Rapport de projet de groupe: Détection des Contours de Pièces

A) Résumé

Le projet a pour objectif d'étudier et de mettre en œuvre des techniques de traitement d'images pour la détection des contours de pièces dans des images numériques. Dans le cadre de notre UE en Image du semestre 6 à l'Université Paris Cité, chaque groupe envoie une vingtaine de photos de pièces, dont l'ensemble constitue la base d'étude de détection de pièces de monnaies.

B) Introduction

La détection des contours d'objets dans des images est une étape essentielle dans de nombreuses applications de vision par ordinateur, telles que la reconnaissance d'objets, la segmentation d'images et l'analyse d'images industrielles. Cette tâche permet d'extraire les limites des objets présents dans une image, facilitant ainsi leur identification et leur analyse. Pour réaliser cette détection, diverses techniques de traitement d'images peuvent être utilisées, notamment les filtres de convolution, les détecteurs de bords comme Sobel et Canny, ainsi que les transformations morphologiques.

Dans ce contexte, des algorithmes peuvent être développés à la main ou en utilisant des bibliothèques comme OpenCV, qui offrent des fonctions optimisées et robustes pour la détection de contours et d'autres opérations sur les images.

Ce rapport de projet vous expliquera à travers ces deux méthodes (avec et sans OpenCV), quels sont les principes qu'on a utilisés, quels sont les résultats avec l'évaluation des erreurs, puis quelles sont les pistes d'améliorations.

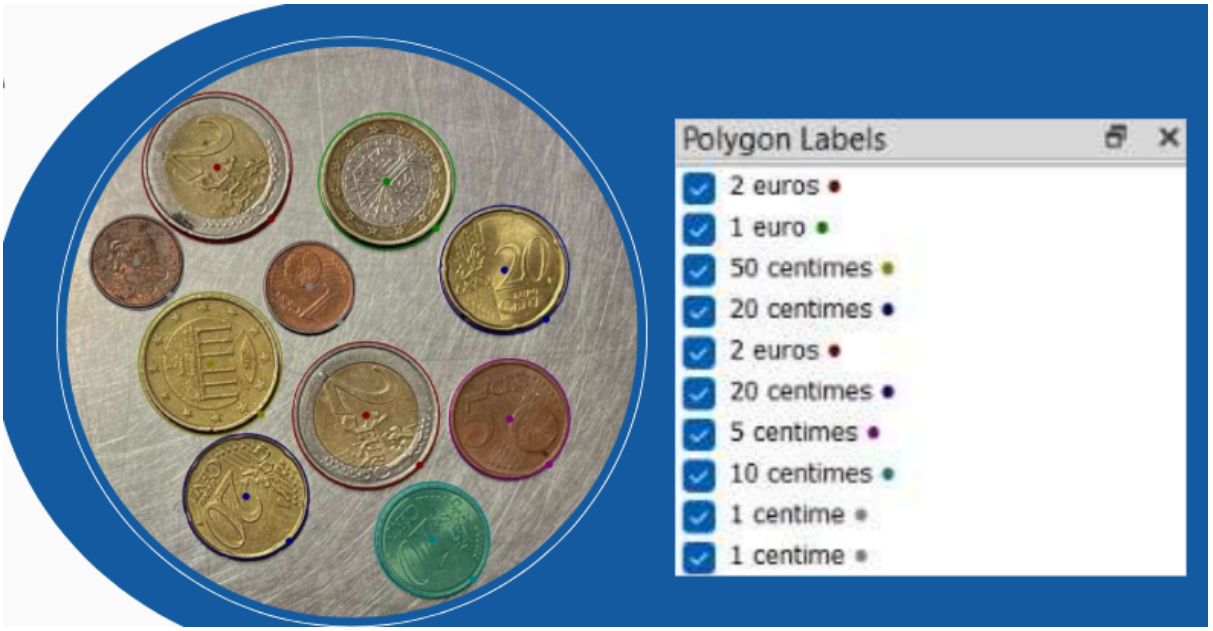
C) Sommaire

D) Méthodologie employée

1. Labellisation des images:

L'ensemble des images ont été labellisés avec le logiciel Labelme, en utilisant les contours en forme de cercle. Certaines pièces qui sont disposés différemment (posé sur le côté au lieu d'être posé sur le côté pile ou face) ou bien des pièces se superposant partiellement on été labellisé avec des polygones.
Chaque les différentes types de pièces d'euro ont été distinguées : 1 centime, 10 centimes, ..., jusqu'à 2 euros.

Applications:



Grâce au logiciel LabelMe, ces publications vont pouvoir être converties en fichiers .json, qui vont nous servir lors de la comparaison entre notre modèle (entrée) et la vérité terrain (cible).

2. Prétraitement des Images

Le prétraitement des images est une étape essentielle dans la préparation des données pour la détection des contours. Nous avons effectué le prétraitement d'une banque de 140 images en appliquant plusieurs techniques de traitement d'images. Ces étapes permettent de réduire le bruit, d'améliorer la qualité des bords, et de préparer les images pour la détection des contours.

L'ensemble des fonctions suivantes ont été implémentés, mais l'utilisation d'OpenCV a également été sollicitée pour des soucis de performances, et d'optimisation.

a. Filtres de Convolution et Padding

Les filtres de convolution sont utilisés pour appliquer des transformations locales sur l'image. Nous avons implémenté les filtres classiques tels que les filtres moyenneurs, médians, gaussiens avec paramètres modifiables. Durant le projet, nous avons choisi d'utiliser le filtre gaussien (Gaussian Blur) pour réduire le bruit.

Le padding est appliqué pour éviter que les bords de l'image ne soient affectés par ces transformations. Le zero padding, copy padding ou le symmetric padding ont été implémentés.

b. Filtres de Sobel

Les filtres Sobel sont utilisés pour détecter les bords horizontaux et verticaux dans une image à l'aide des filtres de de gradient horizontaux et verticaux. Ces filtres permettent de détecter les transitions de couleur ou de luminosité, qui sont souvent les bords des objets pour la majorité de nos images. Nous avons appliqué les filtres Sobel à chaque image pour identifier les bords des pièces.

c. Détecteur de Bords de Canny

Pour des soucis de performances , nous avons décidé de délaisser le filtre de Sobel pour nous concentrer sur le filtre de Canny. Le détecteur de bords de Canny est un algorithme plus sophistiqué qui permet de détecter les bords en identifiant les changements rapides de l'intensité des pixels.

Les 4 étapes ont été les suivantes:

a) Filtre gaussien

b) Calcul des gradients et des directions

c) Suppression des non maximums en 8-connexité

d) Sélection des noeuds que l'on décide de garder en fonction de min_value et max_value

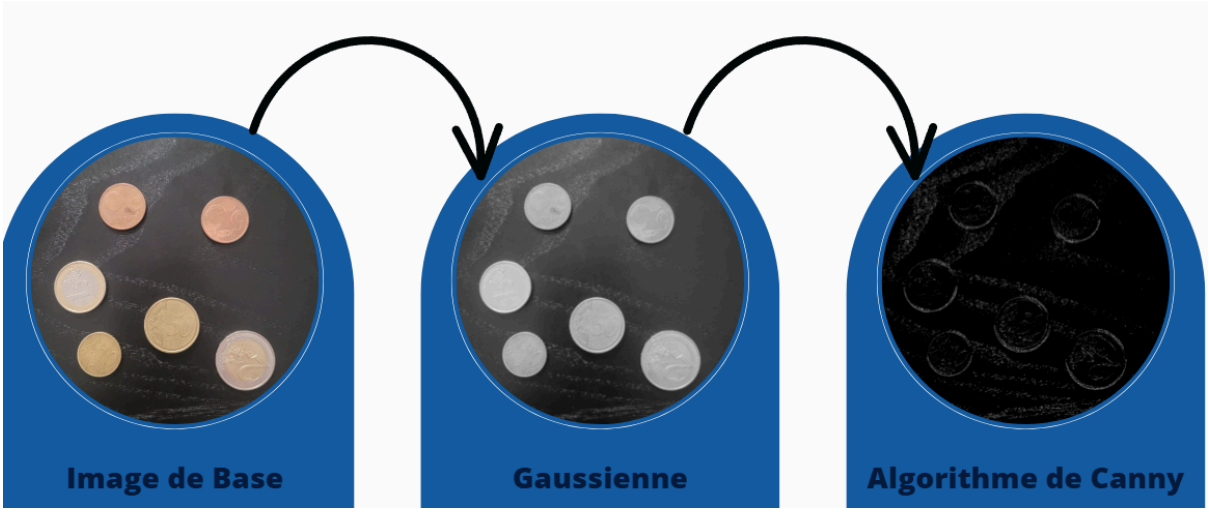
Ce filtre a été appliqué sur toutes les images pour obtenir une carte binaire des bords détectés, qui est ensuite utilisée pour identifier les contours des pièces.

d. Transformations Morphologiques

Les transformations morphologiques, comme l'érosion et la dilatation, sont des notions qui ont été utilisées pour affiner les résultats de la détection des bords. L'érosion réduit les objets dans une image, tandis que la dilatation les agrandit. Ces opérations ont été appliquées pour améliorer la détection des contours et pour combler les trous dans les objets détectés.

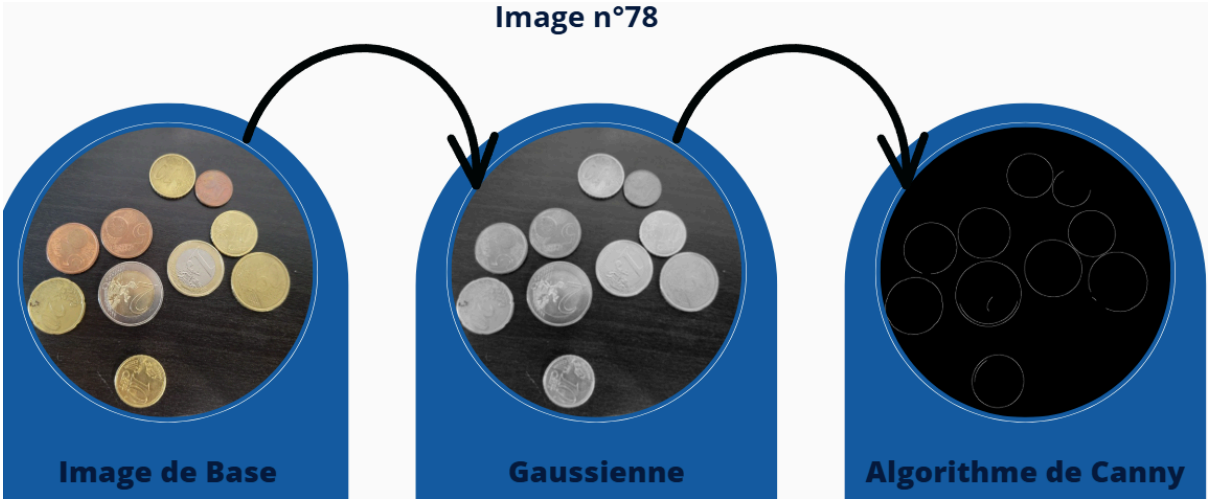
Même si une ouverture (erosion, dilatation) semble plus logique pour notre problème, nous avons choisi la fermeture (dilatation, érosion) utilisant un élément structurant sous forme de croix (Matrice [[0,1,0],[1,1,1],[0,1,0]]). Avec la contrainte de temps, nous avons mesuré la transformation morphologique qui donnait les meilleurs résultats expérimentalement.

<u>Applications:</u>



Dans un premier temps, on a l'utilisation de nos propres fonctions pour l'utilisation du filtre gaussien, puis de l'algorithme de Canny.

On remarque que les contours ont été majoritairement détectés sur cette image, mais que les variations du fond de l'image ont été également détectées par notre image. L'utilisation d'une fermeture ou ouverture n'a pas donné visuellement de résultats notables, mais en théorie, elle devrait bien fonctionner.



Dans un second temps, l'utilisation de l'algorithme de Canny avec OpenCV permet d'avoir un contour fin, précis et sans nuisance dans le fond de l'image. On verra cependant que parfois pour des images plus complexes constituant une minorité de la base (voir exceptionnelle), il y aura des faux positifs.

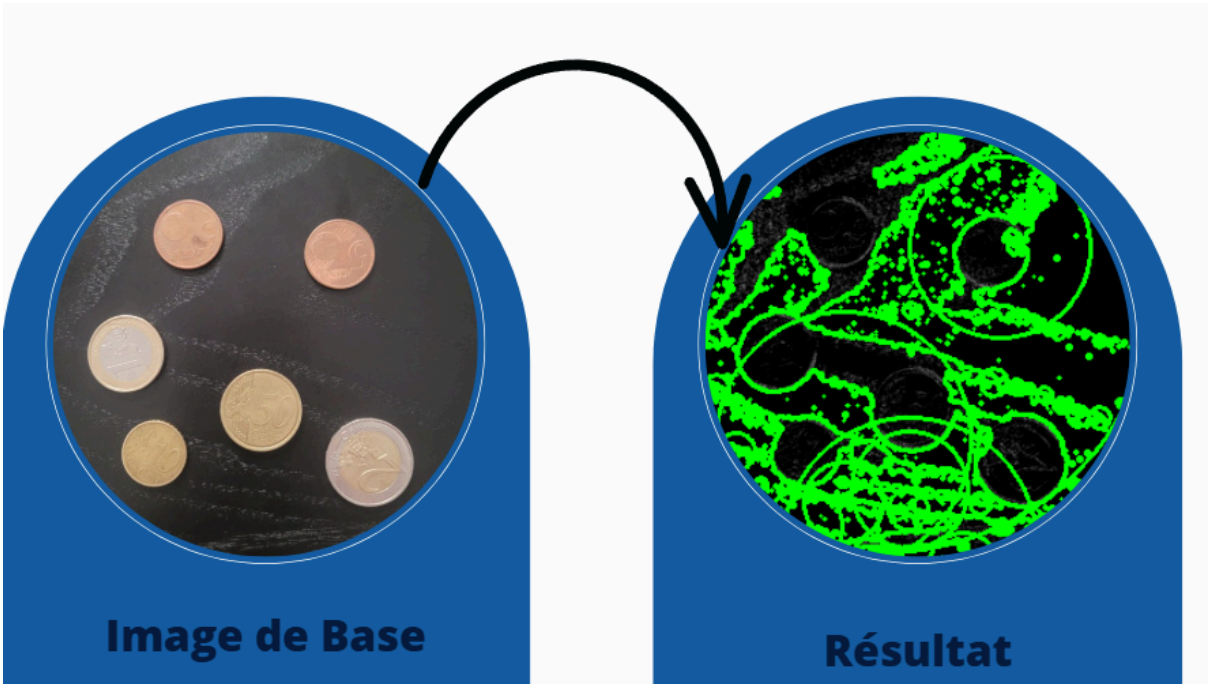
3. Détection des Contours

Une fois les images prétraitées, nous avons utilisé la fonction cv2.findContours d'OpenCV pour détecter les contours dans les images. Cependant, deux versions du code ont été développées pour la détection des contours :

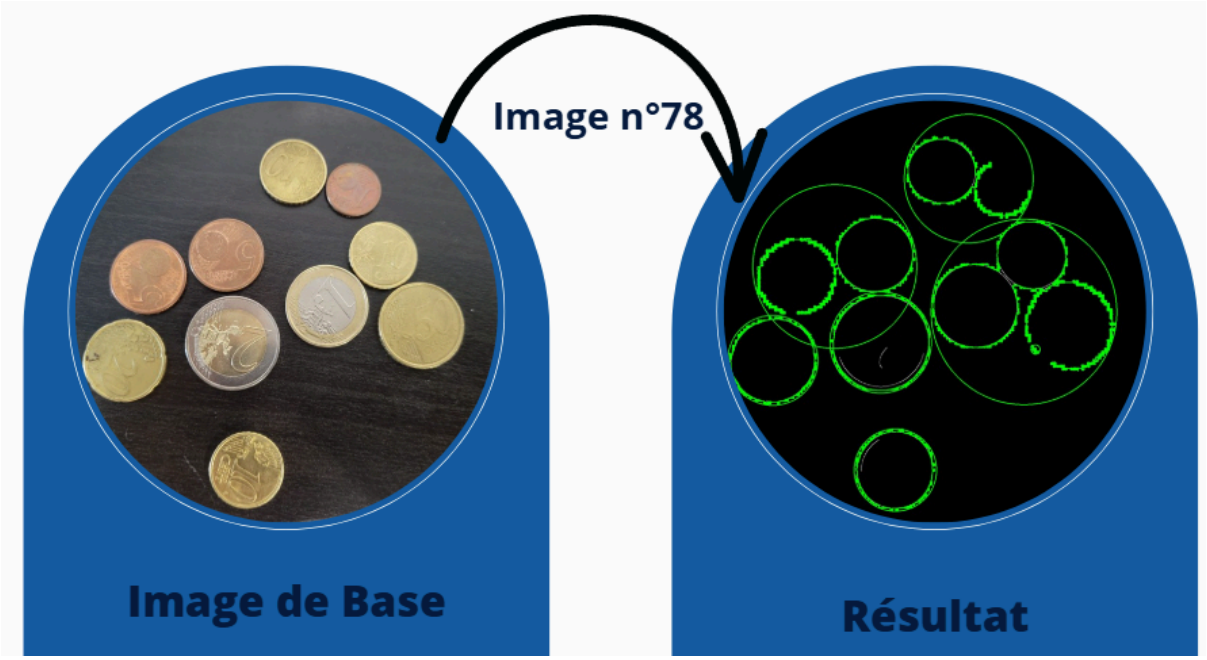
En premier temps, nous avons considéré des méthodes comme la transformation de Hough pour la détection des cercles, ainsi que la méthode de l'enveloppe convexe pour extraire les contours. Cependant, nous avons par la suite décidé de nous tourner sur d'autres fonctions.

La deuxième version repose entièrement sur la fonction cv2.findContours d'OpenCV, qui est optimisée pour la détection de contours dans les images. Cette version offre des performances plus rapides et est plus robuste face aux artefacts d'image.

Applications:



Voici un exemple d'application de findContours sur l'image prétraitée avec nos fonctions. On remarque que les moindres détails ronds sont détectés comme pièces. Il y a très fréquemment de faux positifs (lorsque des pièces non existantes sont détectées) et de faux négatifs (quand les pièces ne sont pas détectées).



On remarque que la fonction fonctionne bien mieux sur l'image prétraitée avec les méthodes d'OpenCV. Cependant, ici on remarque 3 faux positifs.

4. Association des Contours aux Annotations

Une fois les contours détectés, nous avons comparé les résultats avec les annotations sous forme de JSON pour identifier les pièces dans l'image. Les annotations contiennent les coordonnées du centre et du rayon des pièces. Nous avons utilisé ces informations pour associer chaque contour détecté à une annotation.

5. Choix du Ratio 60/20/20 pour la Base d'Entraînement, de Test et de validation

Pour évaluer les performances de nos algorithmes, nous avons choisi de diviser notre base de données d'images en deux sous-ensembles : un ensemble d'entraînement et un ensemble de test. Nous avons utilisé un ratio de 60/20/20, ce qui signifie que 60 % des images ont été utilisées pour l'entraînement de nos modèles, tandis que 20 % ont été réservées pour l'évaluation des performances. Les 20 derniers % sont gardés pour la validation de notre modèle.

6. Conversion des Images en JSON

Afin de faciliter l'annotation et la gestion des données, nous avons converti les informations relatives aux pièces détectées (centres et rayons) en un format JSON, grâce au logiciel

LabelMe. Cela permet de stocker facilement les métadonnées associées à chaque image, y compris les informations sur les contours des objets annotés.

E) Résultats

Les résultats de l'évaluation des performances ont montré des différences importantes entre les deux versions du code, tant en termes de précision que de performances.

Évaluation des Erreurs

Sans OpenCV (code personnalisé) :

a) MAE “moyen” par image = 9,1
b) MAE sur l'ensemble de la base (22 images) = 206
c) MSE “moyen” = 315,1
d) MSE sur l'ensemble de la base = 6302

L'évaluation des erreurs montre que la méthode sans OpenCV présente une erreur importante, sur uniquement 22 images prétraitées. Il s’agit probablement en raison de l'implémentation rudimentaire des fonctions, une performance moindre pour l’implémentation du filtre de Canny, ou alors l’utilisation d’une fermeture au lieu d’une ouverture.

L'erreur moyenne par image est relativement élevée, ce qui suggère que les techniques de prétraitement utilisées sont moins robustes.

Avec OpenCV :

a) MAE “moyen” par image = 2,9
b) MAE sur l'ensemble de la base (140) = 616
c) MSE “moyen”= 20
d) MSE sur l'ensemble de la base = 5700

L'utilisation d'OpenCV a permis de réduire considérablement l'erreur, en particulier la MAE par image (environ 3 pièces mal détectées par image). On remarque un MSE et un MAE du même ordre de grandeur, alors qu’on a appliqué notre modèle sur 140 images au lieu de 22 images. Les performances restent assez médiocres en utilisant les fonctions d’openCV, mais étant donné la complexité des images, il s’agit de statistiques bien plus satisfaisantes que notre première méthode.

F) Problèmes rencontrés

Lors de notre projet de détection de contours de pièces, plusieurs problèmes sont survenus, chacun touchant différentes étapes du travail. Voici un résumé des principaux défis rencontrés :

1. **Étiquetage des labels**

Un problème majeur a été lié à l'étiquetage des labels dans les fichiers JSON. Les annotations étaient parfois imprécises, et les formes étaient mal définies ou incomplètes, ce qui rendait difficile l'association correcte entre les contours détectés et les pièces réelles sur l'image. Bien qu'il s'agit d'une erreur humaine de notre groupe, l'uniformisation entre chacun a été difficile, donc on a dû repasser plusieurs fois sur cette partie pour s'adapter à notre code.

2. **Recoder les fonctions d'OpenCV**

Nous avons aussi essayé de recréer certaines fonctions d'OpenCV pour mieux comprendre leur fonctionnement. Cependant, reproduire des fonctions comme cv2.findContours n'a pas été facile. La gestion des contours et leur précision n'étaient pas toujours au rendez-vous, ce qui nous a obligés à revenir à l'utilisation des fonctions d'OpenCV, qui sont plus efficaces et fiables pour ce genre de tâche.

3. **Les images avec des fonds complexes**

Les erreurs ont été principalement dues à un fond d'image pouvant perturber la détection d'images. Par exemple, il y avait des images avec pour fond un matelas avec des motifs rond (qui pouvaient donc être détectés comme des pièces), ou alors des images avec une table rayée qui donc avec le filtre de Canny, a été détectée comme positive pour les contours.

4. **Temps de pré-traitement avec nos méthodes**

Une autre difficulté est venue du temps de calcul. Nos propres fonctions étaient beaucoup plus lentes que celles d'OpenCV. Le prétraitement prenait beaucoup de temps, surtout pour des images complexes, ce qui rendait l'analyse d'un grand nombre d'images (comme notre base de 140 images) plus lente et moins efficace.

5. **Détection sans les labels JSON**

L'utilisation des cibles a été fondamentale pour nous aider à détecter la présence des pièces. Mais sans les fichiers JSON, il nous était difficile de concevoir un modèle non supervisé. Cette indépendance peut être un axe d'amélioration.

G) Pistes d'améliorations

1. **Optimisation des paramètres :** Le choix des paramètres de Canny et Sobel peut être optimisé pour des images spécifiques, notamment en ajustant les seuils de détection des bords.

2. **Méthodes Avancées :** L'utilisation de techniques d'apprentissage automatique pourrait améliorer la précision de la détection dans des images complexes, par exemple en utilisant des réseaux neuronaux convolutifs (CNN).

H) Conclusion

Le projet a montré que la détection des contours des pièces peut être réalisée efficacement en utilisant des techniques de traitement d'images classiques, telles que les filtres Sobel, Canny et les transformations morphologiques. Bien que les deux versions du code (personnalisée et OpenCV) aient montré des résultats similaires, la version OpenCV est plus rapide et plus robuste, ce qui la rend idéale pour des applications à grande échelle.

Le projet pourrait être amélioré en utilisant des techniques d'apprentissage automatique pour affiner la détection et l'analyse des contours dans des images plus complexes.