Description du code, projet TIM: La méthode de Viola et Jones et les caractéristiques de Haar

"Rapid Object Detection using a Boosted Cascade of Simple Features" - Paul Viola et Michael Jones, Cambridge, 2001

Léopold Maillard
Mathias Van Audenhove
Lucie Clair

Octobre - Décembre 2020

Table des matières

1 Principe et objectif de l'algorithme

La méthode de Viola et Jones est une méthode de reconnaissance de forme très efficace et révolutionnaire à sa parution. Leur objectif était de résoudre deux gros problèmes que pose la reconnaissance faciale : la vitesse et la robustesse. Le principe est d'obtenir un classifieur permettant de reconnaître efficacement la présence d'un objet d'intérêt au sein d'une image. Ce classifieur est formé en calculant puis en sélectionnant des caractéristiques à partir d'exemples d'apprentissage supervisé. On forme ainsi une cascade de classifieurs, qui appliquée à une image test, permet de reconnaître rapidement l'objet d'intérêt, peu importe sa taille ou sa position dans l'image.

2 Structure de l'archive

L'archive est composée de cinq documents : la documentation du code, deux fichiers ipynb d'implémentation de la méthode et d'applications et enfin les deux fichiers HTML correspondants. En plus de cela, elle contient nos différentes ressources : les images et vidéos utilisées lors de l'implémentation ainsi que le modèle pré-entrainé d'OpenCV.

3 Partie implémentation

3.1 Description du code et objectifs

La première implémentation de la méthode de Viola et Jones est le calcul de l'image intégrale, qui permettra de calculer efficacement les caractéristiques de Haar. Ainsi, dans la fonction "image_integrale", à partir d'une image passée en entrée, on renvoie son image intégrale. Pour cela, on calcule la valeur de chaque pixel de l'image intégrale, c'est à dire la somme des pixels situés au dessus et à gauche de chaque pixel de coordonnées x, y de l'image d'origine. À la fin de la fonction, on rajoute une ligne et une colonne de "0" en haut et à gauche de l'image.

Dans une deuxième fonction, on va extraire l'ensemble des caractéristiques rectangulaires de Haar de 5 types différents. Ainsi, dans la fonction "extract_haar_features", on prend en entrée une image intégrale et on renvoie la liste de ses caractéristiques de Haar. Les 5 types de caractéristiques seront calculés, pour chaque position dans l'image, pour toutes les longueurs et pour toutes les largeurs possibles. Quelques accès à l'image intégrale permettront de calculer la somme des pixels des différentes zones (blanches ou noires) des caractéristiques. On ajoutera enfin la valeur associée à la liste en précisant le type de la caractéristique, les coordonnées du pixel où elle a été calculée, ainsi que que sa longueur et sa largeur.

La dernière implémentation correspond à l'application d'une caractéristique

rectangulaire à tous les points d'une image. Similairement au calcul des caractéristiques, pour chaque pixel de l'image, on calcule la somme des pixels situés dans la zone noire de la caractéristique à l'aide de l'image intégrale, que l'on soustrait à la somme des pixels dans la zone blanche.

3.2 Analyse des résultats obtenus

Le calcul de l'image intégrale donne un résultat satisfaisant. Comme l'affichage via la fonction "imshow" de pyplot normalise les valeurs, on voit que les pixels prennent des valeurs de plus en plus élevées quand on se rapproche du coin inférieur droit de l'image. L'image intégrale est calculée assez rapidement et son intérêt dans la méthode de Viola et Jones est qu'elle n'est calculée qu'une seule fois.

La fonction d'extraction des caractéristiques permet bien d'obtenir toutes les caractéristiques d'une image. On voit qu'il en existe énormément, pour une image 24x24 pixels nous en obtenons 162 336 différentes. Le temps d'exécution n'est pas négligeable, mais ce n'est pas forcément un problème car le calcul de toutes les caractéristiques ne se fait que pour l'apprentissage, par lors de la détection. Il ne faut pas oublier que seul un petit nombre de ces caractéristiques seront sélectionnées pour détecter un objet.

L'application d'une caractéristique pour reconnaître la zone des yeux donne aussi des résultats cohérents. On remarque bien une zone plus claire aux coordonnées des yeux, car on a soustrait la somme des pixels d'une zone sombre à celle d'une zone claire de l'image.

3.3 Pistes d'amélioration

On aurait pu implémenter le calcul d'autres caractéristiques de Haar, comme des caractéristiques rectangulaires inclinées (à 45° par exemple). Ces dernières permettent de mieux prendre en compte les structures inclinées au sein d'une image. Il a été ainsi montré que leur utilisation apportait une amélioration des performances par rapport à celles utilisées par Viola et Jones.

4 Partie applications

4.1 Description du code et objectifs

Afin de mettre en pratique la méthode de Viola et Jones, nous avons décidé de reconnaître des visages dans différentes situations. Pour chaque cas d'usage, nous utilisons un code similaire qui vient s'adapter à la situation. La première étape est de charger le modèle pré-entrainé de OpenCV "Haarcascade frontalface default.xml" puis de créer un objet correspondant que l'on appellera "faceCascade". La création de cet objet se fait à travers la méthode CascadeClassifier de OpenCV. Ensuite, on détecte et stock l'ensemble des visages avec la

méthode "detectMultiscale". Le scale factor correspond au facteur multiplicatif de la fenêtre de recherche. En cas de détection multiple d'un même visage, il faut minNeighbors détections pour que la détection soit comptabilisée.

- Détection sur une photo : simple détection puis dessin du rectangle pour chaque visage détecté.
- Ajout d'un filtre "Snapchat" : détection et dessin du rectangle et des oreilles. Les oreilles sont remises à la bonne taille (en conservant l'échelle) puis copiées pixel par pixel au dessus du visage.
- Détection en temps réel sur vidéo : On charge la vidéo puis dans une boucle on détecte les visages image par image. Pour un soucis de rapidité, on convertit l'image en niveaux de gris. On ajoute une écriture dans une jolie police pour afficher le nombre de visage détectés. Enfin, on quitte la boucle uniquement lorsque l'utilisateur appuie sur la touche "q".
- Détection en temps réel par webcam : Cette méthode est très similaire à la précédente, la principale différence est qu'on charge la vidéo depuis la webcam. A partir de là on développe deux applications :
  - Ajout du filtre en temps réel
  - Détection du non-respect de la distanciation

4.2 Analyse des résultats obtenus

La détection de visage par la méthode de OpenCV fonctionne globalement assez bien. Elle est cependant sensible à l'éclairage mais aussi à l'orientation du visage. Un visage qui n'est pas tout à fait face à la caméra n'est souvent pas détecté. De plus, on a pu voir notamment sur la vidéo de Titanic que la méthode détecte parfois des visages là où il n'y en a pas. Pour ce qui est de la vitesse de détection, cela se fait en temps réel. Les vidéos restent tout à fait fluide même avec la détection de plusieurs visage à la fois.
En revanche, lorsque nous ajoutons le filtre Snapchat au dessus du visage, la vidéo n'est plus fluide. En effet, copier les pixels un par un est très coûteux.

4.3 Pistes d'amélioration

Concernant, la méthode de reconnaissance de visage en elle même, nous ne pouvons pas réellement améliorer la méthode d'OpenCV. En revanche, nous aurions pu essayer d'implémenter notre propre méthode ce qui nous aurait permis d'adapter parfaitement notre modèle à l'usage que nous voulions en faire. Mais, soyons honnête, nous n'aurions certainement pas fait mieux qu'OpenCV.
D'autre part, nous pourrions améliorer notre technique de réalité augmentée. Nous avons "bêtement" copié les pixels un par un sur l'image car nous n'avions pas connaissance des technologies existantes. En effet, des recherches académiques plus poussées nous permettraient d'implémenter l'ajout des oreilles de manière bien moins coûteuse.