Labo 1: Inleiding tot Python en basisbewerkingen met beelden

Gianni Allebosch, Martin Dimitrievski

Het eerste labo Multimedia omvat een inleiding tot beeldverwerking in Python. De focus ligt op de voorstelling van digitale beelden binnen deze programmeeromgeving en op enkele basisbewerkingen. Hieronder vind je een aantal interessante informatiebronnen:

`https://docs.python.org`
`https://opencv.org/`
`https://docs.opencv.org/master/d6/d00/tutorial_py_root.html`
`https://numpy.org/`

1 Python en Conda

Python is een hoogniveau programmeertaal. Dit betekent dat een deel van de taken van een programmeur (in meer 'klassieke' talen als C / C++ en Java) hier verschoven werden naar een lager niveau (de *interpreter*). Een aantal voordelen hiervan zijn bijvoorbeeld:

- Bij het initialiseren van een object of een variabele hoeft (meestal) geen type gespecificeerd te worden.
- De interne datastructuren zijn zeer flexibel. Zo kunnen er verschillende types gecombineerd worden (bv. strings en integers) en kunnen de interne elementen makkelijk aangesproken worden. Meer info hierover vind je op: `https://docs.python.org/3/tutorial/datastructures.html#` .
- Complexe bewerkingen voor grote datahoeveelheden (bijvoorbeeld matrixberekeningen) kunnen zeer compact neergeschreven worden door gebruik te maken van gespecialiseerde packages zoals *Numpy*. Dit is bijzonder handig voor het verwerken van beelden, zoals tijdens de uitvoering van dit labo duidelijk zal worden.

Om mogelijke compatibiliteitsproblemen te vermijden, vragen we dat jullie in dit labo werken in een specifieke *virtual environment* via Conda, de package manager van Anaconda (`https://anaconda.cloud/`). In zo'n virtual environment zijn alle Pythonpackages geïsoleerd van de rest van je systeem. Deze manier van werken heeft meerdere voordelen:

- Bij het installeren van nieuwe packages in hoef je geen rekening te houden met eerdere (of toekomstige) projecten die mogelijk andere versies nodig hebben.
- Conda controleert zelf of er mogelijke conflicten zijn tussen de packages binnen dezelfde environment, op het moment dat je een package installeert.

– Bij het werken in groep kan je ervoor zorgen dat jullie zeker met dezelfde versies van packages / dependencies werken aan dezelfde code.
– Wij kunnen jullie code runnen met dezelfde dependencies voor de beoordeling.

1.1 Richtlijnen virtual environment

Om deze virtual environment op te zetten, volg je de volgende stappen.

1. (Mensen die al eerder Miniconda of de volledige versie van Anaconda hebben geïnstalleerd, kunnen deze stap overslaan).
   Download en installeer **Miniconda**. Volg hiervoor de instructies vanop `https://docs.conda.io/projects/miniconda/en/latest/miniconda-install.html`. We verkiezen zelf Miniconda boven de volledige versie, omdat je op die manier enkel de packages zal installeren die nodig / nuttig zijn en zo geheugen kan besparen. (zie `https://docs.conda.io/projects/conda/en/stable/user-guide/install/download.html#anaconda-or-miniconda`)
   Let op: zorg ervoor dat je miniconda installeert met rechten als **administrator**. Op Windows doe je dit bijvoorbeeld met een rechtermuisklik en `'Run as administrator'` op het installatiebestand.
2. Maak nu de Conda environment met naam `labo_multimedia` aan met de meest recente Pythonversie (op het moment van schrijven, versie 3.11) en activeer deze. Open hiervoor de Anaconda Prompt (Windows) of een terminal (Linux of Mac) en geef in:

   ```
   conda create -n labo_multimedia python=3.11
   conda activate labo_multimedia
   ```

   Geef 'y' in waar nodig.
3. Installeer nu de nodige packages - opencv, numpy, scipy en matplotlib - voor het labo in deze omgeving:

   ```
   pip install opencv-python
   conda install -c conda-forge numpy matplotlib scipy
   ```

   Merk op dat we de packages halen uit de zogenaamde *conda-forge*, een kanaal dat de meest recente **geteste** versies van deze packages bevat. Voor OpenCV zelf gebruiken we pip, omdat conda zelf standaard een *headless* versie installeert (zonder ondersteuning voor bijvoorbeeld `imshow` op Windows).

In principe heb je met deze packages voldoende om de labo-opdrachten rond beeldverwerking te kunnen uitvoeren. Indien nodig kan je altijd extra packages installeren (bijvoorbeeld tijdens het project) of desnoods een nieuwe omgeving aanmaken. In de volgende sectie focussen we specifiek op het gebruik van OpenCV.

1.2 OpenCV

OpenCV (Open Computer Vision) is een *open source* softwarebibliotheek die implementaties van veelgebruikte beeld- en videoverwerkingsalgoritmen bevat. We zullen deze functies gebruiken in elk labo en gedurende het project. Er is ondersteuning voor zowel C++ als Python. In Python wordt OpenCV geïmporteerd als **cv2**.

Test daarom eerst of de installate volledig gelukt is: binnen jouw Conda environment, start python en geef dan volgende commando's in:

```
import cv2
print (cv2.__version__)
```

Het resultaat zou iets moeten zijn zoals:

```
4.8.1
```

1.3 IDE

Om het programmeren zelf te vergemakkelijken, maak je best gebruik van een IDE (*Integrated Development Environment*). Veelgebruikte IDE's voor Python zijn bijvoorbeeld PyCharm en Visual Studio (enkel Windows). Wij verkiezen zelf PyCharm, vanwege de goede ondersteuning voor het werken met Conda environments, *code completion* en *reformatting*.

Je kunt PyCharm downloaden via `https://www.jetbrains.com/pycharm/`. De 'Community Edition' is gratis. Als de Conda environment al geïnstalleerd is, kan je deze in PyCharm koppelen aan jouw project via

```
File > Settings... > Project: <naam XYZ> > Python Interpreter
    > Add Interpreter > Add local interpreter...
    > Conda Environment > Use existing environment
```

Daar zou je de omgeving `labo_multimedia` moeten kunnen terugvinden.

1.4 Template voor Pythonscript

Gebruik bij voorkeur onderstaande structuur voor elke file die een hoofdfunctie (de *main*) bevat:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
# other imports here...

# functions here ...


def main():
```

```
    # main code here...


if __name__ == '__main__':
    main()
```

Er zijn verschillende goeie redenen om **altijd** deze opbouw te gebruiken, vooral eens de programma's complexer worden. Deze YouTube-clip van James Murphy (kanaal *mCoding*) legt zeer helder uit waarom:
`https://www.youtube.com/watch?v=g_wlZ9IhbTs`

2 Een eerste programma (hello image)

Fig. 1 toont een grijswaarden- en een kleurenbeeld; download ze via Ufora en sla ze op in een lokale *work directory*. Een beeld kan ingelezen en getoond worden op verschillende manieren met verschillende packages. Enkele voorbeelden vind je hieronder. Een script wordt opgeslagen in een .py file en uitgevoerd met het volgende commando in een shell:

```
python filename.py
```

Je kunt een programma meestal ook rechtstreeks runnen vanuit de IDE. Let op dat je de juiste environment hebt geactiveerd. Hieronder kan je een eerste voorbeeldprogramma voor binnen de main terugvinden:

```
    image = cv2.imread('lena_color.jpg')
    cv2.imshow('Image',image)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

of

```
    image = plt.imread('lena_color.jpg')
    plt.imshow(image)
    plt.show()
```

Een beeld wordt voorgesteld als een Numpy-matrix van pixelwaarden, met de oorsprong linksboven. Wanneer een grijswaardenbeeld ingelezen wordt, wordt dat voorgesteld als een 2D-matrix met pixelwaarden van het type **uint8** (*unsigned integer byte*), waarmee voor elke pixel een intensiteitswaarde tussen 0 en 255 voorgesteld kan worden. De representatie van kleurenbeelden verschilt van grijswaardenbeelden, het zijn 3D-matrices waarin 3 kleurlagen voorkomen: Rood, Groen en Blauw, en worden daarom ook RGB-beelden genoemd.

Vraag 1: ***In welke volgorde worden de kleurkanalen ingelezen bij de functie*** `cv2.imread` ***en bij de functie*** `plt.imread`***? Wat gebeurt er indien je de afbeelding inleest met OpenCV en daaarna afbeeldt met pyplot? Geef de antwoorden in je README.txt.***

(a)

(b)

Figuur 1. Een grijswaardenbeeld, respectievelijk kleurenbeeld; (a) de cameraman (b) Lena.

Om deze matrices verder te bewerken, kan je gebruik maken van de ingebouwde functionaliteit van Numpy en OpenCV. Een overzicht van een aantal veelgebruikte functies kan je terugvinden via volgende links:

`https://docs.opencv.org/master/d7/d16/tutorial_py_table_of_contents_core.html`

`https://docs.opencv.org/master/d2/de8/group__core__array.html`

`https://docs.scipy.org/doc/numpy/user/quickstart.html`

`https://docs.scipy.org/doc/numpy/reference/`

(a) (b) (c)

Figuur 2. Demonstratie van verschillende spiegelingen; (a) input (b) horizontale + verticale spiegeling (c) diagnonale spiegeling.

3 Opdrachten

1. Schrijf twee Pythonfuncties die de afbeelding `woman_baby.jpg` gespiegeld tonen. De functies moeten geen parameters teruggeven, maar tonen wel het gespiegelde beeld. Na het oproepen van de functie moet het programma vragen welke spiegeling moet gebeuren. Gebruik hiervoor de ingebouwde functie `input` waarbij volgende opties zijn toegestaan:
   - H: horizontale spiegeling;
   - V: verticale spiegeling;
   - D: diagonale spiegeling van rechtsboven naar linksonder;.

     Let op: dit is niet hetzelfde als een opeenvolging van een horizontale en verticale spiegeling (zie Fig. 2)!

   Er zijn verschillende oplossingen mogelijk. Programmeer één versie (`mirror1`) die gebruik maakt van ingebouwde OpenCV of Numpy-functies. De tweede versie (`mirror2`) voorziet de gevraagde functionaliteit via *indexing* en *slicing* (zie `https://numpy.org/doc/stable/reference/arrays.indexing.html`) en waar nodig de Numpy-functie `zeros` of `empty` om een lege matrix met een vooraf ingestelde grootte aan te maken.
   Tip: om de grootte van een ingeladen afbeelding op te vragen, kan je de functie `np.shape` gebruiken. Deze mag je ook gebruiken in `mirror2`.

2. Schrijf een Pythonfunctie `swap` die de afbeelding `cameraman.jpg` eerst opsplitst in 4 verschillende stukken met gelijke grootte. Het stuk linksboven wordt omgewisseld met het stuk rechtsonder, en het stuk rechtsboven met het stuk linksonder (zie Fig. 3 voor een voorbeeld).

3. Schrijf een Pythonfunctie `logicimage` die voor het beeld `Xraywrist.jpg` in Fig. 4 enkel de botten teruggeeft. Het resultaat moet een zwart-witbeeld zijn (zwart = 0, wit = 1) waarin enkel de botten als wit worden weergegeven. Beeld het histogram voor dit beeld af, dat voor elke intensiteitswaarde

(a)

(b)

Figuur 3. Demonstratie van de functie `swap`; (a) input (b) output.

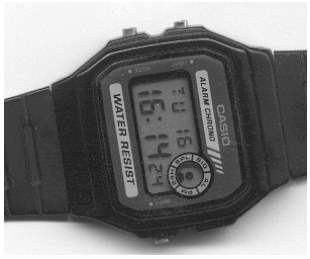
Figuur 4. Afbeelding van een röntgenfoto.

het aantal pixels met die intensiteit weergeeft. (**Tip:** gebruik hiervoor de ingebouwde functies in OpenCV, pyplot of Numpy)
Vraag 2: ***Beschrijf, in je README.txt, hoe je automatisch de pixels met de hoogste intensiteiten (hier de botten) van de lagere zou kunnen scheiden voor dit soort röntgenbeelden aan de hand van het histogram. Hoe kies je hierbij automatisch de drempelwaarde? Beschrijf grondig hoe je dit aanpakt voor dit voorbeeld.***

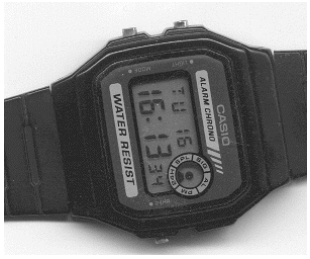
4. **Debug**: hieronder vind je een Pythonscript met de functie `change` die het verschil teruggeeft tussen twee beelden (zie Fig. 5). We willen dat deze functie een verschilbeeld teruggeeft met waarden in het uint8-bereik.

```
import numpy as np
import cv2
import matplotlib.pyplot as plt


def change(image1,image2):
    return image1 - image2
```



(a)



(b)

Figuur 5. Twee beelden waartussen een relevant verschil kan berekend worden; (a) watch1 (b) watch2.

```
def main():
    im1 = plt.imread('images/watch1.jpg')
    im2 = plt.imread('images/watch2.jpg')
    changeIm = change(im1,im2)

    plt.figure()

    plt.subplot(1,3,1)
    plt.title('Image 1')
    plt.axis('off')
    plt.imshow(im1,cmap='gray')

    plt.subplot(1,3,2)
    plt.title('Image 2')
    plt.axis('off')
    plt.imshow(im2,cmap='gray')

    plt.subplot(1,3,3)
    plt.title('Change')
    plt.axis('off')
    plt.imshow(changeIm,cmap='gray')
    plt.show()


if __name__ == '__main__':
    main()
```

Wat is er mis met de functie `change`? Corrigeer deze en zorg dat de verschillen tussen de beelden duidelijk zichtbaar zijn binnen het uint8-bereik.
Vraag 3: ***Waarom geeft deze functie het verwachte resultaat niet weer? Waarmee moet je rekening houden bij bewerkingen op afbeeldingen?***

4 Indienen

Indienen van het labo gebeurt via Opdrachten op Ufora. De code dien je in als 1 pythonscript genaamd **labo1.py** waarin alle opdrachten sequentieel worden uitgevoerd. Voeg een **README.txt** bestand toe waarin je programma beschreven is en waarin de vragen beantwoord worden. Alle gevraagde outputafbeeldingen geef je mee in een .zip-file genaamd **output.zip**.

Tip: gebruik `cv2.imwrite` of `plt.savefig` of een afbeelding op te slaan. Gebruik geen screenshots hiervoor.