DrunkSum

Projet Traitement d'image

Joaquim Perez & Axel Roy
03.05.2017
HE-Arc - Ingénierie

But du projet

Compter les billets de consommations en fin de soirée est parfois difficile, ceci en raison de la fatigue ou de l'alcool. Nous somme conscients de ce phénomène, et nous vous proposons une solution révolutionnaire:

"Somme en état d'ébriété", ou plus simplement "DrunkSum".

Le but est donc de fournir une application qui permet d'analyser l'image de la caméra d'un smartphone, d'en extraire les chiffres et de proposer à l'utilisateur d'additionner le chiffre reconnu au total de l'application.

Plateforme cible

Etant donné que nous avons comme public cible des utilisateurs en soirée ou sur terrasse, nous avons opté pour un déploiement sur smartphone, ce qui limite le choix de bibliothèques de reconnaissance de caractères.

Utilisation

Ce chapitre décrit l'installation et l'utilisation de l'application.

Installation

Lancez l'application DrunkSum depuis le smartphone de test fourni par l'école, ou installer l'apk fourni dans les délivrables.

Lien vers l'APK signé de l'application :
https://drive.google.com/file/d/0B9wmO5HutHghaS16VGY1RnpwZnc/view?usp=sharing

Lien d’aide pour installer un APK :
http://www.frandroid.com/comment-faire/tutoriaux/trucs-et-astuces/184151_comment-installer-un-fichier-apk-sur-son-terminal-android

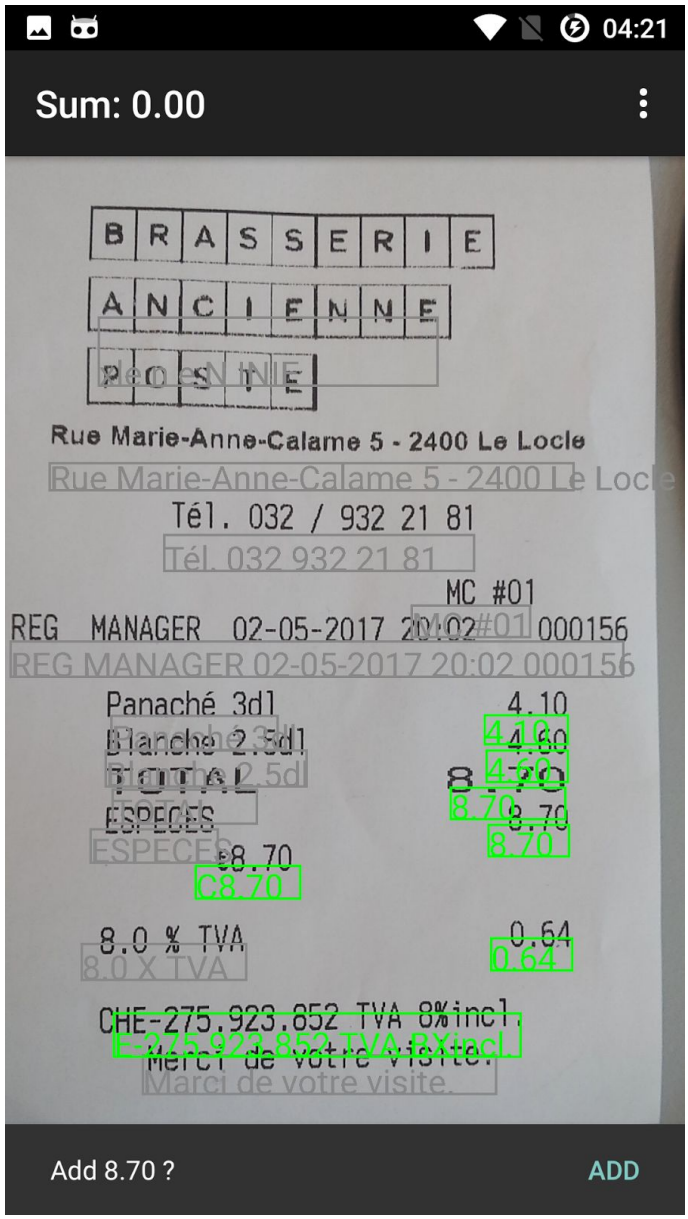
Lors du premier lancement après l’installation de l’APK, le smartphone va vous demander d’installer Google OCR reader. Ceci est nécessaire car c’est la bibliothèque de reconnaissance de caractère utilisée pour ce projet.

Utilisation

Une fois l'application lancée, vous devriez voir le rendu de la caméra de la face arrière. Si vous donnez à scanner un ticket de caisse, vous devriez voir deux couleurs de champs :

- Grisé pour les textes reconnus, mais non reconnus comme potentiellement des nombres.
- Vert pour les textes reconnus et dont on peut extraire les nombres, et donc qui sont utilisables par l'utilisateur pour sommer. Ces zones doivent contenir au moins un chiffre, un point ou une virgule suivi d'un point.

Voici un exemple, ici l'utilisateur vient de cliquer sur le total, 8.70 :



Une fois le focus ajusté, les zones vertes apparaissent et on peut cliquer dessus afin de faire apparaître la barre du bas. Vous pouvez dès lors vérifier le montant, et l'ajouter via le bouton add si le montant affiché est correct.

Le total se met alors à jour en haut, et vous pouvez scanner le ticket suivant.

Une fois tous les tickets scannés, l'application offre un menu qui propose deux options :

- Divide : Une boîte de dialogue demande un nombre par lequel diviser la somme. Le résultat est alors affiché.
- Reinitialize : La somme est remise à 0.

Deux problèmes majeurs subsistent :

- Le focus peut être pénible dans le cadre de l'application. Celui-ci peine à s'ajuster alors qu'il est en mode autofocus. La qualité de l'autofocus varie fortement entre les appareils, les appareils récents fonctionnent bien mieux.
- La vitesse de rafraîchissement est très rapide. Nous avons défini la fréquence de rafraîchissement à 2fps afin de laisser le temps à l'utilisateur de sélectionner la zone qui l'intéresse. Malheureusement, si l'appareil ne peut pas gérer ce framerate, il le défini à 30fps, ce qui est très rapide.

Reconnaissance

Il existe plusieurs méthodes de reconnaissance de texte, les plus modernes sont basées sur des réseaux de neurones. Ceux ci ont le gros avantage de prendre en entraînement des images de caractères auxquelles sont attribués des lettres, et permettent de configurer les paramétrages et les liens entre les neurones afin de s'entraîner de cas réels.

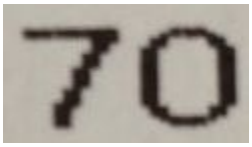
La taux de reconnaissance pour cette méthode est généralement meilleur que dans le cas d'une analyse via le procédé classique de traitement d'image qui est le suivant :

1) Pré processing : Traitement de l'image afin d'obtenir les caractéristiques les plus pertinente pour la reconnaissance : Binarisation, deskew, etc.
2) Segmentation : Récupération de caractéristiques selon le positionnement des pixels (Vecteurs en général), mais aussi méthodes métriques, liés à des méthodes statistiques fixes (p.e. chaînes de Markov).
3) Reconnaissance et caractérisation selon les champs de vecteurs puis attribution via des méthodes statistiques.
4) Post processing via des dictionnaires (pas dans notre cas)
5) Sortie de texte : Fichier texte, ou dans notre cas direct dans le flux de l'appli sous forme de chaîne de caractère.

Tesseract est basé sur cette méthode. On ne peut pas déterminer si c'est le cas de la bibliothèque utilisée, mais on peut tout de même préciser que les problèmes suivants sont récurrents.

- Problèmes pour différencier les S des 5 ainsi qu'entre les O et les 0.

- Problèmes de segmentation lorsque l'impression a produit des résultats crénelés.



Etant donné que nous n'avons pas de contrôle direct sur la bibliothèque de reconnaissance, nous avons effectué des manipulation post-reconnaissance afin d'améliorer l'expérience utilisateur.

Développement

Nous avons commencé le développement via des prototypes simplistes afin de tester des bibliothèques sur plateforme Windows via Python.

Pytesseract[1] est un wrapper Python qui permet d'utiliser Tesseract[2], la bibliothèque Open-Source la plus utilisée et la plus maintenue. Elle est passée dans sa version 4.0a à une reconnaissance via des réseaux de neurones, mais le wrapper ne permet pas de l'utiliser, et elle est encore en version Alpha. Tesseract a fourni des résultats très médiocres sur des images de test fixes, disponibles à l'adresse :
https://github.com/axelroy/DrunkSum/tree/master/samples

Etant donné que notre but était la plateforme android, nous nous sommes ainsi renseignés sur les bibliothèques du système d'exploitation. Google fournit gratuitement l'API Vision, dont la documentation se trouve à l'adresse :
https://developers.google.com/vision/text-overview

Nous avons ainsi suivi le tutoriel à l'adresse :
https://codelabs.developers.google.com/codelabs/mobile-vision-ocr/#0

Celui-ci est correctement tenu à jour, et a permis d'arriver à nos fins sans trop de surprises.

Détection des nombres

Depuis l'application de base, nous avons ajouté un système de reconnaissance des nombres basé sur une expression régulière. Les lignes reconnues comme comportant des nombres sont affichées en vert.
Pour contrer les problèmes de reconnaissances entre le 0 et le O ainsi qu'entre les S et les 5 décrits dans le chapitre reconnaissance, nous avons adapté ces expressions régulières afin d'autoriser ces caractères en milieu de phrase. Il arrive aussi que des espaces soient détectés, ceux-ci sont alors supprimés avant que les éléments soient analysés.

L'expression régulière finale est : `(\d|O|S)+(\.|,)(\d|O|S){2}`. Elle est divisés en trois groupes principaux :

- `(\d|O|S)+` un chiffre [0-9], un O ou un S, 1 ou plusieurs fois

[1] https://github.com/madmaze/pytesseract
[2] https://github.com/tesseract-ocr/tesseract

- `(\.|,)` un point ou une virgule
- `(\d|O|S){2}` un chiffre, un O ou un S, 2 fois

Les problèmes d'autofocus sont connus, et une simple implémentation d'un autofocus on tap est très compliquée. Nous avons essayé tous les modes de focus possibles pour la caméra[3], mais aucun n'apporte d'amélioration notable. Il semblait que de fixer le focus et d'effectuer la mise à bonne distance manuel était une bonne idée, mais la distance donnée par ce mode est fixe, et elle est bien trop éloignée de notre ticket (environ 40cm). Ces problèmes sont dus au matériel qui peut varier entre chaque smartphone, et il faut réécrire un driver[4] pour le périphérique afin de pouvoir gérer correctement le focus. Ceci n'est pas faisable dans le cadre de notre application.

Références

Documentation de la bibliothèque OCR utilisées :

https://developers.google.com/android/reference/com/google/android/gms/vision/CameraSource
https://developers.google.com/vision/text-overview

Références des problèmes de focus :
http://stackoverflow.com/questions/15623944/how-to-autofocus-android-camera-automatically
http://stackoverflow.com/questions/14580281/turn-off-auto-focus-in-android
https://github.com/mapillary/mapillary_issues/issues/159

[3] https://developer.android.com/reference/android/hardware/Camera.Parameters.html
[4] http://stackoverflow.com/a/24136653