**Solutions d’analyse d’image pour Android**

**Websocket API :**

Une première possibilité est de déporter tous les calculs sur un serveur « puissant » qui sera plus à même de faire l’analyse de l’image rapidement. On utilise un websocket pour transférer les images du client vers le serveur suffisamment rapidement.

Le client fait appel à l’API Camera et récupère les preview frames (des tableaux de byte) qui sont envoyées directement au serveur. Le serveur construit alors une matrice OpenCV avec ce tableau de nombres et convertit l’image du format YUV420SP (NV21) au format BGR pour l’analyse de l’image. L’analyse de l’image est ensuite effectuée et renvoyée au client qui peut donc réagir et éventuellement afficher une nouvelle publicité.

Le code source du côté serveur est disponible ici : <https://pastebin.com/raw/S4vFQUd2>

Le code source du côté client est disponible ici : <https://github.com/TravelToTheTop/CameraAPI>

Dépendances :

* Côté client :
  + Java-Websocket (<https://github.com/TooTallNate/Java-WebSocket>) : On peut l’ajouter via les fichiers Gradle sur Android Studio
* Côté serveur
  + Websocketpp (<https://github.com/zaphoyd/websocketpp>)
    - Boost 1.69.0 (<https://www.boost.org/users/history/version_1_69_0.html>) 🡪 compiler avec Visual Studio 2017 : lancer le cmd de visual studio 2017, faire la commande cd sur le dossier racine boost puis faire les deux commandes suivantes : « bootstrap » puis « .\b2 » (la compilation prend du temps)
  + OpenCV open model zoo (<https://github.com/opencv/open_model_zoo/tree/master/demos/interactive_face_detection_demo>)
    - CMake >= 2.8
    - OpenVino toolkit (<https://software.intel.com/en-us/openvino-toolkit>)
    - Après avoir compilé les démos, lancer le fichier sln (Visual Studio), choisir le projet interactive\_face\_detection\_demo et ajouter les dépendances de boost et websocketpp dans les propriétés du projet (se référer aux projets de démonstration de websocketpp)

Spécifications côté serveur :

* GPU dédié recommandé
* Testé sur Windows 10 avec 8go de ram
* Windows 10 ou Ubuntu 16.04 LTS 64-bit / CentOS 7.4 64-bit

Optimisations à effectuer :

* Passage de camera API à camera2 API côté client
* Créer de la redondance de données dans le tableau de nombres représentant l’image, grâce aux similitudes entre deux images consécutives, et compresser les données
* Essayer les différents formats d’image disponibles avec l’API camera2 et constater s’il y a des différences en taille d’image / vitesse de récupération des images

HyperLandmark (facial landmarks) + Tensorflow (Age/Sexe) :

Une deuxième solution est de faire l’analyse exclusivement en local : le projet HyperLandmark fait la détection faciale et détermine 106 points du visage. L’application Tensorflow détermine l’âge et le sexe d’une personne en fonction d’un bitmap de son visage. On peut combiner les deux applications (voir to-do ci-dessous).

Dépendances :

* HyperLandmark (<https://github.com/zeusees/HyperLandmark/tree/master/Prj-Android>) 🡪 Pour que l’analyse fonctionne il faut que les preview frame soient orientées dans un sens particulier. Pour changer l’orientation de l’image et trouver la bonne orientation (qui dépend de l’appareil android) on peut utiliser la fonction suivante pour faire des rotations de 90 degrés : <https://pastebin.com/raw/Y7KXPYj3>. Il est important de noter qu’on ne peut pas configurer l’orientation des preview frames avec l’API de la caméra.
* Tensorflow Age/Sexe (<https://github.com/KhAngelina/AgeGenderClassifierAppTF>) 🡪 Il faut utiliser les classes AgeGenderClassifier, AgeGenderTfMobileClassifier ou TfLiteImageClassifier et appeler la méthode classifyFrame. Il y a en fait trois classes différentes, avec des rapports précision/vitesse différents.

Benchmarks :

* Temps d’exécution pour déterminer âge et sexe d’un visage avec les 3 méthodes différentes proposées dans l’application Tensorflow (les bornes inférieures correspondent aux méthodes les plus rapides, les bornes supérieures aux moins rapides) :
  + 35-100 ms sur Snapdragon 855 (OnePlus 7T)
  + 350-900 ms sur rk3368 (la tablette)
  + 61-270 ms sur Snapdragon 820 (Galaxy Tab S3)
  + Tests à effectuer sur Snapdragon 810 (Nexus 6P), mais de souvenir c’était entre 100 et 200 ms avec les méthodes les plus rapides
* HyperLandmark affiche une détection fluide sur Snapdragon 855 et 820. Le Snapdragon 810 doit être encore testé. La détection affichée sur rk3368 est beaucoup plus saccadée mais le temps de réponse m’a l’air suffisant.

To-do :

* Merge les deux projets : convertir le byte array NV21 de HyperLandmark en bitmap, déterminer quelle est la personne la plus proche de l’écran qui regarde l’écran puis découper le bitmap pour avoir seulement le visage de la personne. Avec ce bitmap déterminer l’âge et le sexe de la personne avec Tensorflow.
* Déterminer si les personnes regardent l’écran en fonction des points du visage (HyperLandmark)

En ce qui concerne la synchronisation des vidéos pour les appareils, on pourrait imaginer un système comme ceci :

