Compte-rendu 5 Projet Image

Détection de falsifications dans des images

Aissa BERBER Pierre RICHARD

24/03/24

1 Ce qui a été fait

L'utilisation de la luminence et de la détection de contours à été mis en place et des images ont été extraite de certaines bases de donnée. Les précédentes fonctions ont également été adaptées aux bibliothèques python. Les résultats avec la luminence de sont pas satisfaisant à notre avis, il faudrait la combiner avec d'autre méthode afin de mieux repérer les falsifications d'images.

2 détection de contours

2.1 Convertir l'image en niveaux de gris

La conversion en niveaux de gris est la première étape de nombreuses tâches de traitement d'image. Ce processus simplifie l'image en la réduisant à une échelle de gris, en éliminant les informations de teinte et de saturation tout en conservant la luminance. L'équation utilisée pour cette conversion est :

$$Gris = 0.299 \times R + 0.587 \times G + 0.114 \times B \tag{1}$$

où R, G, et B représentent respectivement les composantes rouge, verte et bleue d'une image RVB.

2.2 Appliquer un flou gaussien pour réduire le bruit

Le flou gaussien est appliqué pour lisser l'image en réduisant le bruit et les détails. Ce processus utilise une fonction gaussienne pour flouter l'image, réalisé en convoluant l'image avec un noyau gaussien. La fonction gaussienne en deux dimensions est donnée par :

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2}$$

où x et y sont les distances par rapport à l'origine dans les axes horizontal et vertical, respectivement, et σ est l'écart type de la distribution gaussienne.

2.3 Appliquer la détection de bords de Canny

La détection de bords de Canny est un algorithme en plusieurs étapes conçu pour détecter une large gamme de bords dans les images. L'algorithme comprend :

- Filtrer le bruit avec un filtre gaussien
- Trouver le gradient d'intensité de l'image
- Appliquer la suppression non maximale pour éliminer la réponse spure aux détections de bords
- Appliquer un seuil double pour déterminer les bords potentiels
- Utiliser le suivi de bord par hystérésis pour finaliser la détection des bords

La complexité de l'algorithme de détection de bords de Canny signifie qu'il ne correspond pas à une seule équation mais est un processus impliquant plusieurs étapes de calcul.

2.4 Trouver les contours

Trouver les contours implique de détecter les bords des objets dans une image, essentiellement trouver la limite des objets blancs contre un fond noir. Ceci est basé sur les bords détectés dans l'étape précédente. La fonction `cv2.findContours` est utilisée pour identifier ces contours. Le mode de récupération `cv2.RETR_EXTERNAL` et la méthode d'approximation `cv2.CHAIN_APPROX_SIMPLE` sont spécifiés pour optimiser le processus de détection des contours. Cette étape ne correspond pas à une simple équation mathématique mais plutôt à un processus de calcul qui identifie et suit les bords des objets dans une image.

3 Dessin de contours et segmentation d'image

Après la détection des contours, les étapes suivantes impliquent de dessiner ces contours et de segmenter l'image en fonction de ceux-ci. Nous discutons ici du code qui réalise ces tâches.

3.1 Dessiner des contours sur un canevas

Tout d'abord, un canevas est préparé avec les mêmes dimensions que l'image originale mais initialisé à zéro, créant essentiellement une image vierge.

```
canvas = np.zeros((height, width, 3), dtype=np.uint8)
```

Pour chaque contour détecté précédemment, nous vérifions si sa surface est significative (supérieure à 0,5 dans ce cas). Pour ceux qui se qualifient, nous dessinons les contours sur le canevas avec une couleur spécifiée (vert dans ce cas, représenté par (0, 255, 0) en RGB) et une épaisseur de ligne.

```
for contour in contours:
    if cv2.contourArea(contour) > 0.5:
        cv2.drawContours(canvas, [contour], -1, (0, 255, 0), 2)
```

3.2 Segmenter l'image en fonction des contours

La segmentation implique d'isoler les parties de l'image correspondant aux contours détectés. Pour chaque contour :

- Un masque est créé avec les mêmes dimensions que l'image originale. Initialement, le masque est vierge (rempli de zéros).
- Le contour est dessiné sur le masque avec `cv2.FILLED` pour remplir la zone du contour, rendant le masque blanc (255) dans la zone du contour et noir (0) ailleurs.
- L'image originale et le masque sont combinés en utilisant l'opération ET binaire. Cette opération isole la zone de l'image dans le contour, segmentant efficacement l'image en fonction du contour.

```
for i, contour in enumerate(contours):
    mask = np.zeros((image_height, image_width), dtype=np.uint8)
    cv2.drawContours(mask, [contour], -1, (255), thickness=cv2.FILLED)
    segment = cv2.bitwise_and(image, image, mask=mask)
```

Ce processus résulte en segments individuels de l'image originale, chacun correspondant à un contour détecté. Ces segments peuvent être utilisés pour une analyse ou un traitement ultérieur, tels que l'extraction de caractéristiques, l'identification d'objets, ou simplement pour mettre en évidence des zones spécifiques d'intérêt dans l'image.

4 Ce qu'il y a à faire

Il faudrait avoir un début d'interface, pour cela nous allons utiliser la bibliothèque tkinter. Nous allons également combiner les techniques utilisées afin d'avoir de meilleurs résultats. La technique de détection par compression jpeg a aussi été recherché au cours de la semaine. Son principe ayant été compris son implémentation ne devrait pas s'avérer trop compliqué.

5 Lien du github

https://github.com/Ranger1986/projet-image