

IPCV: Übung 1

Übung von: Remo Schwarzentruher

Intro: Einführung in OpenCV mit python

Installation OpenCV:

OpenCV kann am einfachsten mit dem python installer pip installiert werden. Je nach Umgebung oder aufgesetzter python Version ist das `pip` oder `pip3`. Wir werden in der Vorlesung mit python Version 3.5 (oder später) arbeiten.

```
pip install opencv-contrib-python
```

oder

```
pip3 install opencv-contrib-python
```

Dies installiert sowohl die python Bindings für OpenCV, wie auch die eigentliche OpenCV Bibliothek.

Dokumentation

- Documentation von OpenCV gibt es auf: <https://docs.opencv.org/3.4.0/> (<https://docs.opencv.org/3.4.0/>)
- Dokumentation von Python, numpy oder matplotlib sollte Über das Help Menu in Jupyter erreichbar sein

Verwendung

Um OpenCV in Python verwenden zu können muss die Bibliothek importiert werden. Für Berechnungen und Darstellung brauchen wir auch noch numpy und matplotlib:

```
In [1]: # OpenCV needs to be included first
import cv2
import numpy as np

# for displaying images in jupyter
import matplotlib as mpl
from matplotlib import pyplot as plt

# this makes the images a bit larger
mpl.rcParams['figure.dpi'] = 200
plt.rcParams["figure.figsize"] = (10,20)

# plots directly in the notebook
%matplotlib inline

# if you have a high-dpi monitor
%config InlineBackend.figure_format = 'retina'
```

Bilder lesen

Für einige Operationen gibt es sowohl in OpenCV wie auch in anderen Python Bibliotheken implementationen. Zum Beispiel zum Lesen von Bildern. Wir werden, wenn immer möglich zuerst die OpenCV Implementation verwenden.

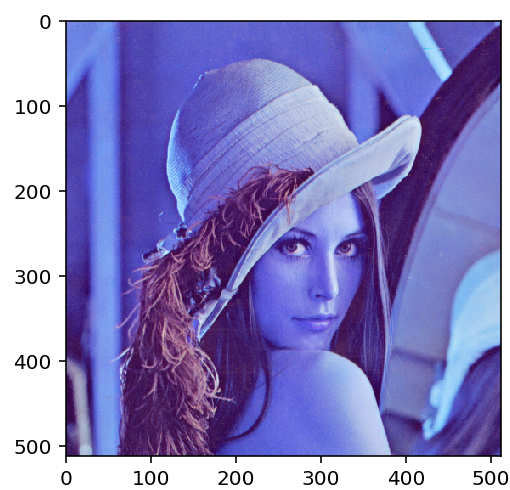
Ein Bild kann mit dem OpenCV Befehl `imread` gelesen werden:

```
In [2]: image = cv2.imread('../data/lena_std.tif')
```

OpenCV hat auch die Möglichkeit ein Fenster zu öffnen und ein Bild darin darzustellen. Innerhalb jupyter (i.e. im Browser) funktioniert dies aber im Allgemeinen nicht. Wir können aber ein Bild mit der Matplotlib Bibliothek darstellen.

```
In [3]: plt.imshow(image)
```

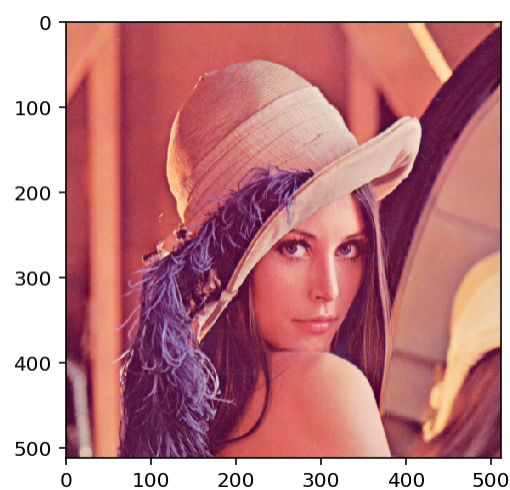
```
Out[3]: <matplotlib.image.AxesImage at 0x11ddb630>
```



Wie wir sehen, sieht das Bild aber falsch aus. Das liegt daran, dass OpenCV das Bild in BGR Format liest, die meisten anderen Bibliotheken aber RGB erwarten. Wir können das Bild entsprechend konvertieren und nochmals darstellen.

```
In [4]: image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image_rgb)
```

```
Out[4]: <matplotlib.image.AxesImage at 0x11da037b8>
```



Numpy arrays

Intern ist ein Bild als ein 3-dimensionales Numpy Array abgelegt. Numpy Arrays sind eine Erweiterung der normalen python Arrays auf mehrere Dimensionen und ermöglichen auch einfachere Operationen und Indexierung.

Ein einzelner Farbwert kann direkt über die Indexierung angesprochen werden, währendem die Verwendung von 2 Koordinaten die RGB Werte als Array liefert.

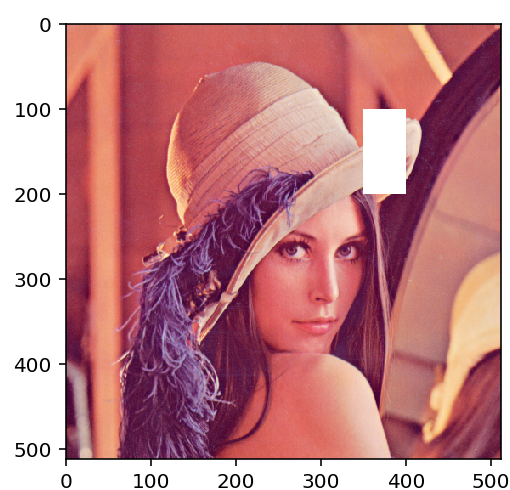
```
In [5]: print(image_rgb[0, 0, 0])
print(image_rgb[0,0])
```

```
226
[226 137 125]
```

Es können auch direkt Bereiche angesprochen werden. Dies setzt ein Teil des Bildes auf weiss:

```
In [6]: image_rgb[100:200, 350:400] = [255, 255, 255]
plt.imshow(image_rgb)
```

```
Out[6]: <matplotlib.image.AxesImage at 0x11f7b1be0>
```



Schauen Sie sich dazu in der numpy Dokumentation die Möglichkeiten der Indexierung an. Eine Übersicht über numpy Operationen gibt es auch auf

https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf

(https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf)

Beachten Sie das die erste Koordinate in der Indexierung der Reihe im numpy Array entspricht, i.e. der y Koordinate im Bild.

Mit numpy Arrays kann auch ein Boolean Array als Index verwendet werden, ausserdem sind Operationen direkt auf einem ganzen Array möglich. Um zum Beispiel alle Pixel die eine Helligkeit kleiner als 100 haben, auf Schwarz zu setzen:

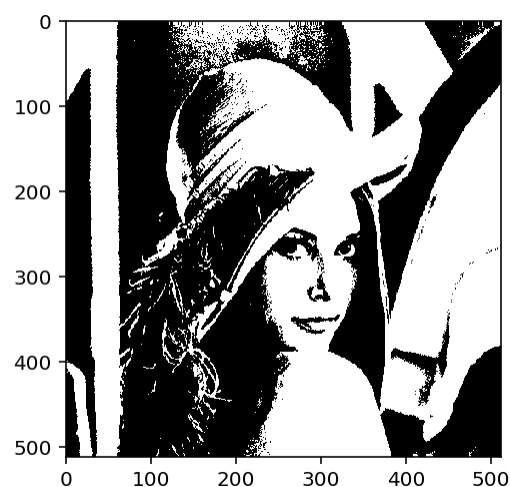
```
In [7]: image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image_grey = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

threshold_map = image_grey <= 128

image_rgb[threshold_map] = [0, 0, 0]
image_rgb[image_grey > 128] = [255, 255, 255]

plt.imshow(image_rgb)
```

Out[7]: <matplotlib.image.AxesImage at 0x11fb9a0f0>



Aufgabe 1: Bildmanipulationen mit OpenCV und numpy

Einführung

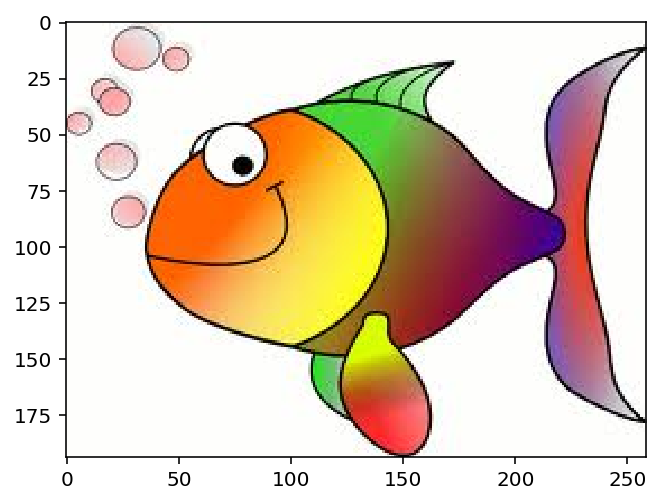
In dieser Übung sollen einfache Bildoperationen mit OpenCV und numpy durchgeführt werden.

Aufgabe 1: Laden und Darstellung von Bildern

Laden sie ein Bild wie oben, konvertieren sie es nach RGB und stellen Sie es dar. Sie können das gleiche, oder ein anderes Bild verwenden.

```
In [8]: image = cv2.imread("../data/HappyFish.jpg")
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image_rgb)
```

Out[8]: <matplotlib.image.AxesImage at 0x11fcf09b0>



Spiegeln sie das Bild an der vertikalen und horizontalen Achse unter Benutzung von Funktionen aus OpenCV und stellen Sie die Bilder dar.

Note Remo:

Flip (Mirror) Vertically : 0

Flip (Mirror) Horizontally: 1

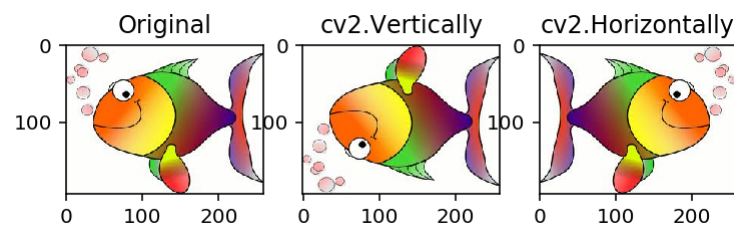
```
In [9]: image_mirror_vert = cv2.flip(image_rgb, 0)
image_mirror_hor = cv2.flip(image_rgb, +1)

plt.subplot(1, 3, 1)
plt.title("Original")
plt.imshow(image_rgb)

plt.subplot(1, 3, 2)
plt.title("cv2.Vertically")
plt.imshow(image_mirror_vert)

plt.subplot(1, 3, 3)
plt.title("cv2.Horizontally")
plt.imshow(image_mirror_hor)
```

Out[9]: <matplotlib.image.AxesImage at 0x120058dd8>



Spiegeln mit numpy

Sie können Bilder auch direkt mit numpy Befehlen verändern. Können sie sie auch damit spiegeln?

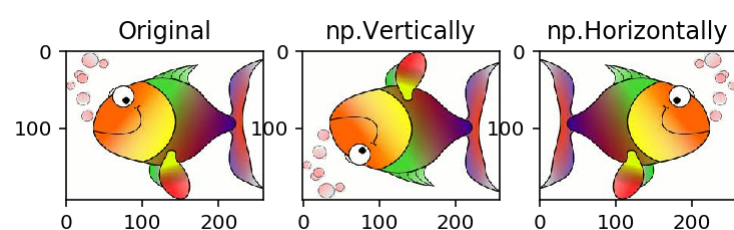
```
In [10]: image_mirror_np_vert = np.flip(image_rgb, 0)
image_mirror_np_hor = np.flip(image_rgb, 1)

plt.subplot(1,3,1)
plt.title("Original")
plt.imshow(image_rgb)

plt.subplot(1,3,2)
plt.title("np.Vertically")
plt.imshow(image_mirror_np_vert)

plt.subplot(1,3,3)
plt.title("np.Horizontally")
plt.imshow(image_mirror_np_hor)
```

Out[10]: <matplotlib.image.AxesImage at 0x1201db160>



Aufgabe 2: Darstellung von Grauwert Bilder

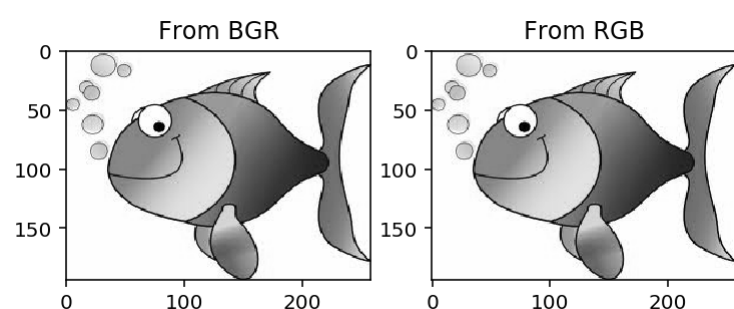
Konvertieren Sie das Bild in ein Grauwert Bild und stellen Sie dieses dar. Per default wird ein 1-Kanal Bild farbig mit einer default Color Map dargestellt. Um das Bild wirklich in grau zu zeichnen kann der Parameter `cmap= 'gray'` für `imshow` mitgegeben werden.

```
In [11]: image_gray_from_bgr = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
image_gray_from_rgb = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2GRAY)

plt.subplot(1,2,1)
plt.title("From BGR")
plt.imshow(image_gray_from_bgr,cmap='gray')

plt.subplot(1,2,2)
plt.title("From RGB")
plt.imshow(image_gray_from_rgb,cmap='gray')
```

Out[11]: <matplotlib.image.AxesImage at 0x120299b00>



Aufgabe 3: Bild Kanäle

Stellen Sie die Anteile des rot, grün und blau Kanals einzeln (als Grauwert Bild) dar. Mit subplot können sie verschiedene Bilder neben- oder untereinander darstellen.

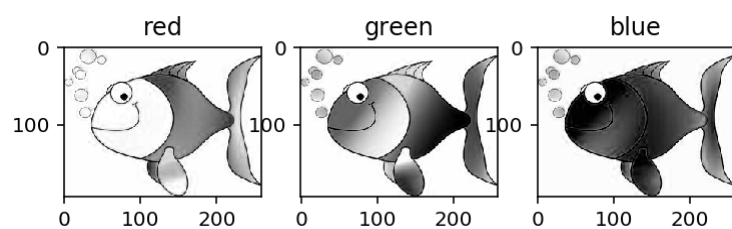
```
In [12]: b, g, r = cv2.split(image)
         image_red = r
         image_green = g
         image_blue = b

         plt.subplot(1, 3, 1)
         plt.title("red")
         plt.imshow(image_red, cmap='gray')

         plt.subplot(1, 3, 2)
         plt.title("green")
         plt.imshow(image_green, cmap='gray')

         plt.subplot(1, 3, 3)
         plt.title("blue")
         plt.imshow(image_blue, cmap='gray')

Out[12]: <matplotlib.image.AxesImage at 0x120395c88>
```



Erstellen Sie nun wieder ein Farbbilder aus den 3 Kanälen, sie können die Kanäle auch anders mischen. Verwenden Sie die numpy Funktion `stack`.

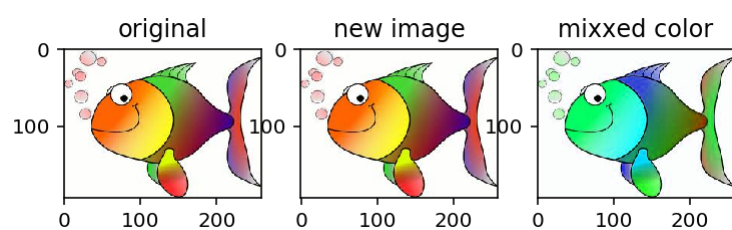
```
In [13]: image_new = np.stack([r,g,b], axis=-1)
         image_mixxed = np.stack([b,r,g], axis=-1)

         plt.subplot(1,3,1)
         plt.title("original")
         plt.imshow(image_rgb)

         plt.subplot(1,3,2)
         plt.title("new image")
         plt.imshow(image_new)

         plt.subplot(1,3,3)
         plt.title("mixxed color")
         plt.imshow(image_mixxed)

Out[13]: <matplotlib.image.AxesImage at 0x1204cceb8>
```



Aufgabe 4: Plot einer Zeile

Stellen Sie die Bildwerte der Zeile ~~200~~ 100 des Grauwert Bildes als Plot dar. Dazu müssen sie die Pixelwerte einer Zeile auslesen wobei Sie wiederum ein (1D) numpy Array erhalten. Dieses kann mit `plt.plot(...)` dargestellt werden. Entspricht das Ergebnis den Erwartungen?

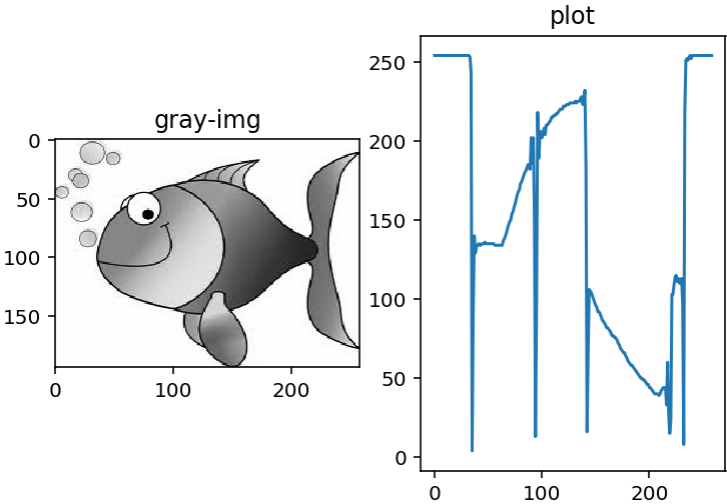
```
In [14]: image_gray = image_gray_from_rgb.copy()
         image_gray.shape

Out[14]: (194, 259)
```



```
In [15]: line = image_gray[100, :]  
  
plt.subplot(1,2,1)  
plt.title("gray-img")  
plt.imshow(image_gray,cmap='gray')  
  
plt.subplot(1,2,2)  
plt.title("plot")  
plt.plot(line)
```

Out[15]: [



Ergebnis

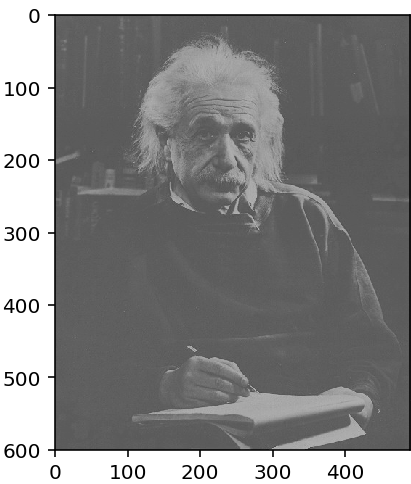
Man sieht hier genau, dass bei den Übergänge (von dem Hintergrund zum Fisch) die Farbe entsprechend geändert werden. Auch sieht man den Verlauf der Graustufe auf dem Kopf des Fisches gut. (Von Dunkel zu Hell)

Aufgabe 5: Histogramm und Histogramm Ausgleich

Das folgenden Bild hat einen schlechten Kontrast. Grauwert Bilder werden per default von OpenCV als 3-Kanal Bilder geladen. Was ist der minimale und maximale Pixelwert im Bild?

```
In [16]: einstein = cv2.imread('../data/EinsteinLowContrast.tif')  
einstein_grey = cv2.cvtColor(einstein, cv2.COLOR_BGR2GRAY)  
plt.imshow(einstein, cmap='gray')  
  
colors = einstein_grey.flatten().tolist()  
min = min(colors)  
max = max(colors)  
print('min: {}, max: {}'.format(min, max))
```

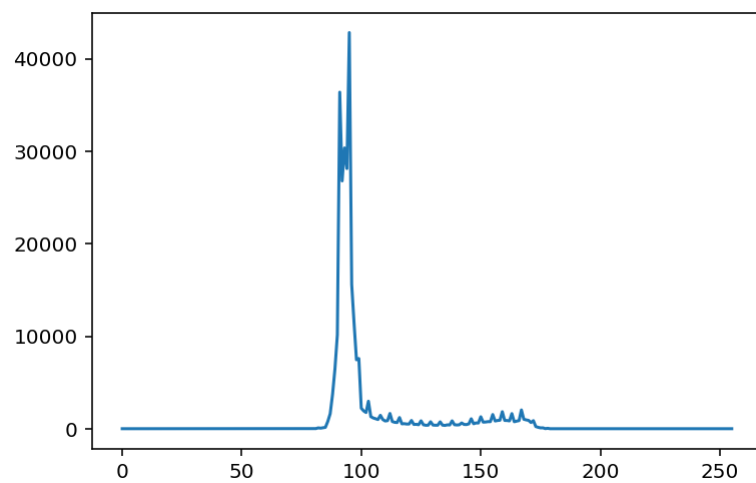
min: 74, max: 178



Stellen Sie das Histogramm des Bildes dar. Benutzen Sie dazu die OpenCV Funktion `cv2.calcHist(...)`. Benutzen Sie die OpenCV Dokumentation um die Parameter der Funktion zu ermitteln.

```
In [17]: hist = cv2.calcHist([einstein_grey], [0], None, [256], [0, 256])
plt.plot(hist)
```

```
Out[17]: [<matplotlib.lines.Line2D at 0x1206fd198>]
```

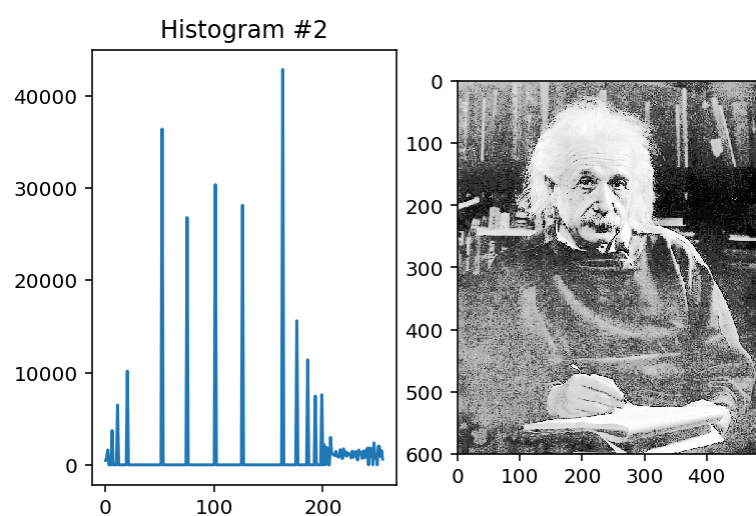


Führen Sie einen Histogramm Ausgleich mit der Funktion `equalizeHist(...)` durch und stellen Sie das Bild und das Histogramm des veränderten Bildes dar.

```
In [18]: einstein_equal = cv2.equalizeHist(einstein_grey)
hist2 = cv2.calcHist([einstein_equal], [0], None, [256], [0,256])

plt.subplot(1, 2, 1)
plt.title("Histogram #2")
plt.plot(hist2)
plt.subplot(1, 2, 2)
plt.imshow(np.stack([einstein_equal, einstein_equal, einstein_equal], axis=2))
```

```
Out[18]: <matplotlib.image.AxesImage at 0x1207d4b00>
```



Aufgabe 6: Punktbildfunktionen

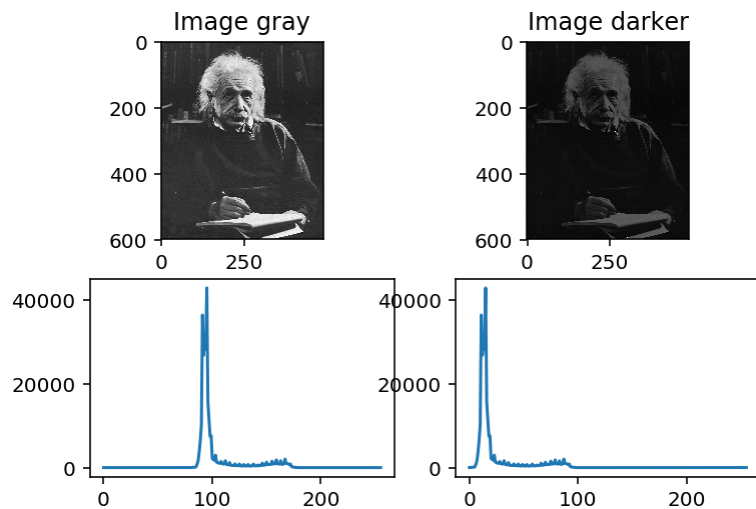
In der nächsten Zelle ist eine Funktion definert, die als Eingabe ein Bild enthält, und ein verändertes Bild zurückgibt. Ändern Sie die Funktion, sodass das Bild heller (oder dunkler) wird.

```
In [19]: def f(x):
    darker = x.copy()
    darker[:, :] = darker[:, :] - 80
    return darker
```

```
In [20]: plt.subplot(2,2,1)
plt.title("Image gray")
plt.imshow(einstein_grey, cmap='gray')
plt.subplot(2,2,3)
hist = cv2.calcHist([einstein_grey], [0], None, [256], [0, 256])
plt.plot(hist)

image_result = f(einstein_grey)
plt.subplot(2,2,2)
plt.title("Image darker")
plt.imshow(image_result, cmap='gray')
plt.subplot(2,2,4)
hist = cv2.calcHist([image_result], [0], None, [256], [0, 256])
plt.plot(hist)
```

Out[20]: [<matplotlib.lines.Line2D at 0x121774630>]



Aufgabe 2: Morphologische Operationen

Einführung

In dieser Aufgabe wollen wir morphologische Operationen in OpenCV betrachten.

```
In [21]: # OpenCV needs to be included first
import cv2
import numpy as np

# for displaying images in jupyter
import matplotlib as mpl
from matplotlib import pyplot as plt
%matplotlib inline
mpl.rcParams['figure.dpi'] = 200
```

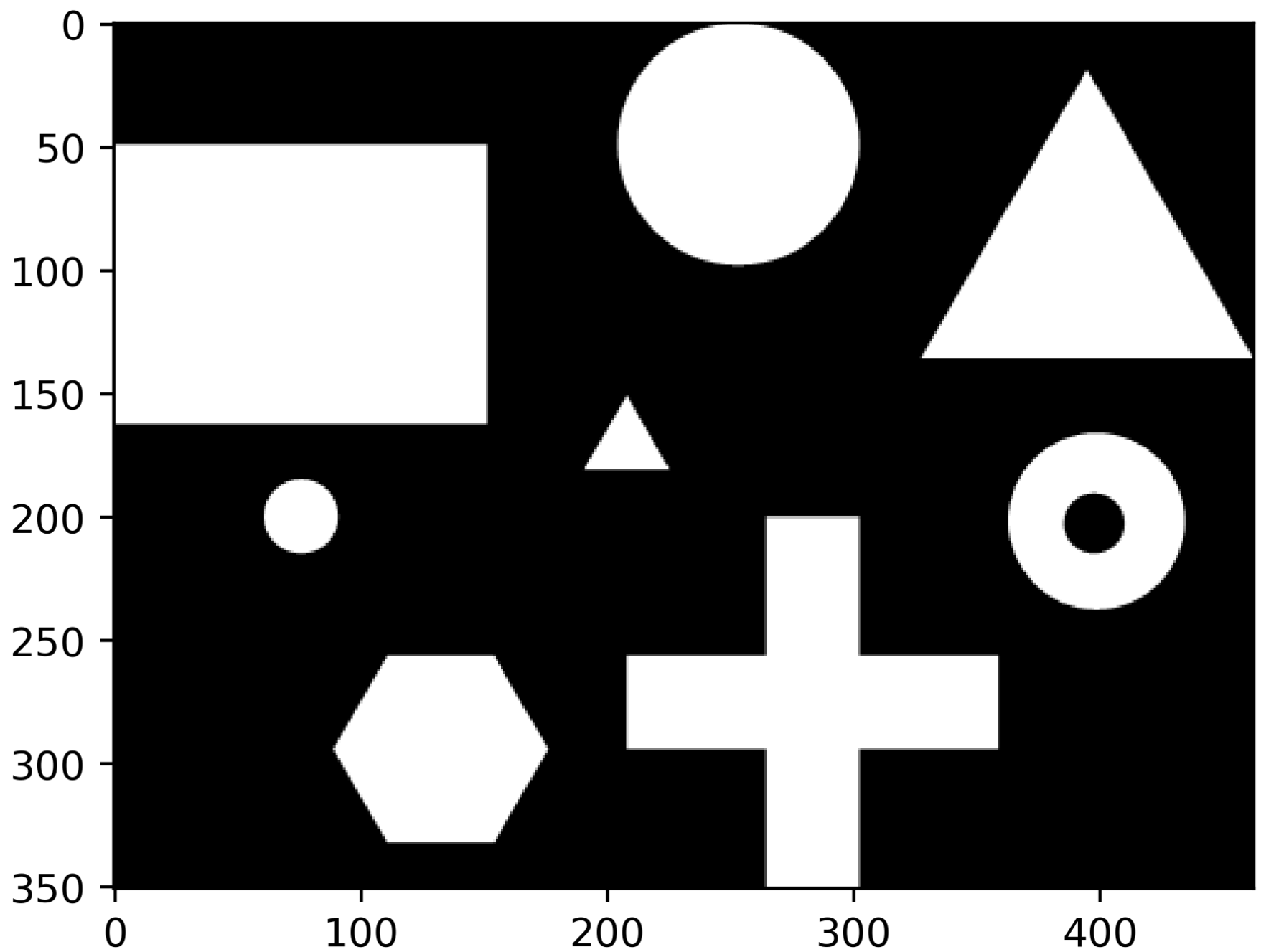
Aufgabe 2.1: Dilate and Erode

Morphologische Operationen werden typischerweise auf binären Bildern angewendet. (Die OpenGL Implementationen verwenden jeweils die Maximum Operationen zwischen Strukturelement und Bild, sodass die Operationen zumindest auch auf Grauwert Bildern durchgeführt werden können, ausserdem werden die Operationen auf allen Kanälen angewendet.)

Ein Beispiel Bild steht unter images/Figuren.png zur Verfügung. Typischerweise erhalten sie ein binäres Bild sonst mit einer Schwellwert (Threshold) Operation.


```
In [22]: image = cv2.imread('../data/Figuren.png')  
plt.imshow(image)
```

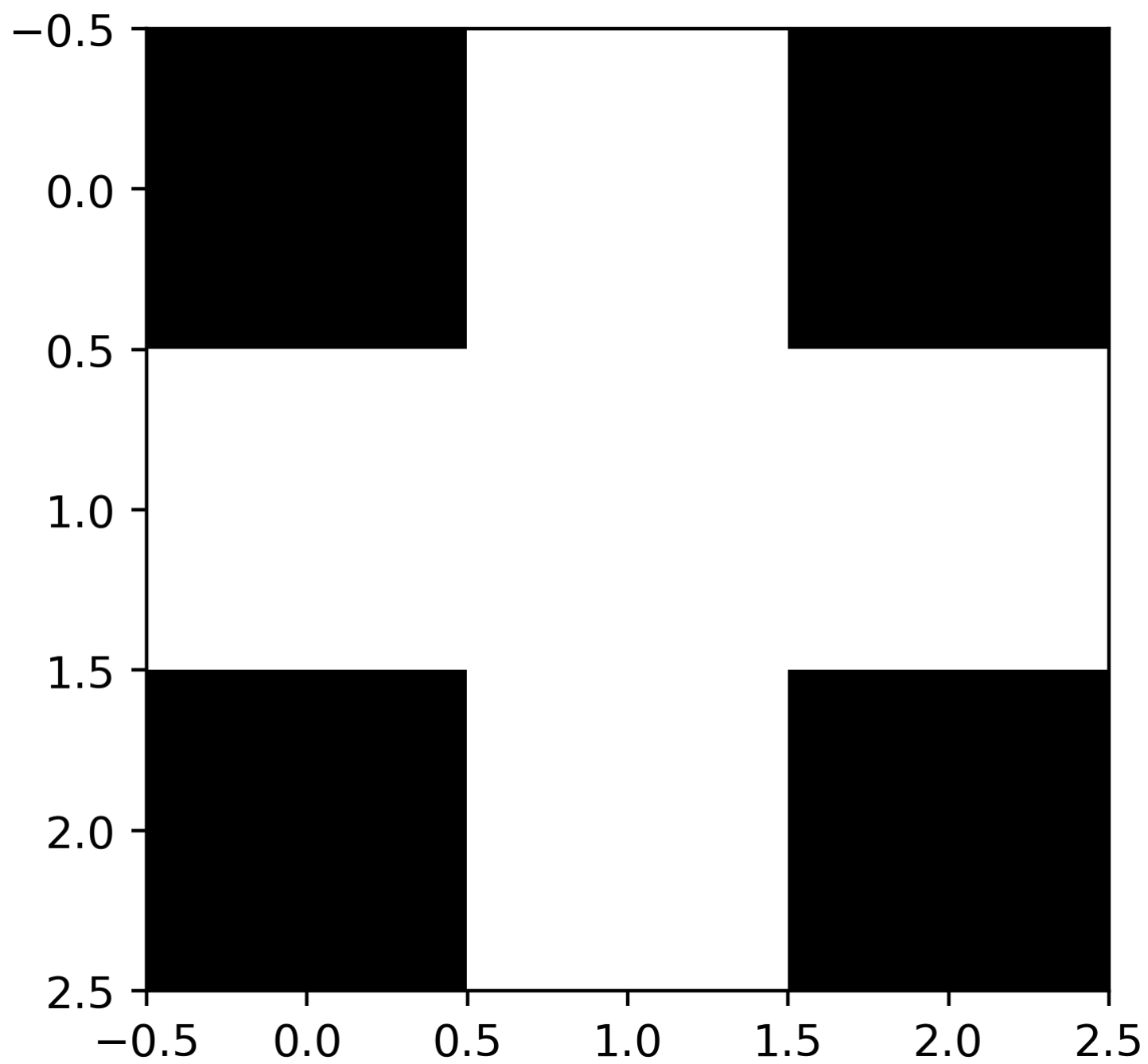
```
Out[22]: <matplotlib.image.AxesImage at 0x121842ef0>
```



Wenden Sie die morphologischen Operationen dilate und erode mit verschiedenen Strukturelementen und verschiedenen Grössen an. Strukturelemente können mit der Funktion `getStructuringElement` erzeugt werden. Die OpenGL Implementation erlaube es auch die Operationen mit einem Aufruf mehrmals anzuwenden.

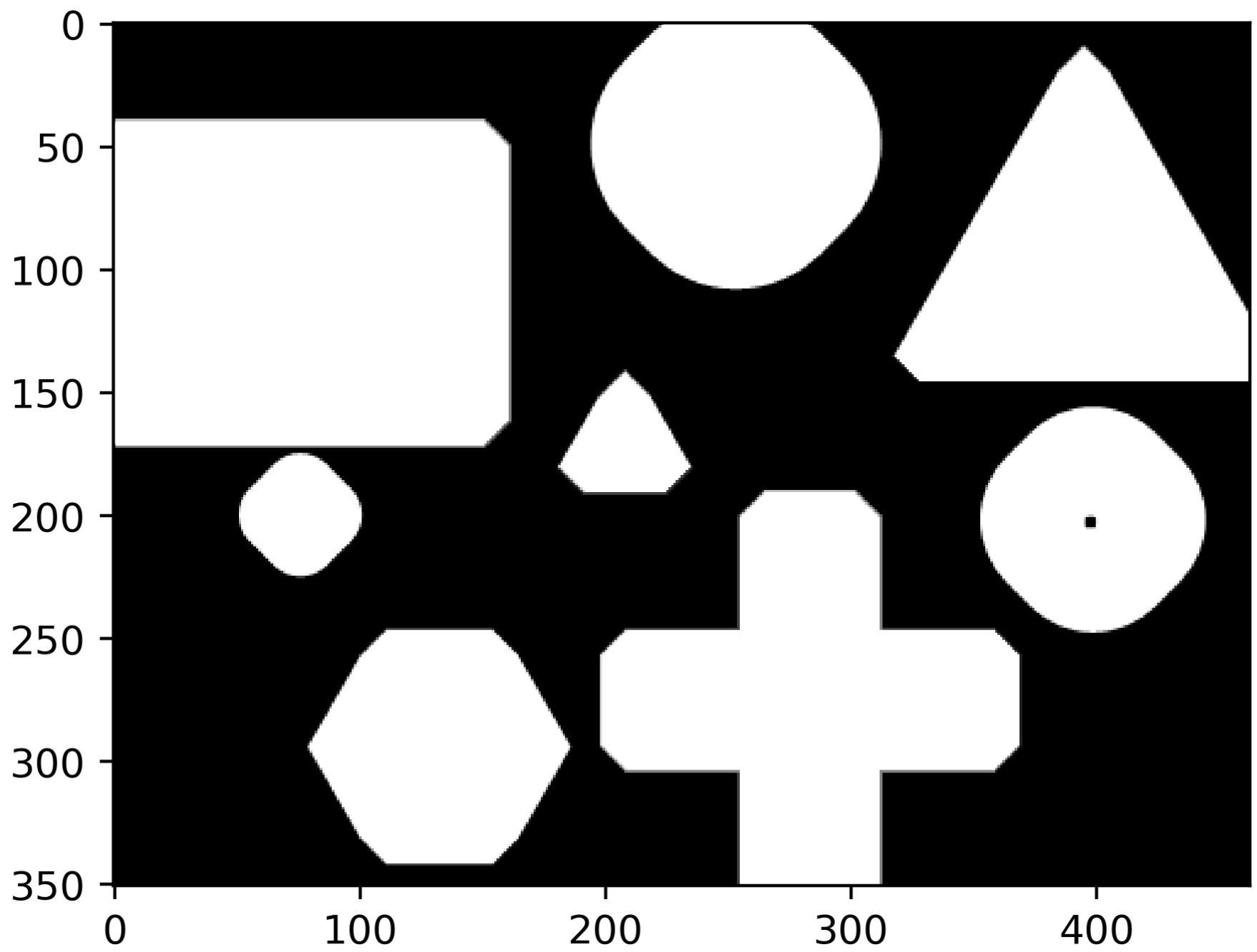
```
In [23]: struct = cv2.getStructuringElement(cv2.MORPH_CROSS, (3,3))  
plt.imshow(struct, cmap='gray')
```

```
Out[23]: <matplotlib.image.AxesImage at 0x123f0c4e0>
```



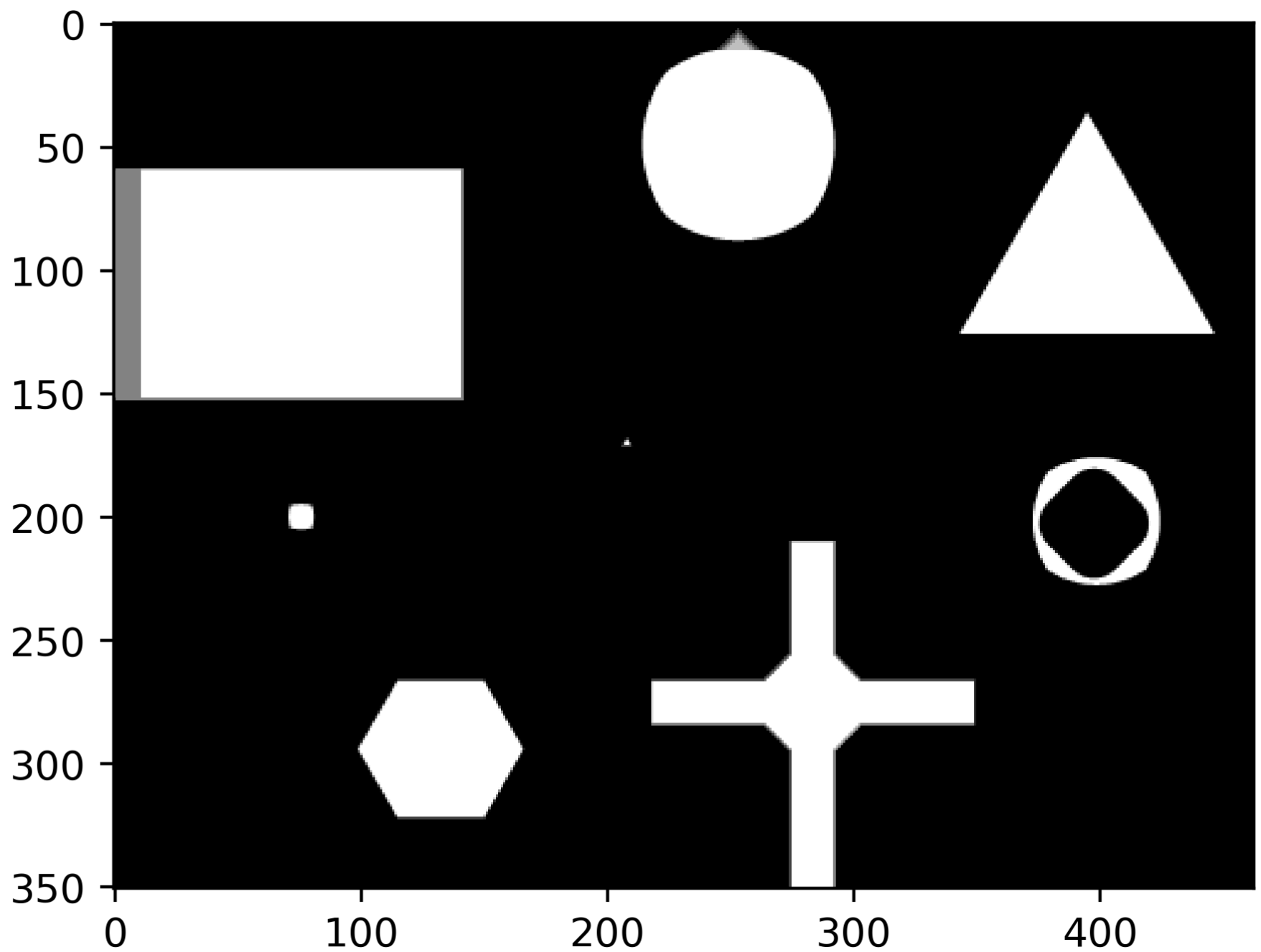
```
In [24]: image_dilate = cv2.dilate(image, struct, iterations=10)
plt.imshow(image_dilate)
```

```
Out[24]: <matplotlib.image.AxesImage at 0x124228cc0>
```



```
In [25]: image_erode = cv2.erode(image, struct, iterations=10)
plt.imshow(image_erode)
```

```
Out[25]: <matplotlib.image.AxesImage at 0x1244660b8>
```

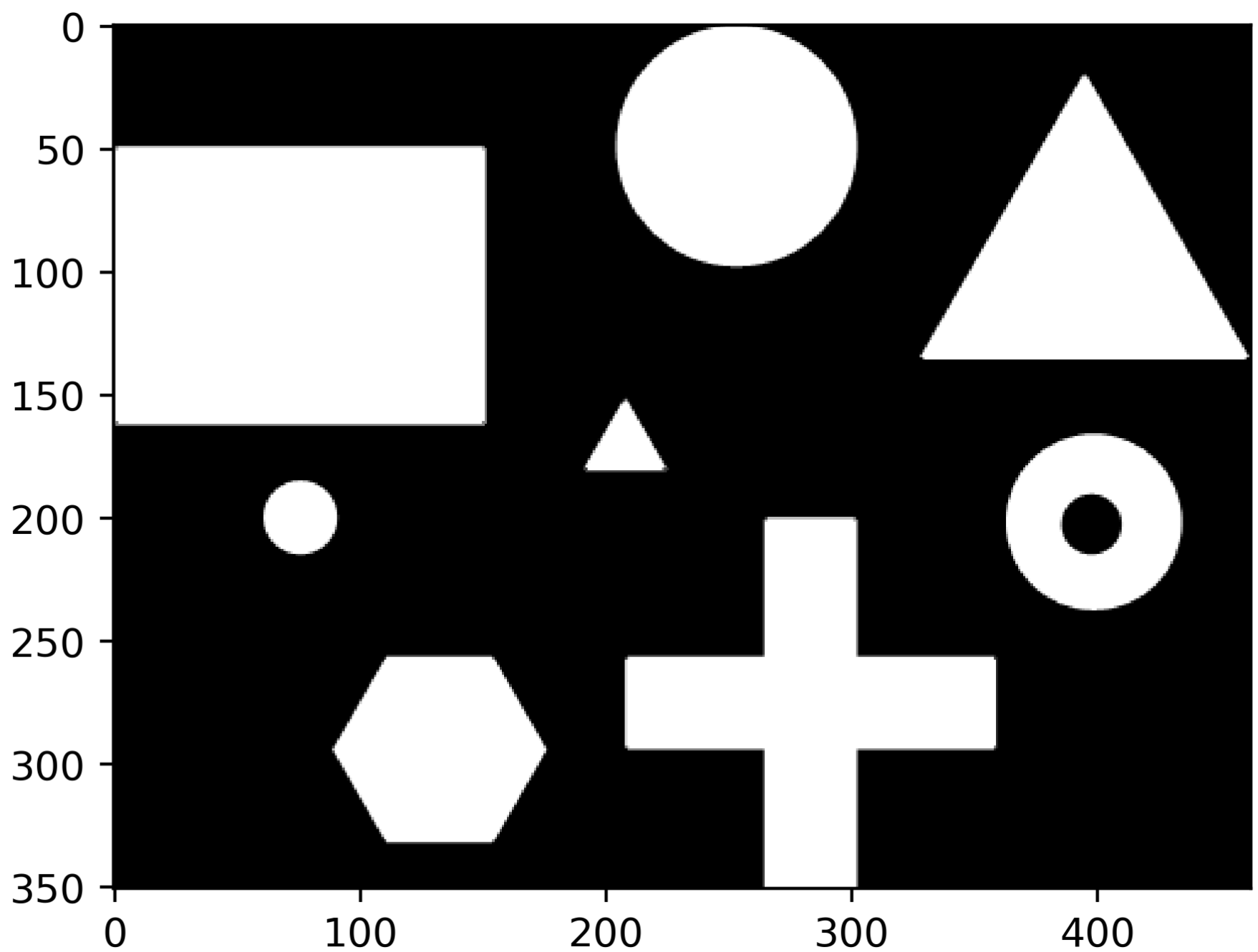


Aufgabe 2.2: Opening/Closing

Für Opening und Closing kann in OpenCV die Funktion `morphologyEx` verwendet werden. Wie wirkt sich opening and closing auf das Beispielbild aus?

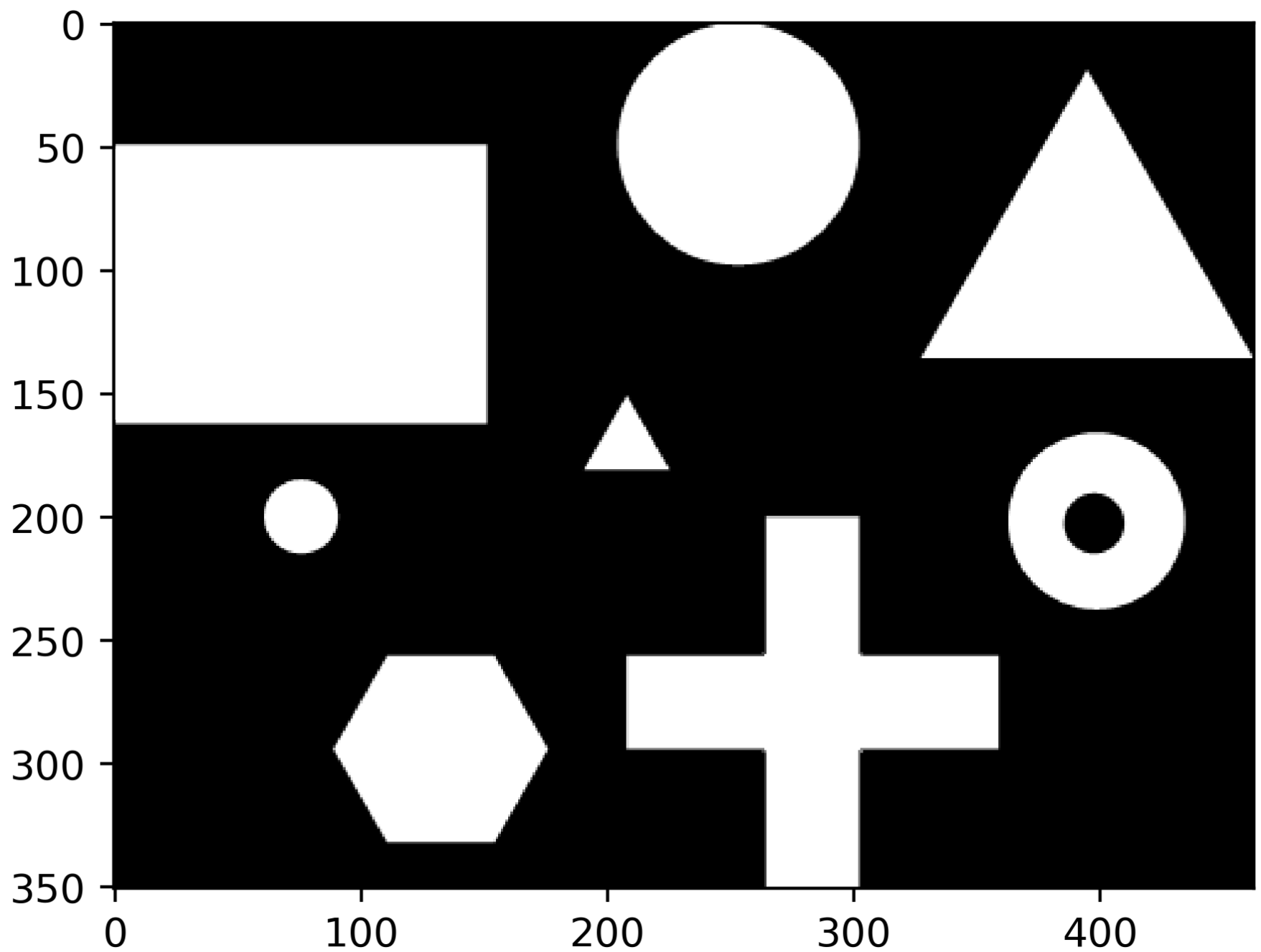
```
In [26]: image_open = cv2.morphologyEx(image, cv2.MORPH_OPEN, struct)  
plt.imshow(image_open)
```

```
Out[26]: <matplotlib.image.AxesImage at 0x124499470>
```



```
In [27]: image_close = cv2.morphologyEx(image, cv2.MORPH_CLOSE, struct)
plt.imshow(image_close)
```

```
Out[27]: <matplotlib.image.AxesImage at 0x127bad828>
```



Aufgabe 2.3: Kugeln zählen

Finden Sie eine Sequenz von OpenCV Operationen die die Kugeln in folgenden Bild zählt? Als Ausgabe sollten sie dann also schlussendlich eine Zahl (zu Speichern in `nr_kugeln`) erhalten.

Hinweis: Beachten Sie dass die Kugeln schwarz auf hellem Hintergrund dargestellt sind.


```
In [28]: kugeln = cv2.imread('../data/Kugeln.jpg')

kugeln = cv2.cvtColor(kugeln, cv2.COLOR_BGR2GRAY)
struct = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3,3))
kugeln = cv2.erode(kugeln, struct, iterations=3)

plt.subplot(1,2,1)
plt.imshow(kugeln, cmap='gray')

kugeln = cv2.dilate(kugeln, struct, iterations=20)
plt.subplot(1,2,2)
plt.imshow(kugeln, cmap='gray')

ret, thresh = cv2.threshold(kugeln, 127, 255, 0)
contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

nr_kugeln = len(contours)
print('Anzahl Kugeln: {}'.format(nr_kugeln))
```

Anzahl Kugeln: 78

