Cégep Limoilou

TP 1(complet)

420-SF3-RE

Pour commencer : Créez un nouveau projet Pycharm et utilisez un répertoire source différent par exercice

Exercice 1 : Traitement d'images avec Numpy- Détection de contours avec des noyaux de convolution

Objectif : *Manipuler des tableaux Numpy, et en profiter pour comprendre le principe des filtres en imagerie numérique et la détection de contours à partir de la convolution avec des noyaux*

Commencer par lire cet article qui explique le principe de filtre de convolution

Partie 1 :

Dans le fichier `convolution.py` implémenter la fonction `convolution()`, selon le principe de la fenêtre glissante expliquée dans l'article. Vous devez ici manipuler directement les tableaux numpy.

Indication : `np.pad(…)` permet de rejouter une bordure autour d'un tableau existant, ça va vous simplifier la vie!

(Observer la différence entre votre convolution et celle de `Numpy`, elle devrait être explicable)

Partie 2 :

Dans un fichier `exercice1.py` implémentez les étapes suivantes

1. **Charger une image**
   - Utilisez une petite image (par exemple `data.camera()` de `skimage.data` pour commencer). Il faut importer le module skimage
   - Affichez-la avec `matplotlib`.

2. **Appliquer un filtre de flou**

   On définit le noyau de moyenne (blur) 3×3 :

$$K_{flou} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- Convoluez (en utilisant `signal.convolve2d`) l'image avec ce noyau.
- Affichez le résultat **en niveau de gris** (paramètre `cmap= 'gray'` de la methode `imshow()`) et observez l'effet.

3. Détecter les gradients

On utilise les noyaux de Sobel :

Horizontal (dérivée par rapport à X) :

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Vertical (dérivée par rapport à Y)

$$K_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

- Convoluez l'image floutée avec K_x puis séparément avec K_y.
- Affichez les deux images **en niveau de gris** obtenues G_x et G_y (gradient horizontal et vertical).

4. Calculer l'amplitude du gradient

Combinez les deux images précédentes en calculant le gradient :

$$G = \sqrt{G_x^2 + G_y^2}$$

- Affichez cette nouvelle image G **en niveau de gris**, qui fait apparaître les contours.

5. Appliquer un seuillage

- Appliquez un seuil (par exemple 100). Autrement dit, transformez l'image en binaire : pixels blancs = contour détecté, pixels noirs = fond d'image, de sorte d'afficher les contours en noir et le fond en blanc

6. Préparer la remise

- Enlever les affichages précédents et afficher les 6 images simultanément sur 3 lignes et 2 colonnes avec des titres à chaque image pour présenter les résultats successifs
- Mettez de l'ordre dans votre code (utiliser des fonctions pour regrouper les opérations qui doivent l'être, etc.)
- Choisissez d'autres images de votre choix pour expérimenter les effets du seuil et sélectionnez en une pour la remise. Utiliser ici ceci pour ouvrir et convertir en niveaux de gris :

```
Image.open('mon_image.jpg').convert('L')
```

- Essayez d'améliorer le processus, en réfléchissant aux paramètres sur lesquels vous pourriez jouer et expliquez vos choix en commentaires!

Exercice 2 : Manipulation symbolique avec SymPy

Objectifs *:*

- *Déclarer des symboles et expressions symboliques*
- *Calculer des dérivées et intégrales*
- *Factoriser et simplifier des expressions*
- *Résoudre des équations et systèmes*
- *Tracer des fonctions*

1. On considère

$$f(x) = 3x^3 - 5x^2 + 2x - 7$$

   a) Afficher convenablement à la console la fonction f,

   b) Calculez la dérivée de $f(x)$,

   c) Résolvez l'équation $\mathrm{f}'(\mathrm{x}) = 0$ pour trouver les points critiques de f,

   d) Évaluez f aux points critiques pour obtenir les valeurs correspondantes,

   e) Calculer l'intégrale indéfinie de f,

   f) Calculer l'intégrale définie de f entre $x = 0$ et $x = 2$,

   g) Tracer les graphes de $f(x)$ et $f'(x)$ sur l'intervalle [−2,3] de deux couleurs différentes et bien identifiées,

h) Ajoutez la mise en évidence de points critiques et de leurs coordonnées,

i) Ajoutez le tracé de la tangente à f au point $x = -1$.

2. On considère le système :

$$\begin{cases} 2x + 3y + z = 2 \\ -x + 2y + 3z = -1 \\ -3x - 3y + z = 0 \end{cases}$$

a) Résoudre le système avec SymPy .

b) Vérifier la solution.

3. Soit une nouvelle fonction

$$g(x) = 6x^5 - 9x^4 - 49x^3 + 87x^2 - 17\,x + 30$$

a) Factorisez $g(x)$

b) Développez de nouveau pour vérifier.

Exercice 3 : Modélisation d'un atome avec affichage graphique des orbites électroniques

Objectif

- *Créer une classe Atome qui modélise un atome avec ses électrons répartis sur des orbites (couches électroniques)*
- *Visualiser graphiquement la distribution des électrons sur les orbites avec* ***matplotlib***
- *Manipuler l'orienté objet en python, en l'appliquant aux principes élémentaires de la structure atomique*

Classes à créer

- **Electron**
  Attributs :
  - `id (int)` : identifiant unique
  - `orbite (int)` : numéro de la couche (1, 2, 3, ...) où il se trouve
- **Atome**
  Attributs :
  - `nom (str)` : nom de l'atome (ex : "Carbone")
  - `numero_atomique (int)` : nombre de protons (= nombre d'électrons dans l'atome neutre)
  - `electrons (List[Electron])` : la liste des électrons, qui doit être crée et remplie dans le constructeur

  Méthodes :
  - `__init__(nom, numero_atomique)` : crée les orbites d'électrons selon le numéro atomique et la règle décrite plus bas
  - `orbites()` : retourne un dictionnaire {numéro_orbite: nombre_electrons}
  - `afficher_orbites ()` : affiche textuellement la configuration électronique dans la console
  - `dessiner_orbites()` : affiche **graphiquement** les orbites et les électrons à l'aide de matplotlib. Afficher les électrons en couleur différente selon l'orbite.

- **Tableau Périodique**

  Attributs :

  - `elements (List[Atomes])` : une liste d'atomes

  Méthode :

  - `__init__(List[Atomes])` : Utilise le fichier tableau_periodique.json pour initialiser le tableau grâce à une méthode de désérialisation
  - `get_atome(numero_atomique)` : Retourne l'atome correspondant au numéro atomique fourni

Règle simplifiée pour remplir les orbites

- Chaque orbite peut contenir jusqu'à $2\ n^2$ électrons, où n est le numéro de l'orbite
- Exemples :
  - Orbite 1 : max 2 électrons
  - Orbite 2 : max 8 électrons
  - Orbite 3 : max 18 électrons

Programme principal

- Charger le tableau périodique depuis le json fourni
- Demander le numéro atomique à afficher
- Afficher la configuration électronique dans la console
- Afficher la représentation graphique des orbites et électrons
- Demander un autre numéro atomique en gérant une condition de sortie

Conseils pour la réalisation :

- Commencer par implémenter les classes `Atome` et `Electron` sans l'affichage graphique avec `matplotlib`, juste le textuel. Tester votre code avec quelques atomes instanciés manuellement.
- Implémenter ensuite l'affichage graphique de l'atome, n'hésitez pas à faire des recherches pour cette partie (comment dessiner un cercle en `matplotlib`, comment dessiner le noyau, etc..)

- Implémentez la classe `TableauPeriodique` et sa désérialisation json depuis le fichier
- Implémentez la boucle du programme principal pour mettre le tout ensemble.