Aufgabenblatt 1

Einführung in die Bildverarbeitung
Christian Wilms und Simone Frintrop
SoSe 2020

Ausgabe: 24. April 2020 - **Abgabe bis: 8. Mai 2020, 10:00**

Hinweis: Ihr habt zwei Wochen Zeit für dieses Aufgabenblatt!

Aufgabe 1 — NumPy und Matplotlib Tutorial - 30 Punkte - Theorieaufgabe
Arbeitet das kurze Tutorial[1] von Justin Johnson zu NumPy und Co. durch. Bearbeitet dabei nur die Teile zu NumPy (bis Numpy Documentation) und Matplotlib (bis zum Ende). Testet anschließend euer Wissen und beantwortet folgende Fragen schriftlich jeweils mit einem kurzen Antworttext (1-2 Sätze).

1. Wie kann die Anzahl der Zeilen und Spalten eines NumPy-Arrays bestimmt werden?
2. Was unterscheidet die Funktionen `numpy.array` und `numpy.zeros`?
3. Wie kann der Datentyp eines gegebenen NumPy-Arrays ermittelt werden?
4. Was bedeutet `.T` hinter dem Variablennamen eines NumPy-Arrays? Beispielsweise: `a.T`
5. Was machen die Funktionen `xlabel` und `ylabel` aus `matplotlib.pyplot`?

Aufgabe 2 — Erste Schritte in NumPy - 25 Punkte - Programmieraufgabe
In dieser Aufgabe sollen Matrizen und Vektoren aus NumPy-Arrays erzeugt und manipuliert werden. Alle Schritte sollen in Python programmiert und in einem Python-Skript festgehalten werden. Im Rahmen dieser Aufgabe ist nur die Nutzung von NumPy erlaubt.

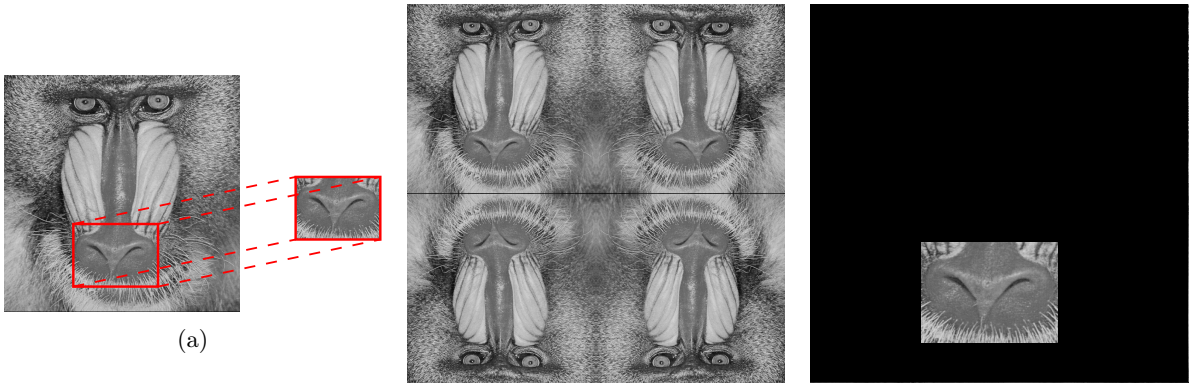
1. Erzeugt zunächst einen Vektor `u` als 1D-Array mit 100 Nullen.
2. Erzeugt zudem einen Vektor `v` als 1D-Array aus der Liste: `[0,1,2,3,4,5,6,7,8,9,10,11]`.
3. Formt den Vektor `v` nun in eine Matrix `m` (2D-Array) um, die aus drei Zeilen und vier Spalten besteht. Tipp: NumPy kennt eine Funktion, um die Form (engl. *shape*) eines Arrays zu ändern.
4. Multipliziert alle Einträge der Matrix `m` mit dem konstanten Faktor 1.2.
5. Ändert nun den Datentyp der Matrix `m` auf `np.int`. Tipp: Nutzt die Methode `astype` eines NumPy-Arrays.
6. Multipliziert alle Einträge der Matrix `m` erneut mit dem konstanten Faktor 1.2. Von welchem Datentyp ist jetzt das Ergebnis?
7. Berechnet das Hadamard-Produkt[2] (elementweise Multiplikation) der Matrix `m` mit sich selbst (`m`$\odot$`m`).

Aufgabe 3 — Basisoperationen mit Bildern - 25 Punkte - Programmieraufgabe
Diese Aufgabe dient dem ersten Kontakt mit Bildern in Python. Dazu sollt ihr gegebene Bilder manipulieren und alle Schritte erneut in einem Python-Skript festhalten. Im Rahmen dieser Aufgabe ist nur die Nutzung von NumPy, Matplotlib sowie `skimage.io.imread` und `skimage.io.imsave` zum Laden bzw. Speichern eines Bildes erlaubt.

1. Speichert euch das Bild `mandrill.png` aus dem Moodle ab und ladet es in Python mithilfe der Funktion `skimage.io.imread`.
2. Zeigt das Bild mithilfe von Matplotlib an.
3. Erstellt ein neues Array als Bildausschnitt, das, wie in Abbildung 1a visualisiert, etwa die Nasenspitze des Mandrill beinhaltet.

[1] https://cs231n.github.io/python-numpy-tutorial/
[2] https://de.wikipedia.org/wiki/Hadamard-Produkt

4. Speichert den Bildausschnitt mit der Funktion `skimage.io.imsave` ab.
5. Erstellt eine Kopie des Bildes und setzt ein einziges Pixel (Arrayelement bzw. Bildelement) auf schwarz. Lasst euch das manipulierte Bild anzeigen. Erkennt man den Unterschied? (kein Antworttext nötig)
6. Erstellt eine weitere Kopie des Bildes und setzt den gesamten Bereich der Augen des Mandrills auf schwarz. Lasst euch das manipulierte Bild erneut anzeigen. Erkennt man nun den Unterschied? (kein Antworttext nötig)



(a) (b) (c)

Abbildung 1: **(a)**: Bildausschnitt der Nasenspitze aus dem Originalbild (links) als neues Bild. **(b)**: Komposition eines neuen Bildes aus vier, teilweise gespiegelten Originalbildern. **(c)**: Maskiertes Originalbild in dem nur die Nasenspitze übrig ist.

Aufgabe 4 — Fortgeschrittene Bearbeitung von Bildern - 40 Punkte - Programmieraufgabe
In dieser Aufgabe sollen weitere Manipulationen an einem Beispielbild erprobt werden. Schreibt dazu wieder ein Python-Skript, das alle Schritte eurer Manipulationen beinhaltet. Im Rahmen dieser Aufgabe ist nur die Nutzung von NumPy, Matplotlib sowie `skimage.io.imread` zum Laden eines Bildes erlaubt.

1. Erstellt ein neues Bild basierend auf drei Variationen des Bildes `mandrill.png` aus dem Moodle sowie dem Originalbild.
   a) Ladet dazu zunächst erneut das Bild `mandrill.png` aus dem Moodle in Python.
   b) Spiegelt das Bild an der vertikalen Achse (linke und rechte Seite tasuchen) und speichert das Ergebnis in eine neue Variable als erste Variation.
   c) Spiegelt das Originalbild (`mandrill.png`) nun an der horizontalen Achse und speichert das Ergebnis erneut in einer neuen Variable. Dies ist die ist die zweite Variation.
   d) Führt jetzt beide Spiegelungen nacheinander auf dem Originalbild aus speichert das Ergebnis wiederum in einem neuen Bild als dritte Variation.
   e) Erzeugt nun ein neues Bild, das doppelt so hoch und breit ist wie das Bild `mandrill.png`. Setzt in jeden Quadranten des neuen Bildes eine der vier Versionen des Bildes ein. Oben links soll das Origianlbild zu sehen sein, rechts daneben das an der vertikalen Achse gespiegelte Bild, links unten das an der horizontalen Achse gespiegelte Bild und rechts unten das an beiden Achsen gespiegelte Bild. Das Ergebnis sollte etwa wie in Abbildung 1b aussehen. Tipp: Zum Erzeugen von Arrays bzw. Bildern gibt es in NumPy mehrere Funktionen neben `np.array`.
2. Erzeugt nun ein Negativ des Originalbilds (`mandrill.png`). In einem Negativ sind alle Helligkeitswerte umgedreht. D.h. schwarz wird zu weiß, weiß wird zu schwarz und alle Werte dazwischen werden ebenso gedreht. Lasst euch das Bild zur Kontrolle anzeigen.
3. Schneidet erneut die Nasenspitze des Mandrills wie in Abbildung 1a dargestellt als neues Bild aus. Ändert nun in dem neu erzeugten Ausschnitt ein Pixel und findet heraus, ob diese Änderung auch im Originalbild durchgeführt wird.
4. Erstellt euch nun eine Maske für den Bereich der Nasenspitze des Mandrills.
   a) Erzeugt dazu zunächst ein neues Bild, das nur aus Nullen besteht, und die gleiche Größe wie das Originalbild hat. Dieses Bild wird eure Maske.

b) Setzt nun in der Maske alle Pixel, die in dem Bereich liegen, der die Nasenspitze beinhaltet, auf 1.
c) Verrechnet anschließend Maske und Originalbild, sodass der Bereich außerhalb der Nasenspitze schwarz ist und nur im Bereich der Nasenspitze das eigentliche Bild zu sehen ist. Das Ergebnis sollte etwa wie in Abbildung 1c aussehen.

Aufgabe 5 — Broadcasting vs. Schleifen in NumPy - 35 Punkte - Programmieraufgabe
Nun wollen wir die Vorteile des Broadcastings in NumPy gegenüber Schleifen bemessen. Dazu soll eine Abfrage am Bild einmal mit Schleifen und einmal mittels Broadcasting ausgeführt werden. Nebenbei wird die Ausführungszeit mit Hilfe der Funktion `time.time` gemessen. Ein Beispiel für die Verwendung von `time.time` zur Messung von Ausführungszeiten findet ihr unten. Im Rahmen dieser Aufgabe ist nur die Nutzung von NumPy, `skimage.io.imread` zum Laden eines Bildes sowie der Funktion `time.time` zur Zeitmessung erlaubt.

1. Schreibt eine Python-Funktion, die ein Bild als NumPy-Array annimmt und für jeden Arrayeinträge bzw. Pixel prüft, ob der Pixelwert zwischen 99 und 200 liegt. Umgesetzt werden soll dies durch einer Iteration über das Bild mittels zweier `for`-Schleifen.
2. Schreibt eine zweite Funktion für selbigen Zweck. Diesmal jedoch mit Broadcasting, d.h. komplett ohne die Benutzung von Schleifen.
3. Messt wie lange 100 Aufrufe der beiden Funktionen jeweils auf das Bild `mandrill.png` aus dem Moodle dauern. Um die beiden Funktionen aufzurufen, dürft ihr natürlich Schleifen verwenden. Wie groß ist der Unterschied?

```
>>> import time
>>> tic = time.time()
>>> #your magic here
>>> toc = time.time()
>>> diff = toc-tic
>>> print(diff)
```

Aufgabe 6 — Bildstatistiken in NumPy - 35 Punkte - Programmieraufgabe
In dieser Aufgabe geht es um das Finden, Kombinieren und effiziente Benutzung von NumPy-Funktionen auf Bildern. D.h. im Rahmen dieser Aufgabe ist nur die Verwendung von NumPy-Funktionen und `skimage.io.imread` zum Laden eines Bildes erlaubt. Explizit nicht erlaubt ist die Verwendung von Schleifen zum Iterieren über Bilder.

1. Ladet erneut das Bild `mandrill.png` in Python.
2. Ermittelt Minimum, Maximum, Durchschnitt und die Standardabweichung des Bildes mithilfe der entsprechenden NumPy-Funktionen.
3. Lokalisiert das Minimum und das Maximum im Bild. Gebt dazu die x- und y-Koordinate eines Minimums und Maximums auf der Konsole aus. Tipp: Schaut in die Dokumentation der entsprechenden Funktionen, um zu erfahren, wie ihr aus dem einen Ergebnisindex zwei Koordinaten generiert.
4. Zählt wie viele Pixel es mit einem geraden bzw. einem ungeraden Wert im Bild gibt. Nutzt auch hierfür wiederum keine Schleifen!
5. Ermittelt außerdem alle Koordinaten an denen Pixel mit geradem Wert lokalisiert sind. Die Form der Koordinaten bspw. $((x_1, y_1), (x_2, y_2), (x_3, y_3), \dots)$ oder $((x_1, x_2, x_3, \dots), (y_1, y_2, y_3, \dots))$ ist unerheblich.