Le miniprojet est à faire par groupes de deux ou individuellement. Les groupes peuvent s'échanger des idées ou des approches générales, mais pas du code directement.

Préparation

Téléchargez sur moodle l'archive **`miniproject_b.zip`**. Vous y trouverez le fichier **`miniproject_b.py`** que vous devez placer dans le même dossier que les fichiers **`miniproject.py`** et **`miniprojectutils.py`**. Si ce n'est pas déjà fait, vous pouvez ensuite réouvrir le dossier dans Visual Studio Code pour continuer le projet. Vous trouverez les signatures des nouvelles fonctions à implémenter dans **`miniproject_b.py`**, ainsi que de quoi vous aider à tester votre code. Il est recommandé de faire une backup avant de continuer.

Lancez le fichier miniproject_b.py. Il devrait afficher quelques lignes sur la console en disant qu'il commence le travail et qu'il charge des images, puis s'arrêter juste après. Si cela ne fonctionne pas, appelez un·e assistant·e.

Partie 1. Flouter pour mieux réduire

Travailler avec des grandes images peut parfois demander des longs temps de calcul. Pour remédiez à ce problème, il est parfois plus simple de réduire la résolution de l'image avant de l'utiliser.

Commencez par implémenter la fonction **`simple_blur`**, dont le but est de flouter une image. Cette fonction reçoit deux arguments : l'image d'entrée en niveaux de gris **`img_gray`**, et un entier **`ksize`**. La fonction doit retourner une copie flouté de l'image d'entrée. Le flou a appliquer est un simple "box blur" de dimension **`ksize`**×**`ksize`**, c'est à dire le flou obtenue par la convolution avec un noyeau (ou kernel) carré de cette taille, dont tout les poids sont égaux.

Vous devez implémenter le box blur en 2 passes: une première passe qui applique un flou horizontal ou chaque ligne est flouté avec un noyau de taille $1\times$**`ksize`**, suivi d'une seconde passe floutant verticalement avec un noyau de taille **`ksize`**$\times 1$. Cette approche permet une implémentation performante du box blur. Pour implémenter chaque passe, vous devez utiliser la function **`np.convolve`** (vu en cours) avec le mode "valid". Chacune des 2 passes de convolution applique ainsi un flou de mouvement qui réduit respectivement la largeur ou la hauteur de l'image de $ksize - 1$.

Notez qu'il est important que le kernel utilisé pour vos opérations de flou soit un array numpy constitué de floats et que leur somme soit (environ) égale à 1. Vous pouvez utiliser **`np.ones`**, **`np.zeros`** ou **`np.ndarray`** pour créer un array numpy de floats. Similairement, pour s'assurer de ne pas trop perdre en précision, le résultat intermédiaire obtenu lors de la première passe de flou doit aussi être constitué de floats. Pour finir, assurez vous d'arrondir au plus proche lorsque vous stockez le résultat de la 2ème passe dans l'image finale.

Décommentez les lignes correspondante de la fonction **`blur_test`** pour tester votre code. Comme précédemment vous pouvez ajouter d'autres tests en bas du fichier tant qu'ils sont placé dans la condition **`if __name__ == "__main__"`**.

image obtenu après un box blur de taille 4 (à gauche), et après downscale (à droite)

Partie 2. Accélération de la recherche d'un motif

Dans cette partie, vous allez implémenter une recherche de motif accéléré, en vous servant d'images à la résolution réduite.

(a) Implémentez d'abord **`find_most_likely_positions`**. L'objectif de cette function est le même que celui de **`find_similar_pattern_position`** que vous avez implémenté dans **`miniprojet.py`**, à l'exception du fait qu'elle doit retourner la liste des **`top_n`** positions avec les différences les plus faibles. Tout comme **`find_similar_pattern_position`**, cette fonction doit aussi faire appel à **`pattern_difference`**. Assurez vous de construire la liste en une seule passe, pour éviter de recalculer plusieurs fois les différences.

(b) Ensuite, implémentez **`faster_search`**. L'objectif de cette fonction est là aussi le même que pour **`find_similar_pattern_position`**, à l'exception qu'elle doit aller plus vite. Pour ce faire, vous devez commencer par obtenir une copie plus basse résolution de l'image et du pattern d'entrée en réduisant leur taille du facteur **`factor`** passé en argument. Utilisez ensuite **`find_most_likely_positions`** sur l'image et le pattern de résolution plus faible, en lui passant l'argument **`top_n`**. Vous obtenez alors une liste de positions probables du pattern, mais attention: ces positions ont été calculé sur l'image de résolution réduite, et vont servir d'indices pour retrouver le motif dans l'image originale.

La recherche dans l'image originale commence alors: Pour chaque position de la liste de positions probables obtenue, il faut reconstruire la position équivalente dans l'image originale, et chercher le pattern autour de cette position. Les positions[1] à essayer sont celle dans un carré centré autour de la position reconstruite, de taille **`factor`**$\times$**`factor`** si **`factor`** est impair ou **`(factor+1)`**$\times$**`(factor+1)`** si **`factor`** est pair. La position avec la plus faible différence ainsi trouvé est alors retourné.

(c) Pour finir, vous allez maintenant implémenter la fonction **`faster_highlight`**. Cette fonction est identique à la fonction **`highlight_similar_pattern`** que vous avez implémenté dans **`miniprojet.py`**, à l'exception qu'elle doit utiliser **`faster_search`** au lieu de **`find_similar_pattern_position`**. Elle prend 2 arguments supplémentaires **`factor`** et **`top_n`** et les passe à **`faster_search`**.

Testez votre code en décommentant les étapes de la fonction **`faster_highlight_test`**. Lorsque les valeurs de **`factor`** et **`top_n`** sont bien choisi, les résultats doivent être identiques à ceux obtenue à la fin de la partie 1. Mais quand **`factor`** est trop grand ou que **`top_n`** est trop petit, il est possible que ce nouvel algorithme ne trouve pas le pattern à la bonne position. Plus **`factor`** est grand, plus l'algorithme est rapide (jusqu'à un certain point). Vous pouvez modifier ou ajouter des appels à **`faster_highlight_test`** pour tester votre code avec d'autres configurations, images et patterns d'exemple.

Avec **`tablet.jpg`** et **`glyph.jpg`** et un **`factor`** de 3, la recherche rapide ne trouve pas la bonne position avec **`top_n=4`** (à gauche) mais avec **`top_n=100`** (à droite), l'algorithme retrouve correctement le pattern.

[1] Lorsqu'on parle de position, on parle toujours de la position du premier pixel du pattern. Les autres pixels peuvent être en dehors du carré de recherche.

Instructions pour la soumission

- Inscription aux groupes (sur moodle) :
  - Délai d'inscription : avant lundi 12 mai 2025
  - Même les étudiant.e.s seul.e.s doivent être dans un groupe
- Délai de rendu : 23 mai 2025 à 23h59
  - Rendez en avance, vous pouvez faire autant de soumissions que vous souhaitez !
- Deux fichiers à rendre (sur moodle) :
  - **`miniproject.py`**
  - **`miniproject_b.py`**
- Une seule personne doit rendre le projet par groupe
- Attention :
  - il ne faut pas modifier les signatures des fonctions que vous devez programmer !
  - Si votre code dépend de fonctions que vous avez ajoutées ou modifiées, ces fonctions doivent être présentes dans le fichier **`miniprojet.py`** ou **`miniprojet_b.py`**.
- Evaluation : uniquement de vos fonctions à compléter, pas du main.