DM4 MP2I : redimensionnement d'images avec OCaml

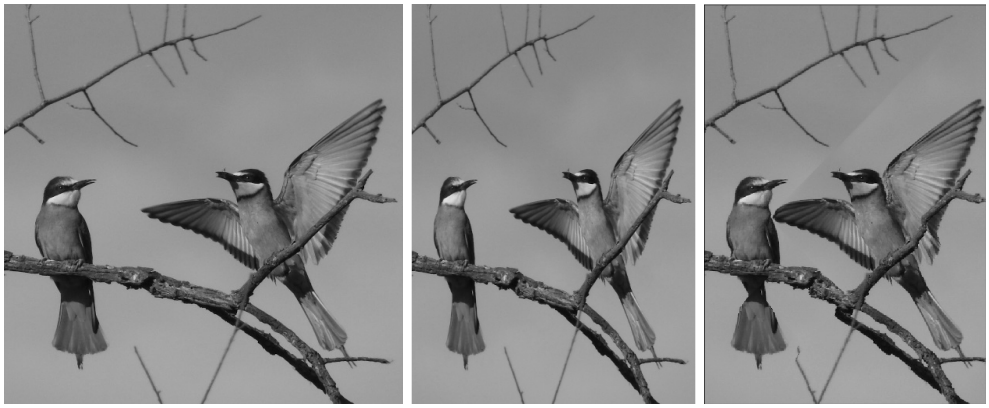

FIGURE 1 – Image originale, redimensionnement naïf, et redimensionnement «intelligent»

Dans ce devoir tiré d'un travail de Mickaël Péchaud, nous implémentons différentes méthodes permettant de **réduire une image** dans le sens de la largeur. Les consignes se trouvent dans le fichier `readme.pdf` de l'archive. Les photos sur lesquelles nous rtavaillons sont aussi dans l'archive.

Une image I est codée sous la forme d'une liste de listes d'entiers, qui peut être vue comme un tableau à deux dimensions. On note h le nombre de lignes de l'image, et w son nombre de colonnes (respectivement pour **h**eight et **w**idth). Le pixel d'indice $(0, 0)$ correspond par convention au pixel situé en haut à gauche de l'image. Chaque pixel de coordonnées $(i, j) \in [\![0, h-1]\!] \times [\![0, w-1]\!]$ a une valeur entière $I_{i,j}$ comprise entre 0 et 255, correspondant à son niveau de gris (0 pour noir, 255 pour blanc).

Prise en main

Q.1 On effectue des changements simples sur l'image :

(a) Écrire la fonction `seuil : int -> img -> img` qui seuille les pixels : tout ce qui est plus grand que s prend la valeur 255, ce qui est plus petit strictement prend la valeur 0 (voir figure 2b) .

(b) Écrire la fonction `symh : img -> img` qui effectue une symétrie horizontale de l'image (voir figure 2c). Il s'agit d'inverser l'ordre des lignes.

(c) Écrire la fonction `symv : img -> img` qui effectue une symétrie verticale de l'image (voir figure 2d). Il s'agit d'inverser l'ordre des colonnes.

FIGURE 2 – La photo des flamants en négatif, seuillée à 128, symétrisée horizontalement puis verticalement

(a) (b)

(c) (d)

Réduction de largeur naïve

Q.2 Écrire la fonction `reduction_moitie_ligne : int array -> int array`, qui prend en argument un tableau `l` de longueur paire $2n$ contenant des entiers $a_0, a_1, a_2, \dots a_{2n-1}$, et renvoie la liste de longueur n contenant $(a_0 + a_1)//2$, $(a_2 + a_3)//2$, etc.

```
# reduction_moitie_ligne [|1;3;4;6|];;
- : int array = [|2; 5|]
```

Q.3 Écrire la fonction `reduction_moitie_image : img -> unit` qui prend en argument une image (ayant une largeur paire) et la modifie en appliquant à chaque ligne l'opération décrite dans la question précédente (la fonction modifie l'image par effet de bord, et ne renvoie rien). Quelle est sa complexité en fonction du nombre de lignes h et du nombre de colonnes w de l'image ?

On observe figure 3 l'application de cette fonction à la matrice décrivant les flamants.

Nous allons maintenant développer des algorithmes plus «intelligents», dans la mesure où ils prennent en compte le contenu de l'image. On commence par déterminer une mesure d'«importance» des pixels - que nous appellons *énergie* dans la suite.

FIGURE 3 – Réduction de moitié de chaque ligne

Calcul de l'énergie d'un pixel

On définit le type

```
type energy = float array array;;
let foi = float_of_int;;
let iof = int_of_float;;
```

Il s'agit d'un simple alias pour les matrices de flottants. Comme il va falloir transformer des entiers en flottants, on s'aquipe avec un alias de `float_of_string` plus concis à écrire. La fonction `foi` réalise cette opération et `iof`, l'opération inverse.

Si l'on note $I_{i,j}$ le niveau de gris du pixel de coordonnées $(i,j) \in [\![0, h-1]\!] \times [\![0, w-1]\!]$, l'*énergie* d'un pixel i,j intérieur à l'image est définie comme

$$e_{i,j} = \left|\frac{I_{i+1,j} - I_{i-1,j}}{2}\right| + \left|\frac{I_{i,j+1} - I_{i,j-1}}{2}\right|.$$

Cette quantité, non entière, est d'autant plus petite que le pixel est dans une zone uniforme de l'image, et d'autant plus grande qu'il est dans une zone de forte variation du niveau de gris (par exemple au niveau d'un contour) [1].

Q.4 Écrire la fonction `energie : img -> f`, qui prend en argument une image, et renvoie une nouvelle image correspondant à son énergie (les pixels d'énergie sont donc des flottants).

Il faut en particulier adapter la formule ci-dessus à un pixel situé sur un bord de l'image : on remplace dans la formule le ou les voisins manquants par le pixel courant.

Bien que pas très difficile à comprendre, il s'agit de la fonction au code le plus long du devoir.

1. Il s'agit en fait d'une version discrétisée de $\|\nabla I\|_1 = \left|\frac{\partial I}{\partial x}\right| + \left|\frac{\partial I}{\partial y}\right|$, qui est une norme du gradient de l'image.

```
# let img = [|[|192; 157; 86; 25; 215; 44; 40|];
              [|236; 17; 226; 209; 69; 205; 118|];
              [|159; 92; 65; 238; 84; 165; 121|];
              [|121; 62; 15; 107; 32; 240; 131|];
              [|52; 194; 129; 60; 19; 67; 229|];
              [|253; 169; 92; 39; 195; 240; 143|];
              [|111; 2; 84; 251; 20; 90; 186|]|]
in
energie img;;
                    - : energy =
[|[|39.5; 123.; 136.; 156.5; 82.5; 168.; 41.|];
  [|126.; 37.5; 106.5; 185.; 67.5; 85.; 84.|];
  [|91.; 69.5; 178.5; 60.5; 55.; 36.; 28.5|];
  [|83.; 104.; 54.5; 97.5; 99.; 98.5; 108.5|];
  [|137.; 92.; 105.5; 89.; 85.; 105.; 87.|];
  [|71.5; 176.5; 87.5; 147.; 101.; 37.5; 70.|];
  [|125.5; 97.; 128.5; 138.; 168.; 158.; 69.5|]|]
```

Quelle est la complexité de votre fonction ?

Q.5 Pour afficher une image d'énergie, il faut au préalable trasnformer ses coefficients en entiers. Écrire la fonction `energie_to_image : energy -> img` qui réalise cela.

```
# let img = let img = Array.make_matrix 3 3 0 in
  for i = 0 to 2 do
    for j= 0 to 2 do
      img.(i).(j)<-Random.int 256;
    done;
  done;
  img;;
              val img : int array array =
  [|[|184; 116; 81|]; [|113; 31; 213|]; [|7; 53; 116|]|]
```

Vous pouvez alors afficher l'image d'énergie des guêpiers et vérifier que l'énergie est très utile pour délimiter les contours.

Image originale, image des énergies

Réduction ligne par ligne

Maintenant que nous disposons d'une image et de son énergie, pour réduire la largeur d'une image d'un pixel, nous allons simplement enlever un pixel d'énergie minimale dans chaque ligne.

Nous avons pour cela besoin de quelques fonctions élémentaires de manipulations de listes.

Q.6 Écrire la fonction `enlever : int array -> int -> int array` qui prend en argument un tableau liste `l` et un indice `i` compris entre 0 et $|l|-1$, et renvoie une nouveau tableau correspondant à `l` dont l'élément d'indice `i` a été supprimé.

```
# enlever [|10;20;30;40|] 2;;
- : int array = [|10; 20; 40|]
```

Q.7 Écrire la fonction `indice_min : 'a array -> int`, qui prend en argument un tableau et renvoie un indice auquel apparaît la valeur minimale du tableau.

```
# indice_min [|100;23;89;12;89|];;
- : int = 3
```

Q.8 Écrire la fonction `reduction_ligne_par_ligne : img -> unit`, qui prend en arguments une image `img`, et la modifie en supprimant un pixel d'énergie minimale dans chacune de ses lignes (il y a des effets de bord).

Q.9 (a) Écrire la fonction `itere_reduction_ligne_par_ligne : img -> int -> unit`, qui prenant en arguments une image `img` et un entier $n > 0$, applique n fois la procédure décrite dans la question précédente à `img`.

Avec la photo des flamants et 200 lignes supprimées, le temps pris pour calculer cette compression est de 10s sur ma machine.

(b) Quelle est sa complexité ?

(c) Sur la figure 4, on observe le résultat de cette fonction appliquée à la photo des flamants et $n = 200$: il est peu satisfaisant sauf si on aime la bouillie de flamant.

Expliquez qualitativement, de façon brève, les distortions observées (en commentaire dans votre code).

Réduction par colonne

Pour y remédier, on souhaite lors d'une itération enlever uniquement des pixels situés sur une même colonne.

Q.10 Écrire la fonction `meilleure_colonne : energy -> int`, qui prend en argument une matrice d'énergies `e`, et renvoie un indice de colonne d'énergie minimale (l'énergie d'une colonne étant définie comme la somme des énergies de ses pixels).

Par exemple, la meilleure colonne pour les flamants est la 448.

Q.11 Écrire la fonction `reduction_meilleure_colonne : img -> unit`, qui prend en argument une image, et en supprime une colonne d'énergie minimale (la 1ere s'il y en a plusieurs). Elle réalise des effets de bord.

FIGURE 4 – réduction de 200 lignes

Q.12 En déduire une fonction `itere_reduction_meilleure_colonne : img -> int -> unit`, qui applique la méthode décrite ci dessus pour réduire une image de `n` pixels dans sa largeur. Quelle est sa complexité ?

Sur ma machine, avec 200 colonnes supprimées, la durée d'exécution est inférieure à 10s.

Q.13 On constate que les résultats obtenus sont «bons» sur les images `guepiersNB.jpg` et `flamantsNB.jpg`, mais pas sur `rueNB.jpg` (voir figures 5a, 5b, 5c). Expliquez qualitativement pourquoi en commentaire dans votre code.

FIGURE 5 – La réduction des 100 meilleures colonnes appliquées aux flamants, aux guêpiers et à la rue

(a)

(b)

(c)

Seam carving

L'idée de l'algorithme de *Seam carving*, introduit en 2007 par S.Avidan et A.Shamir est d'assouplir un peu la contrainte de réduction colonne par colonne.

On définit un *chemin* de pixels comme une suite de pixels connectés (verticalement ou en diagonale) dont le premier appartient au bord haut de l'image, le dernier au bord bas, et contenant exactement un pixel par ligne de l'image.

L'énergie d'un chemin est la somme des énergies des pixels le constituant.

Voici par exemple un chemin d'énergie 6 dans une image des énergies jouet e de 4×4 pixels.

1	1	0	3
4	1	2	4
1	2	2	1
4	1	1	0

Pour réduire la largeur d'une image d'un pixel, on souhaite trouver puis enlever un chemin d'énergie minimale.

On définit un *chemin partiel* comme un chemin, sans la contrainte que son dernier pixel atteigne le bord bas de l'image. Afin de calculer un chemin minimal, on va commencer par calculer, pour chaque pixel, l'énergie minimale d'un chemin partiel terminant sur ce pixel.

Par exemple, pour notre image des énergies e, on obtient le *tableau des énergies minimales* E suivant :

1	1	0	3
5	1	2	4
2	3	3	3
6	3	4	3

Q.14 Calculer à la main le tableau des énergies minimales E pour la matrice d'énergies e suivante, puis trouver un chemin d'énergie minimale.

2	1	1	0
3	3	2	2
2	0	1	2

Q.15 Expliquez comment construire E à partir de e, en procédant ligne par ligne. Donner la réponse en commentaire dans le code.

Q.16 Une fois E construit, comment en déduire un chemin d'énergie minimale ? Donner la réponse en commentaire dans le code.

Une fois pour toute dans ce qui suit, on suppose que `img` est de dimension $w \times h$ (donc `e` et `E` aussi).

Q.17 Écrire la fonction `ajouter_ligne : energy -> energy -> int -> unit` telle que `ajouter_ligne e em i` renseigne la ligne i de la matrice des énergies minimales `em` en utilisant les lignes déjà renseignées de `em` et la matrice des énergies `e`. La fonction fait donc des effets de bord.

Donner la complexité.

```
# let e = [|[|1.;1.;0.;3.|];[|4.;1.;2.;4.|];
[|1.;2.;2.;1.|];[|4.;1.;1.;0.|]|] and
em = [|[|1.;1.;0.;3.|];[||];[||];[||]|] in
ajouter_ligne e em 1; em;;
    - : float array array =
[|[|1.; 1.; 0.; 3.|]; [|5.; 1.; 2.; 4.|]; [||]; [||]|]
```

Q.18 Écrire la fonction `construire_E : energy -> energy` qui renvoie le tableau des énergies minimale E à partir d ela matrice des énergies e.

```
# let e = [|[|1.;1.;0.;3.|];
           [|4.;1.;2.;4.|];
           [|1.;2.;2.;1.|];
           [|4.;1.;1.;0.|]|] in
construire_E e;;
        - : energy =
[|[|1.; 1.; 0.; 3.|]; [|5.; 1.; 2.; 4.|];
  [|2.; 3.; 3.; 3.|]; [|6.; 3.; 4.; 3.|]|]
```

Donner la complexité de `construire_E`.

Q.19 Écrire la fonction `chemin_minimal : energy -> int array array` qui calcule un chemin d'énergie minimale à partir du tableau des énergies minimales.

Le chemin est donné comme la liste des coordonnées des pixels en commençant par celui du haut.

```
# let em = [|[|1.; 1.; 0.; 3.|];
            [|5.; 1.; 2.; 4.|];
            [|2.; 3.; 3.; 3.|];
            [|6.; 3.; 4.; 3.|]|]
in chemin_minimal em;;
        - : int array array = [|[|0; 2|]; [|1; 1|]; [|2; 0|]; [|3; 1|]|]
```

Donner la complexité.

Q.20 Écrire la fonction `reduction_meilleure_energie : img -> unit` qui prend en paramètre une image, calcule un chemin d'energie minimale dans cette image et retire dans l'image les pixels de ce chemin. La fonction réalise des effets de bord.

La complexité de la fonction doit être en $O(hw)$.

Q.21 Écrire la fonction `itere_reduction_meilleure_energie : img -> int -> unit` qui itère la fonction précédente n fois.

Donner la complexité.

L'image obtenue par cette fonction appliquée aux guêpiers avec $n = 100$ est la 3ème photo sur la droite tout en haut de ce devoir. Le calcul prend 8s sur ma machine (qui a 6 ans).