Implémentation: Learning to Trace

Mikhaïl Bessa

April 20, 2020

1 Introduction

Le papier [1] décrit une méthode d'extraction automatique des lignes principales d'une image, à partir de 2 réseaux de neurones convolutifs: un réseau générateur G qui extrait les lignes brutes et un réseau restaurateur R qui les affine. Ils sont entraînés ensemble, sur le dataset BIPED qui est constitué de photos et de leurs contours associés. La fonction de perte jointe est constituée de 3 termes:

$$L_{joint}(x, y) = L_{W1}(G(x), y) + \alpha L_{W1}(R(G(x), y) + \beta L_1(R(y^*), y)$$

x est la photo d'origine, y les contours associés, y^* les contours détériorés par une procédure décrite dans [1], L_1 la perte L1 et L_{w1} la perte définie dans [1] (perte L1 qui met plus de poids sur les contours noirs que sur le fond blanc)

2 Implémentation

J'ai implémenté la méthode en pytorch.

2.1 data.py

Ce fichier contient la classe CustomDataset qui gère et le chargement et le preprocessing des images. La procédure de détérioration est dans la méthode statique *CustomDataset.deteriorate(img)*. Le dataset dézipé est censé être placé dans Inoue/.
cloud.png est une texture nuageuse utilisée dans la procédure de détérioration des contours.

2.2 models.py

Ce fichier contient la définition des réseaux `Generator`, `Restorer` et l'importation de `ResNet50` pré-entrainé.

2.3 losses.py

Ce fichier contient la définition de la perte L_{WL1} définie dans le papier.

2.4 train.py

Ce fichier contient l'instanciation des réseaux, de l'optimiseur, du logger et du CustomDataset, la boucle d'entrainement et la sauvegarde de checkpoints. C'est le seul exécutable avec comme arg optionnel `--v` pour désactiver la visualisation visdom.

2.5 utils.py

Ce fichier contient les classes/fonctions associées au logging des métriques et à la visualisation d'échantillons pendant l'entraînement. La classe `Logger` enveloppe tout. J'ai utilisé la librairie visdom afin de tracer un graphe $mean_{loss}(epoch)$ pour les 4 fonctions de perte (L_{joint}, L_{base}, L_{res}, L_{aux}) et des échantillons (x, y, y_{pred}, $R(y^*)$). Pour chaque batch, les pertes moyennes sont affichées dans la console.

2.6 cfg/config.json

Ce fichier de configuration contient les hyperparamètres de l'entrainement (paramètres de la fonction de perte, de l'optimisation, dossier qui contient les sauvegardes, nombre d'epochs..)

3 Choix et améliorations possibles

- Le dataset BIPED contient des contours blancs sur fond noir. Pour garder la fonction de perte décrite dans le papier, je les ai inversées.
- Pour la procédure de remplissage des trous, j'ai trouvé une texture nuageuse qui donnait un rendu semblable.
- Si l'option est disponible, le GPU sera utilisé.
- Il est certainement possible d'optimiser le pré-traitement des images du jeu de données, et notamment la procédure de détérioration. Pour faire simple j'ai converti les images PIL de contours en arrays numpy pour les détériorer.
- Il faudrait ajouter un monitoring du temps écoulé et de la mémoire utilisée.
- Il faudrait aussi récupérer du code existant des méthodes d'extraction de contours auxquelles ce papier se compare (LPCB, Canny (opencv), Pix2pixHD)

References

[1] N. Inoue, D. Ito, N. Xu, J. Yang, B. Price, and T. Yamasaki. Learning to trace: Expressive line drawing generation from photographs. *Computer Graphics Forum*, 38(7):69–80, 2019.