UNIVERSITÉ
DE MONTPELLIER

Compte rendu 3 - Mise en place de la base de code

BELDJILALI Maxime, CHATEAUNEUF Arthur

Obscurator

Mise en place de la base de code

Lors de cette première semaine d'implémentation, nous avons principalement mis en place notre base de code C++ permettant d'obscurcir des images ou des bases de données.

Le projet utilise actuellement la librairie STB [1] pour le chargement d'images ainsi que ColorM [2] afin d'avoir une base de conversion pour un grand panel d'espaces couleurs. Le code est réparti entre les sources communes d'implémentations de classes (type Image) et les diverses applications que nous voulons créées.

Nous avons ainsi conçu une classe Image permettant de simplement charger une image d'entrée et la convertir en double dans n'importe quel intervalle de valeur. Nous ne serons ainsi pas limités par la précision des entiers non signés sur 8 bits et pouvons également facilement appliquer des changements d'intervalles pour certains filtres. Lorsque l'image est enregistrée sur le disque, elle est automatiquement convertie en intervalle [0-255].

Nous utilisons également une macro ColorSpace permettant de définir rapidement des fonctions de conversion d'espace couleur vers/depuis le RGB.

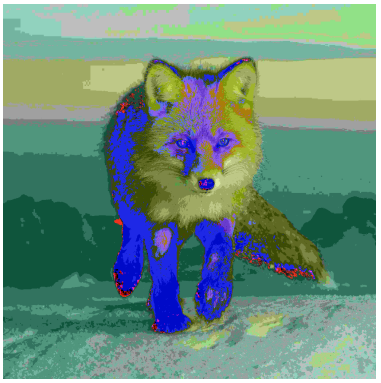
Premiers filtres naïfs

La première application que nous avons écrite "naive.out" permet, à partir d'une image donnée en entrée, d'exporter une série de copies obscurcies à l'aide de 3 méthodes : un flou intense, un coupage des bits de poids faibles en RGB et un coupage des mêmes bits en YCrCb. Voici les résultats ainsi qu'un exemple d'utilisation dans un terminal :

```
┌──(~/Documents/GitHub/Securite-visuelle/code/obscurator)
└─(15:25:16 on main ✖ ● ★)──> make build && ./naive.out ../images/RedFox.png
Built app : naive
Saved image : ../out/NAIVE_LPB_CUT_[RGB]_RedFox.png
Saved image : ../out/NAIVE_LPB_CUT_[YCrCb]_RedFox.png
Saved image : ../out/NAIVE_HEAVY_BLUR_[RGB]_RedFox.png
```

HEAVY BLUR

LPB CUT [RGB]



LPB CUT [YCrCb]

Nous avons choisis ces trois filtres car ils sont rapide à mettre en place pour un premier test de la base de code. Ils possèdent également une grande différence de donnees binaires avec l'original tout en gardant l'identification du sujet possible par un observateur humain.

Pistes pour la semaine prochaine

Nous souhaitons coordonner notre base de code C++ et notre base de code d'identification afin de choisir un format de base de code pour l'obscuration. Cela nous permettra de développer une interface pour appliquer tous nos filtres sur des bases de codes entières au lieu de simples images. Nous souhaitons également avoir de premiers résultats d'impacts de l'obscuration sur la reconnaissance d'images.

Reconnaissance par CNN

La sécurité visuelle repose sur l'incapacité à reconnaître une image après obscuration. Aussi l'idée de reconnaissance peut-être évaluée par niveau, par exemple dans le cas de la reconnaissance d'une personne, on pourrait d'abord chercher à savoir s' il s'agit d'une personne, puis essayer de reconnaître sa tranche d'âge puis enfin reconnaître précisément la personne. Ainsi selon des critères prédéfinis on peut établir un niveau de reconnaissance. Alors le niveau de sécurité visuelle d'une image peut être lui aussi évalué en fonction du niveau de reconnaissance atteint pour une image après obscuration.

Ainsi, évaluer l'appartenance à une classe assez vague comme chat ou chien correspond à un niveau de reconnaissance. Une première étape semble donc être la construction d'un classifieur capable de déterminer l'appartenance à ces classes simples. Pour cela on va utiliser un CNN entraîner sur le dataset cifar-10 [3]. Ce dataset possède 10 classes et 10 000 images par classe. Donc il est facile d'entraîner un CNN sur ce dataset, cependant il sera probablement nécessaire à terme de choisir ou de construire un dataset plus spécialisé à notre problématique de sécurité visuelle.

Modèle de CNN utilisé :

On est resté assez simple dans la construction de notre modèle. On a d'abord 3 couches de 32, 64 puis 128 filtres avec "reLu" comme fonction d'activation puis un max pooling (2, 2) pour chacune d'entre elles. Pour l'optimiseur on utilise "adam" et notre objectif est de maximiser la précision dans un premier temps. Sur les données de test de cifar 10 on obtient une précision de 86,54% au bout de 24 sur des batchs de 512. Ceci est une première version de notre modèle uniquement construite expérimentalement.

Image de chat reconnu comme appartenant à la classe Chat

Image de chat reconnu comme appartenant à la calsse Avion

Résultat :

En utilisant les méthodes d'obscurations mise en œuvre dans un premier temps, on observe des difficultés pour le CNN à classer correctement l'image et cela même si l'image est reconnaissable facilement pour un humain.

Image reconnue comme un chat

Image reconnue une grenouille

Ceci dit, en améliorant notre modèle de CNN, on peut espérer obtenir de meilleurs résultats.

Objectif pour la suite :

Nous souhaitons par la suite améliorer notre modèle de CNN ainsi qu'augmenter son temps d'entraînement afin d'obtenir de meilleurs résultats. De plus, nous voulons aussi permettre une reconnaissance plus précise que des classes aussi peu strictes afin de permettre une évaluation de l'obscuration par niveau.

Annexes

Références

[1] - STB Image - Librairie C++ - https://github.com/nothings/stb

[2] - Colorm - LIbrairie C++ - https://github.com/soreja/colorm

[3] - Cifar-10 - Dataset - https://www.cs.toronto.edu/~kriz/cifar.html

Dépot Github

- https://github.com/Patateon/Securite-visuelle