réalisé individuellement
- Programmation :
  - Produisez vos solutions dans le fichiers fournis, en respectant les instructions et n'utilisant pas d'autres modules que ceux qui y sont importés
  - Une semence du générateur de nombres aléatoires (*random seed*) correspondant à votre numéro d'équipe (de boîte de dépôt) doit être utilisée
  - Assurez-vous d'utiliser la version 0.4.1 de PyTorch
- Remise :
  - La remise du rapport et du code source se fait dans monPortail
  - Le remise doit être effectuée au plus tard le mercredi 5 décembre à 9h30
- Pondération :
  - Ce devoir compte pour 5% de la note finale

---

1. **Entraînement de réseaux de neurones à convolution (10pt)**

   Dans cette question, nous vous demandons d'entraîner un réseau de neurones à convolutions sur le jeu de données *Volcanoes on Venus*[1]. Une version abrégée du jeu de données vous est données pour le devoir sur le site du cours[2] (attention : 120 Mo). Nous fournissons également des fonctions utilitaires nécessaires pour le devoir dans le fichier `d4utils.py`[3], dont pour charger le jeu de données. Utilisez la librairie PyTorch pour toute implémentation de réseaux de neurones. La figure 1 présente l'architecture de base que vous devez traduire en code PyTorch.

   Pour cette question, partez du code `d4q1.py`[4] fourni et le modifiez selon les directives suivantes :

   - Écrivez le code permettant de créer les couches du réseau à convolution `VolcanoesNet`, en utilisant les couches de convolution (`VolcanoesConv`), de normalisation (`BatchNorm2D`) et linéaire (`Linear`), selon l'architecture présentées à la figure 1.
   - Écrivez les lignes de code manquantes pour l'inférence du réseau dans la méthode `forward` du réseau `VolcanoesNet`.
   - Écrivez les lignes de code manquantes pour la préparation de l'entraînement et celles à l'intérieur de la boucle d'entraînement.

1. `https://www.kaggle.com/fmena14/volcanoesvenus/`
2. `http://vision.gel.ulaval.ca/~cgagne/enseignement/apprentissage/A2018/travaux/volcanoes.zip`
3. `http://vision.gel.ulaval.ca/~cgagne/enseignement/apprentissage/A2018/travaux/d4utils.py`
4. `http://vision.gel.ulaval.ca/~cgagne/enseignement/apprentissage/A2018/travaux/d4q1.py`

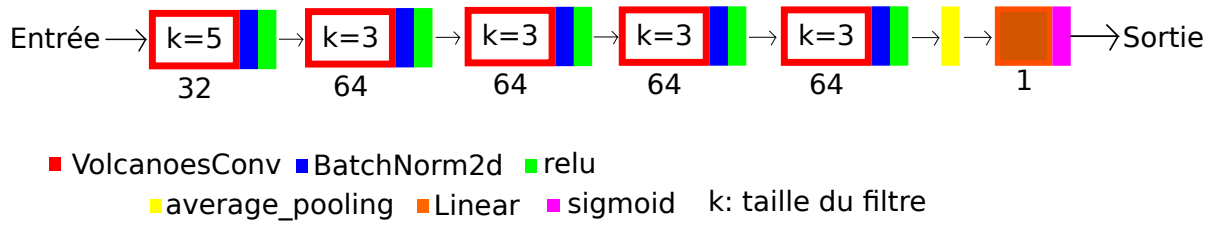

FIGURE 1 – Architecture du réseau de neurones à convolution utilisé pour la question 1. Le valeur de k représente la taille des filtres (ex. $k = 3$ signifie un filtre de 3×3) alors que le nombre en bas des boîtes représente le nombre d'éléments de la couche associée.

— Entraînez le réseau avec le code complété. Vous pouvez tester des hyper-paramètres différents que ceux conseillés, ceux-ci n'ont pas été ajustés précisément pour le problème.
— Rapportez votre score final et la matrice de confusion sur le jeu de test dans le rapport et remettez le modèle enregistré dans votre fichier de remise.

L'entraînement de votre réseau ne devrait pas prendre plus de 10 minutes sur un CPU et vous devriez obtenir un score de classement d'au moins 80 % après 10 époques.

Portez attention à ce que le modèle enregistré dans votre dossier puisse bien être chargé sans erreur par votre classe définie dans `d4q1.py` !

Bonus : Modifiez le réseau pour obtenir le meilleur score possible. Toutes les fonctionnalité disponibles dans PyTorch sont permises. L'équipe produisant le modèle entraîné avec la meilleure performance se méritera un bonus de 1 point sur la note du devoir (avec saturation à 20/20).

2. **Transfert de représentation (10pt)**

Dans cette question, nous vous demandons d'étudier un cas de transfert de représentation. Pour ce faire, nous utiliserons le réseau *ResNet-18* préalablement entraîné sur le jeu d'images naturelles *ImageNet*. On vous demande de modifier ce réseau en remplaçant seulement la tête du réseau pour l'adapter au nouveau jeu de données à 16 classes *Lego Brick* [5], il devrait vous être possible d'atteindre de très bonnes performance en seulement une époque d'entraînement. Le jeu de données est disponible sur le site Web du cours [6] (attention : 190 Mo).

Pour cette question, partez du code `d4q2.py` [7] fourni et le modifiez selon les directives suivantes :

— Changez la dernière couche pleinement connectée du réseau de neurone (couche `fc`) afin de l'adapter au nombre de classes du jeu de données (16). De plus, gelez les autres couches du réseau *ResNet-18* se trouvant avant la nouvelle couche pleinement connectée de sortie.
— Écrivez la ligne de code nécessaire à l'inférence du réseau dans la méthode `forward`.
— Écrivez les lignes de code manquantes pour la préparation de l'entraînement et celles à l'intérieur de la boucle d'entraînement.
— Entraînez le réseau en exécutant le code **sans** pré-entraînement. Le réseau devrait obtenir un taux de classement allant de 6 % à 40 % en moins de 30 minutes. Rapportez la performance dans votre rapport et remettez le modèle enregistré dans votre dossier.
— Entraînez le réseau en exécutant le code **avec** pré-entraînement. Le réseau devrait obtenir un taux de classement d'à peu près 84 % en moins de 15 minutes. Rapportez la performance

5. `https://www.kaggle.com/joosthazelzet/lego-brick-images`
6. `http://vision.gel.ulaval.ca/~cgagne/enseignement/apprentissage/A2018/travaux/lego.zip`
7. `http://vision.gel.ulaval.ca/~cgagne/enseignement/apprentissage/A2018/travaux/d4q2.py`

dans votre rapport et remettez le modèle enregistré dans votre dossier. Remarquez comment l'utilisation d'un réseau pré-entraîné a permis d'obtenir une performance beaucoup plus élevé en deux fois moins de temps !

Comme à la question précédente, portez attention à ce que le modèle soit enregistré dans votre dossier puisse bien être chargé sans erreur par votre classe définie dans `d4q2.py` !

14/11/2018
CG & LER