EPFL

ICC

Programmation

GC/MX, **Cours 8**

18 novembre 2022

Jean-Philippe Pellet

Opening music: Bruce Springsteen, album *Born in the U.S.A.* (1984)

Previously, on Programmation…

- Types de base en Python: int, float, str, bool
- Méthodes, fonctions et slicing pour calculer des valeurs dérivées
- Conditions pour exécuter du code selon la valeur d'une expression booléenne: `if <condition>: ... else: ...` et ses variantes
- Boucles pour exécuter du code plusieurs fois:
  - Boucle `while <condition>: ...`
  - Boucle `for i in range(...): ...`
- Déclaration de fonctions avec type de retour et paramètres:
  - `def calculate_area(r: float) -> float: return ...`
- Utilisation de listes
- Utilisation de sets
- Utilisation de dictionnaires
- Déclaration de classes: `@dataclass class Rectangle: ...`

Cours de cette semaine

Manipulation d'images
Introduction au miniprojet

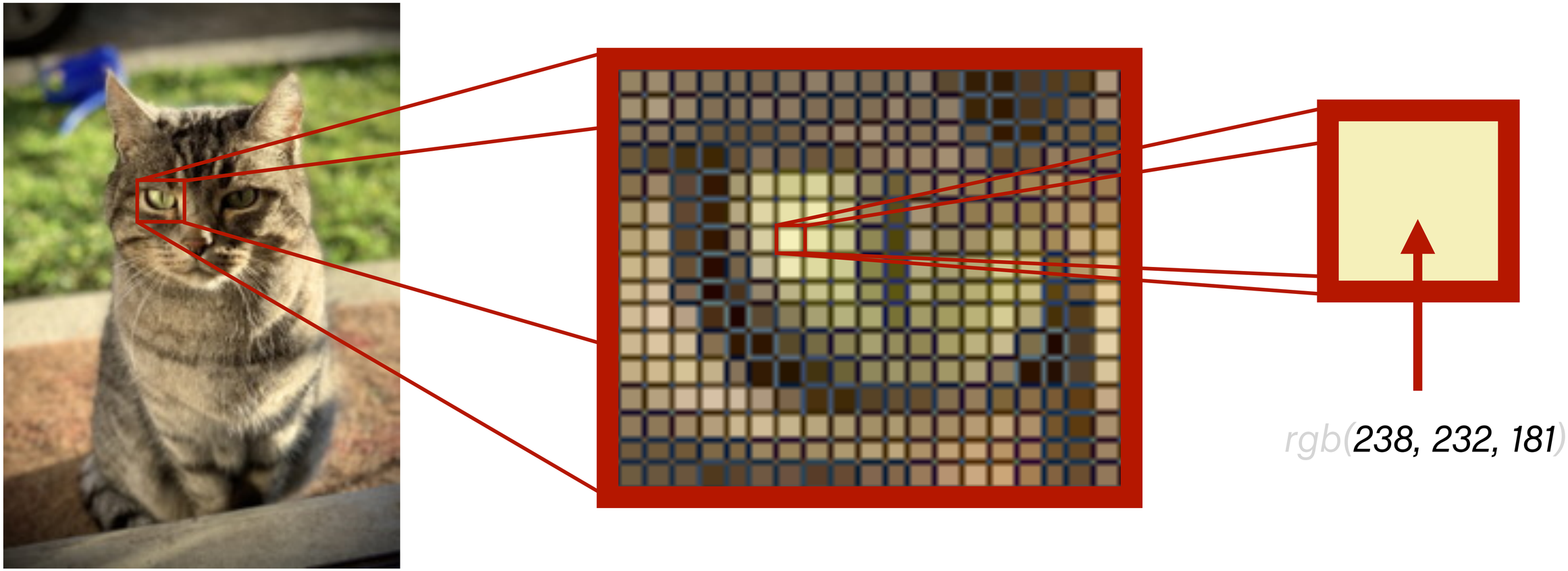
Cours de cette semaine

Manipulation d'images

Introduction au miniprojet

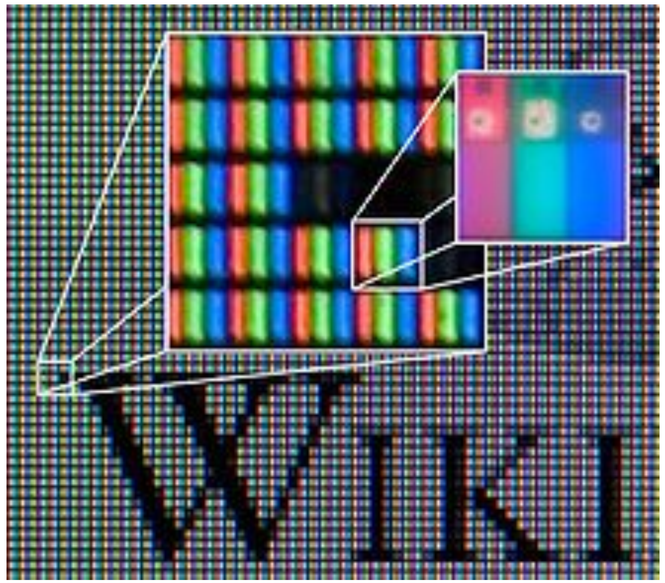
Représentation des images et couleurs

- Images matricielles (JPEG, PNG, …), images vectorielles (SVG, …)
- Image matricielle = grille de pixels (*picture element*)



Stockage d'un pixel

- En couleur: 3 composantes — rouge, vert, bleu
  - Chaque composante va de 0 à 255, donc stockage sur 8 bits
  - 24 bits par pixel, 16'777'216 couleurs possibles (2^{24})



- En niveaux de gris, une seule composante
  - 8 bits par pixel

Manipulation d'images en Python

- Python n'est pas spécialement efficace pour traiter de grandes quantités de données avec les types de base (liste, etc.)
  - On paie la flexibilité du langage en mémoire et en temps d'exécution
  - Il nous faut une structure de données qui fournisse des listes (multidimensionnelles) homogènes
- numpy — Numerical Python
  - Fournit des listes rapides
  - Dim. 0: scalaire; dim. 1: vecteur; dim. 2: matrice; 3+: tenseur…
    - ➡ Ou: 2 dimensions: images en niveaux de gris; 3 dimensions: images RGG
- Bibliothèque Pillow, décode et encode les images et les manipule avec numpy
- On utilisera des fonctions auxiliaires qui elles-mêmes utiliseront Pillow et numpy

Démo

Création d'images en niveaux de gris

Module *miniprojectutils.py* à télécharger depuis Moodle; fournit les fonctions faciles d'accès

```
from miniprojectutils import *

img = new_image_grey(10, 10)
print(img)
save_image(img, "black.png")
```

Nouvelle image en niveaux de gris de 10 × 10 pixels

Affiche les valeurs brutes: une matrice de 10 × 10 zéros = 100 pixels noirs

Enregistre ces données au format PNG (sans pertes). Pour le miniprojet: plutôt .jpg (avec pertes, mais fichiers plus petits)

```
[[0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]]
```

```
for row in range(10):
    for col in range(10):
        img[row, col] = (row + 1) * 20
```

Équivalent! Le ":" sélectionne toute la ligne!

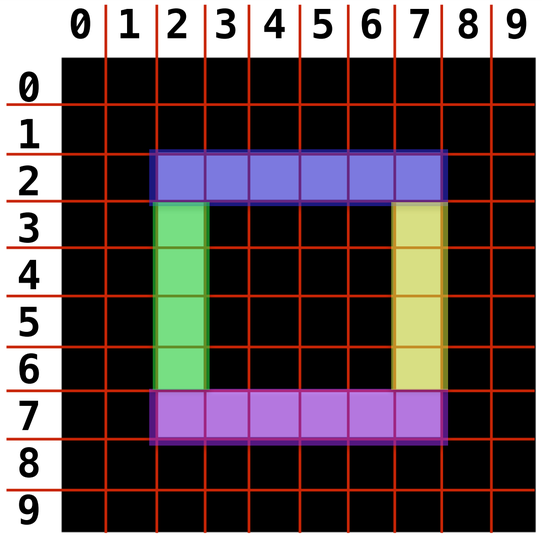
```
for row in range(10):
    img[row, :] = (row + 1) * 20
```

```
print(img)
save_image(img, "gradient.png")
```

```
[[ 20  20  20  20  20  20  20  20  20  20]
 [ 40  40  40  40  40  40  40  40  40  40]
 [ 60  60  60  60  60  60  60  60  60  60]
 [ 80  80  80  80  80  80  80  80  80  80]
 [100 100 100 100 100 100 100 100 100 100]
 [120 120 120 120 120 120 120 120 120 120]
 [140 140 140 140 140 140 140 140 140 140]
 [160 160 160 160 160 160 160 160 160 160]
 [180 180 180 180 180 180 180 180 180 180]
 [200 200 200 200 200 200 200 200 200 200]]
```

Dessiner des formes

Tâche: générer l'image ci-contre ⇒




```
img = new_image_grey(10, 10)

for col in range(2, 8):
    img[2, col] = 255

for row in range(3, 7):
    img[row, 2] = 255
    img[row, 7] = 255

for col in range(2, 8):
    img[7, col] = 255

print(img)
save_image(img, "box.png")
```

Écriture des zones blanches pixel par pixel

Ligne par ligne

```
img[2, 2:8] = 255
img[3:7, 2] = 255
img[3:7, 7] = 255
img[7, 2:8] = 255
```

```
img[[2, 7], 2:8] = 255
img[3:7, [2, 7]] = 255
```

«Zone par zone»

Des images couleur

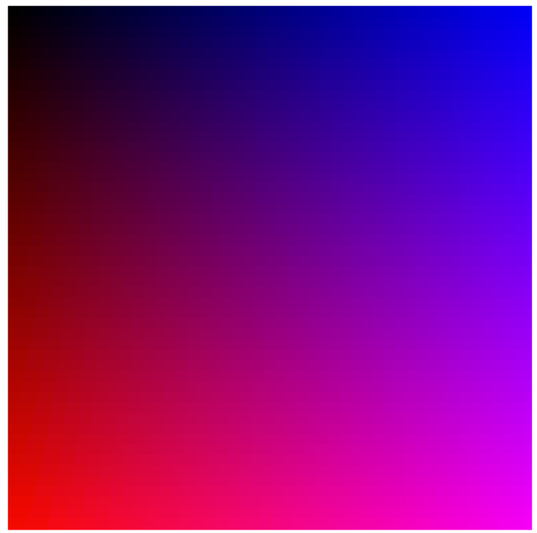
```
img = new_image_rgb(250, 250)

print(img[125, 125])

for row in range(250):
    for col in range(250):
        img[row, col] = [row, 0, col]

save_image(img, "output.png")
```

Image RGB: chaque pixel est maintenant non plus un nombre, mais 3, pour les 3 composantes RGB

Le pixel (125, 125) est noir, représenté ici par la liste de 3 éléments: `[0 0 0]`

Pour modifier un pixel donné, on ne lui affecte pas une seule valeur, mais de nouveau, une liste de 3 composantes

Que génère ce code?



Subscripting dans numpy

- Dans `img[x, y]`, `x` et `y` peuvent être:
  - Une coordonnée sous forme de nombre entier: `0` `10` `i` *etc.*
  - Une plage: `0:10` `5:` `10:` `:` *etc.*
  - Une liste de coordonnées: `[0, 2, 8]` *etc.*
- La lecture d'un pixel via `img[x, y]` donne:
  - Un nombre entier si l'image est en niveaux de gris
  - Une liste de trois nombres si l'image est en RBG
- Dans l'écriture d'un pixel via `img[x, y] = p`, `p` doit être:
  - Un nombre entier si l'image est en niveaux de gris
  - Une liste de trois nombres si l'image est en RBG

Cours de cette semaine

Manipulation d'images

Introduction au miniprojet

Miniprojet: administratif

- **15%** de la note (midterm: 50%, examen final: 35%)
  - Examen final plus court que midterm; pas de question ouverte de programmation
  - Mais toujours un QCM, donc… venez quand même aux cours de programmation!
- Possible de travailler par deux ou tout·e seul·e
  - Groupe à indiquer via lien Moodle qui sera rendu disponible avant la soumission
- Charge horaire estimée: 8 heures
  - 4 heures cette semaine, puis 2 × 2 heures
  - Attention, c'est une **estimation**!

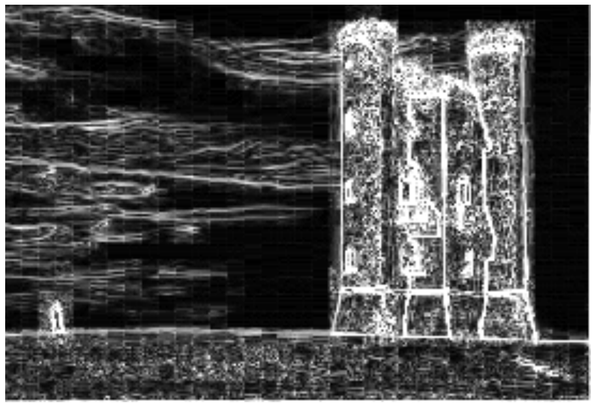
Miniprojet

Selon une idée et projet original de Jamila Sam et Barbara Jobstmann

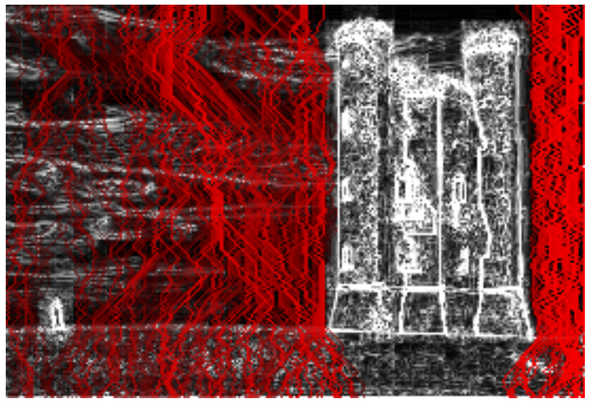
Original

Conversion en niveaux de gris et lissage

Détection des pixels qui sont porteurs de plus ou moins d'information par rapport à leurs voisins avec un filtre Sobel



Détection de tracés de haut en bas qui passent par des pixels de basse information



Suppression de ces tracés dans l'image originale

Résultat

Miniprojet

Déjà dans le code de base sur Moodle

```python
def seam_carving(img_path: str, num_cols: int) -> None:
    name, ext = split_name_ext(img_path)
    folder = name + os.path.sep
    os.makedirs(folder, exist_ok=True)

    img = load_image(img_path)

    img_grey = to_grayscale(img)
    save_image(img_grey, folder + "grey" + ext)

    img_grey = smoothen(img_grey)
    save_image(img_grey, folder + "smooth" + ext)

    img_grey = sobel(img_grey)
    save_image(img_grey, folder + "sobel" + ext)
    img_grey = np.uint8(img_grey)

    for i in range(num_cols):
        seam = find_seam(img_grey)

        img_highlight = highlight_seam(img, seam)
        save_image(img_highlight, folder + f"highlight_{i}" + ext)
        img_grey_highlight = highlight_seam(img_grey, seam)
        save_image(img_grey_highlight, folder + f"highlight_{i}_grey" + ext)

        img = remove_seam(img, seam)
        save_image(img, folder + f"step_{i}" + ext)
        img_grey = remove_seam(img_grey, seam)
```

Création d'un dossier pour stocker le résultat

Chargement de l'image

Conversion en niveaux de gris et sauvegarde intermédiaire

Lissage de l'image (+ sauvegarde)

Application du filtre Sobel (+ sauvegarde)

Boucle qui se répète autant de fois qu'on veut enlever de colonnes

On repère un *seam* — *semaine prochaine*

On enregistre des images intermédiaires où on montre où se trouve le *seam* détecté

On supprime le *seam* de l'image (et aussi de l'image en gris) et on enregistre le résultat, puis on recommence la boucle si nécessaire

Miniprojet: votre travail

```python
def rgb_to_grey(r: int, g: int, b: int) -> int:
    """Convert an RGB color to a greyscale value."""
    return ...  # TODO
```

Convertit un pixel avec ses 3 composantes RGB en un seul niveau de gris (selon formule de la donnée)

```python
def to_grayscale(img: Image) -> Image:
    """Convert the given image to grayscale."""
    print("   Converting to grayscale...")
    return ...  # TODO
```

Convertit une image RGB entière en une image de même taille en niveau de gris

```python
def clamp_index(index: int, length: int) -> int:
    """Return the index, clamped to the range [0, length-1]."""
    return ...  # TODO
```

Retourne un index pas plus petit que 0 et pas plus grand que *length* – 1

```python
def apply_kernel(img_grey: Image, kernel: Kernel) -> Image:
    """Apply a kernel to an image."""
    return ...  # TODO
```

Applique un *kernel* à une image (détails dans la donnée)

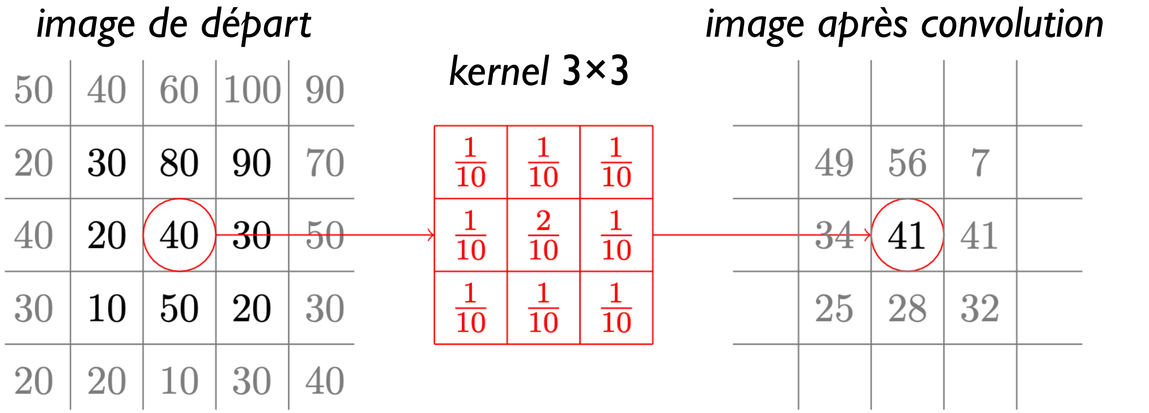
```python
def find_seam(img_grey: Image) -> Seam:
    """Find the seam with the lowest energy."""
    print("   Finding seam...")
    return ... # TODO
```

Cherche un *seam* dans une image (dont on part du principe qu'elle déjà passée par le lissage et le Sobel)

intro théorique la semaine prochaine

Convolution

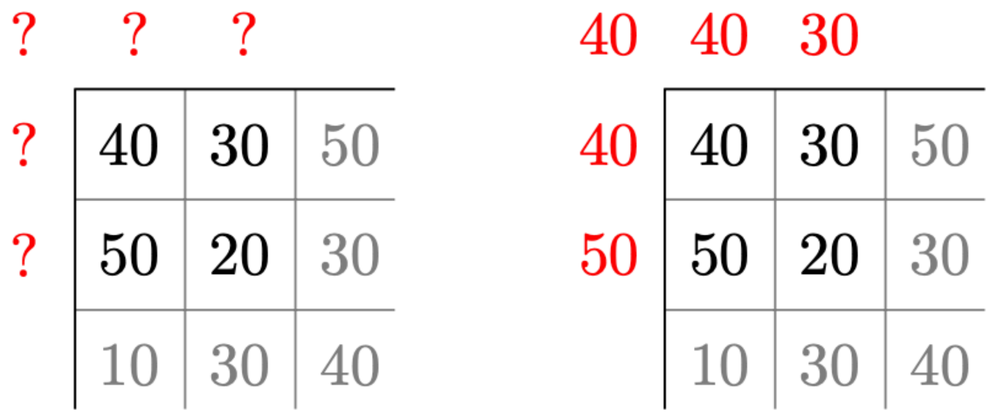
- **`smoothen`** and **`sobel`** ne sont pas à implémenter directement…
  - Car ils sont des cas particulier de **`apply_kernel`**
  - Ceci implémente une opération de **convolution** sur l'image
- La convolution d'une image crée une nouvelle image où chaque pixel est le résultat d'une somme pondérée des pixels voisins
  - La façon de faire cette pondération est déterminée par la **matrice de convolution**, ou **kernel**
  - Exemple du lissage: il s'agit de faire une sorte de «moyenne 2D» des pixels voisins



$$30\tfrac{1}{10} + 80\tfrac{1}{10} + 90\tfrac{1}{10} + 20\tfrac{1}{10} + 40\tfrac{2}{10} + 30\tfrac{1}{10} + 10\tfrac{1}{10} + 50\tfrac{1}{10} + 20\tfrac{1}{10} = 41$$

Que faire dans les bords?

- Pour calculer chaque nouveau pixel, on a besoin des pixels environnants
- Que faire des les bords?
  - On utilise la valeur du pixel le plus proche



Pour calculer le nouveau pixel en position (0, 0),
on simule des pixels de bord supplémentaires lorsqu'on applique le kernel

Kernels

Smoothen

$$\begin{bmatrix} \frac{1}{10} & \frac{1}{10} & \frac{1}{10} \\ \frac{1}{10} & \frac{2}{10} & \frac{1}{10} \\ \frac{1}{10} & \frac{1}{10} & \frac{1}{10} \end{bmatrix}$$

Moyenne des pixels environnants, avec un poids légèrement plus haut pour le pixel central

⇓

Original

Sobel X

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

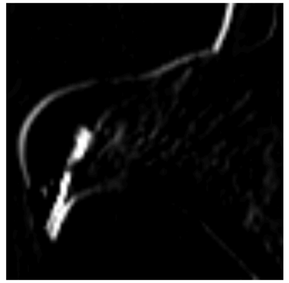
Sobel Y

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

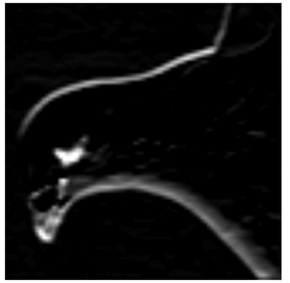
Calcule en quelque sorte une approximation du gradient de l'image, sur l'axe X et sur l'axe Y.

Un gradient de 0 veut dire que les pixels ne changent pas trop le long d'un axe, donc peu d'information. Un haut gradient indique des changements d'intensité des pixels, donc beaucoup d'information.

⇓



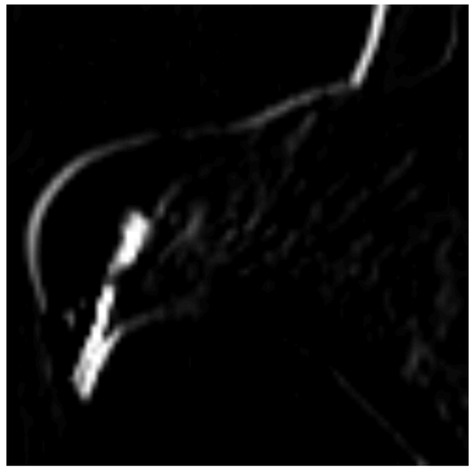
⇓



Sobel, suite

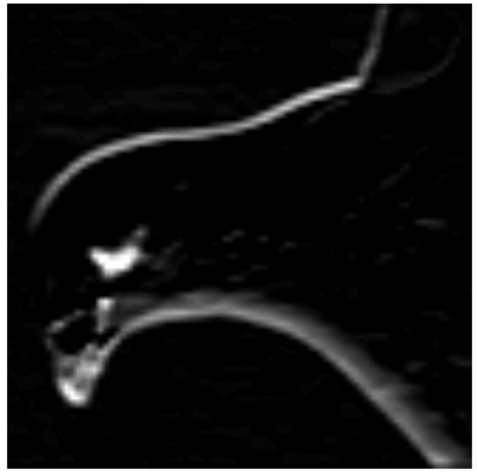
- Comment combiner cette mesure du changement horizontal (Sobel X) et du changement vertical (Sobel Y)?

Composante X
du gradient



x

Composante Y
du gradient



y

⇒

***Norme** du gradient*

$\sqrt{x^2 + y^2}$

(à calculer bien sûr pour chaque pixel)

Implémentation de smoothen

```
def smoothen(img_grey: Image) -> Image:
    """Smooth the image using a 3x3 kernel."""
    print("   Smoothening image...")
    kernel_smooth = np.array([
        [1, 1, 1],
        [1, 2, 1],
        [1, 1, 1],
    ]) / 10

    return apply_kernel(img_grey, kernel_smooth)
```

Définit le kernel de lissage. On utilise un np.array plutôt que des listes «normales» parce que c'est plus efficace

… et parce qu'on peut d'un coup diviser tous les éléments par 10 comme ceci!

Ensuite, le résultat du lissage est «simplement» le résultat de la convolution de l'image en paramètre avec le kernel de lissage

Implémentation de sobel

```
def sobel(img_grey: Image) -> Image:
    """Apply the Sobel filter to the image."""
    print("   Sobel...")
    kernel_sobel_x = np.array([
        [-1, 0, 1],
        [-2, 0, 2],
        [-1, 0, 1],
    ])
    sobel_x = apply_kernel(img_grey, kernel_sobel_x)
    kernel_sobel_y = np.array([
        [-1, -2, -1],
        [ 0,  0,  0],
        [ 1,  2,  1],
    ])
    sobel_y = apply_kernel(img_grey, kernel_sobel_y)
    result = np.sqrt(sobel_x * sobel_x + sobel_y * sobel_y)
    return result
```

On fait la convolution avec Sobel X pour calculer la composante horizontale

On fait la convolution avec Sobel Y pour calculer la composante verticale

On prend la racine de la somme des carrés (opération faite ici d'un coup sur tous les éléments!)

Cours de la semaine prochaine

Algorithme pour trouver le seam de moindre énergie

Résumé Cours 9

- Les images en Python sont (pour notre cas) représentés par structures bidimensionnelles (niveaux de gris) ou tridimensionnelles de numpy
- On peut lire et écrire des pixels individuels, qui sont soit un nombre unique entre 0 et 255 (niveaux de gris), soit trois nombres (RGB)
- Une convolution sur une image génère une nouvelle image selon une matrice de convolution ou kernel
- En lissant une image en niveaux de gris et en prenant la norme du gradient estimé avec les kernels Sobel X et Sobel Y, on obtient une image donnant une bonne idée de l'information générale véhiculée dans chaque pixel