Practicum Beeldverwerking

Computationeel Denken

In dit practicum gaan jullie beelden verwerken. Eigenlijk zijn foto's niets meer dan grote twee-dimensionele matrices van gekleurde puntjes. De kleur van zo'n puntje wordt door een computer voorgesteld door niets anders dan gehele getallen die de hoeveelheid groen, blauw en rood van die kleur voorstellen. Zo zal zwart weergegeven worden door (0,0,0) en wit door (255,255,255).

Het programma dat ontwikkeld moet worden bestaat uit drie delen. De template is te vinden in bijlage 1:

- **Inlezen van de figuur.** Je kan in de code de naam van een figuur (.jpg) ingeven die je wilt manipuleren, bijvoorbeeld "Paradise.jpg". Zorg dat de figuur die je wilt aanpassen in de juist map is toegevoegd. Vervolgens wordt de figuur omgezet in drie matrices waarin respectievelijk de roodwaarden, groenwaarden en blauwwaarden worden opgeslagen. Vervolgens wordt het originele beeld ook nog getoond op een apart scherm:

```
r_in # bevatten de oorspronkelijke roodwaarden
g_in # bevatten de oorspronkelijke groenwaarden
b_in # bevatten de oorspronkelijke blauwwaarden
```

- **Verwerken of bewerken van de figuur.** Dit gebeurt door jullie. De RGB waarden worden tijdens de manipulatie van het beeld veranderd, en opgeslagen in nieuwe variabelen:

```
r_out = np.zeros((r_in.shape[0], r_in.shape[1]))
g_out = np.zeros((g_in.shape[0], g_in.shape[1]))
b_out = np.zeros((b_in.shape[0], g_in.shape[1]))
```

- **Tonen van de output op het scherm**. Dit gebeurt door het volgende stuk code achteraan het programma toe te voegen:

```
r_image_out = Image.fromarray(np.uint8(r_out))
g_image_out = Image.fromarray(np.uint8(g_out))
b_image_out = Image.fromarray(np.uint8(b_out))
output_im = Image.merge("RGB", (r_image_out, g_image_out, b_image_out))
output_im.show()
```

Nu worden 11 bewerkingen op beelden voorgesteld. De bedoeling is deze te implementeren. Bij de start van een nieuwe bewerking kan de oude voorlopig in commentaar worden geplaatst.

Optie 1: een kleur wegfilteren.

Een kleur kan weg gefilterd worden door de kleurwaarde op alle posities in het beeld op 0 te plaatsen.

Optie 2: negatief van een beeld nemen.

Dit gebeurt door alle RGB waarden op alle posities te vervangen door 255 min de oorspronkelijke RGB waarde.

Optie 3: een zwart/wit beeld maken.

Dit kan gebeuren door donkere pixels helemaal zwart (0,0,0) te maken en lichte pixels helemaal wit (255,255,255) te maken. Bedenk zelf een mechanisme om te bepalen of het om een donkere of lichte pixel gaat.

Optie 4: een beeld met enkel grijswaarden maken.

Dit bekomt men door de drie RGB waarden op eenzelfde positie in de matrix gelijk te stellen aan het gemiddelde van de oorspronkelijke waarden op die positie.

Optie 5: een beeld verduisteren of verhelderen.

Dit gebeurt door elke pixel te vermenigvuldigen met een double waarde. Indien de double waarde groter is dan 1 dan wordt het beeld lichter. Indien de double waarde kleiner is dan 1, dan wordt het beeld donkerder. Maak voor mooie effecten de factor niet te groot of te klein. Probeer dit ook eens meerdere malen na elkaar.

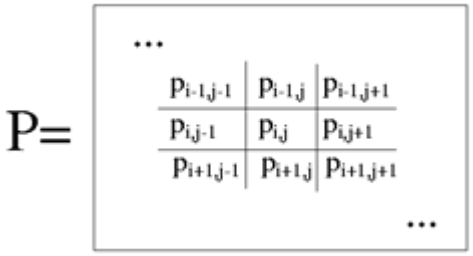
Optie 6: een beeld horizontaal of verticaal spiegelen.

Dit gebeurt door pixels van positie te veranderen.

Optie 7: een beeld roteren.

Dit gebeurt door pixels van positie te veranderen. Merk op dat de dimensies van het nieuwe beeld verschillen van deze van het oorspronkelijke beeld.

Optie 8: een beeld vervagen.



Stel: P is de matrix die het beeld voorstelt. De nieuwe kleurwaarden van een pixel (i.e. elk van de drie getallen die in een pixel worden bijgehouden) zullen berekend worden aan de hand van de kleurwaarden van de huidige pixel en de omliggende pixels (i.e. de pixels rondom het beschouwde pixel). Om een beeld te vervagen wordt het gemiddelde van de waarden van deze 9 pixels genomen.

Let wel op met de Pixels aan de rand van het beeld. Deze kunnen immers niet met de rondliggende pixels gecombineerd worden. Voor de eenvoudigheid mag je die pixel gewoon onveranderd laten.

Optie 9: een beeld herschalen.

Reduceer het beeld naar halve dimensies in de horizontale en verticale richting. De RGB waarden van elke nieuwe pixel wordt nu telkens het gemiddelde van de RGB waarden van 4 pixels uit het oorspronkelijke beeld.

Optie 10: een groene rand van 5 pixels breed rond de figuur.

De afmetingen van het beeld vergroten. Vervolgens worden de randen opgevuld, en tenslotte wordt het beeld gekopieerd van de oorspronkelijke matrices naar de nieuwe.

Optie 11: edge detection.

Het detecteren van randen in een foto is een andere toepassing van de transformatiematrix. De Sobel-matrix - die uiterst geschikt is voor die doeleinden - is gegeven door :

$$S = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

Deze matrix is vooral geschikt om verticale randen te detecteren binnen een foto. (probeer maar eens). De getransponeerde matrix S' wordt gebruikt om horizontale randen te detecteren. Willen we nu alle randen van een beeld ontdekken moeten we beide combineren op de volgende manier: Stel $vert\_a_{i,j}$ is het resultaat van het toepassen van de verticale Sobel-matrix S voor de kleurwaarde $a_{i,j}$, en $hor\_a_{i,j}$ het resultaat van de horizontale Sobel-matrix S'. De nieuwe (gecombineerde) waarde wordt dan $new\_a_{i,j} = \sqrt{vert\_a_{i,j}^{\,2} + hor\_a_{i,j}^{\,2}}$.

Als je wil kan je nog een sterkere scheiding bekomen door een grens in te bouwen die zal beslissen of de nieuwe (berekende) kleur van een pixel naar wit of zwart wordt omgezet.

Intermezzo – Edge Detection en de Sobel Matrix. Bij edge detection wordt de roodwaarde, groenwaarde en blauwwaarde van een pixel op positie [i][j] telkens bepaald door de optelling van 9 termen op basis van de kleurwaarden van de omliggende pixels (zoals dat eigenlijk het geval is bij het vervagen van een beeld). De Sobel matrix S duidt aan in welke mate de termen deel uitmaken van de optelling. De Sobel Matrix ziet er als volgt uit:

-1 0 1
-2 0 2
-1 0 1

Bijgevolg wordt de optelling als volgt gedaan voor het berekenen van de nieuwe roodwaarde:

-1 * ouderoodwaarde[i-1][j-1] + 0 * ouderoodwaarde[i-1][j] + 1 * ouderoodwaarde[i-1][j+1]
-2 * ouderoodwaarde[i][j-1] + 0 * ouderoodwaarde[i][j] + 2 * ouderoodwaarde[i][j+1]
-1 * ouderoodwaarde[i+1][j-1] + 0 * ouderoodwaarde[i+1][j] + 1 * ouderoodwaarde[i+1][j+1]

Het proces verloopt analoog voor het berekenen van nieuwe blauw- en groenwaardes.

Eigenlijk is het vervagen van een beeld analoog. De matrix waarmee de convolutie plaatsvindt ziet er dan echter anders uit:
1/9 1/9 1/9
1/9 1/9 1/9
1/9 1/9 1/9

Uitbreidingen.

Herwerk zodanig dat driedimensionale arrays worden gebruikt. Voor de tweedimensionale arrays zit de template in bijlage 1. Voor de driedimensionale arrays zit de template in bijlage 2.

Breid het programma uit met een menu. In plaats van voor elke bewerking een nieuw programma te maken kan de gebruiker telkens een getal ingeven waardoor vervolgens een functie wordt opgeroepen die het beeld manipuleert. Voorzie een aparte methode voor elke beeldmanipulatie (11 verschillende methodes dus!). Deze uitbreiding kan je pas maken indien de theorie rond methodes is uitgelegd. Dit is na hoorcollege 14. Bijlage 2 kan gebruikt worden als template voor de code voor drie manipulaties. Natuurlijk moet je de methodes `filterKleurWeg(…,…,…)` `neemNegatief(…,…,…)` en `maakZwartWit(…,…,…)` `in dat geval` zelf nog implementeren.

Bijlage 1: Template ter ondersteuning van het practicum (tweedimensionale arrays).

```
# importeer libraries
from PIL import Image
import math
import numpy as np

# converteer een ingelezen afbeelding naar drie matrices
input_image = Image.open("landschap1.jpg")
r_image_in, g_image_in, b_image_in = input_im.split()
r_in = np.uint32(np.array(r_image_in))
g_in = np.uint32(np.array(g_image_in))
b_in = np.uint32(np.array(b_image_in))

# hier komen de matrixbewerkingen !!!
r_out = np.zeros((r_in.shape[0], r_in.shape[1]))
g_out = np.zeros((g_in.shape[0], g_in.shape[1]))
b_out = np.zeros((b_in.shape[0], g_in.shape[1]))

for i in range(r_in.shape[0]):
    for j in range(r_in.shape[1]):
        r_out[i][j] = r_in[i][j]
        g_out[i][j] = g_in[i][j]
        b_out[i][j] = b_in[i][j]

# converteer drie matrices terug naar een afbeelding
r_image_out = Image.fromarray(np.uint8(r_out))
g_image_out = Image.fromarray(np.uint8(g_out))
b_image_out = Image.fromarray(np.uint8(b_out))
output_im = Image.merge("RGB", (r_image_out, g_image_out, b_image_out))
output_im.show()
```

Bijlage 2: Template ter ondersteuning van het practicum (driedimensionale arrays).

```
# importeer libraries
from PIL import Image
import math
import numpy as np

# converteer een ingelezen afbeelding naar drie matrices
input_image = Image.open("./beeld/landschap1.jpg")
complete_in = np.uint32(np.array(input_image))

# hier komen de matrixbewerkingen
complete_out = np.zeros((complete_in.shape[0], complete_in.shape[1], 3))
for i in range(complete_out.shape[0]):
    for j in range(complete_out.shape[1]):
        complete_out[i][j] = complete_in[i][j]

# converteer drie matrices terug naar een afbeelding
image_out = Image.fromarray(np.uint8(complete_out))
image_out.show()
```