Aufgabenblatt 7

Einführung in die Bildverarbeitung
Christian Wilms und Simone Frintrop
SoSe 2020

Bei Fragen und Problemen schickt eine Mail an
wilms@informatik.uni-hamburg.de

Ausgabe: 19. Juni 2020 - **Abgabe bis: 26. Juni 2020, 10:00**

```
 Gelöste Aufgaben:
(_) Aufgabe 1.1
(_) Aufgabe 1.2
(_) Aufgabe 1.3
(_) Aufgabe 1.4
(_) Aufgabe 1.5
(_) Aufgabe 1.6
(_) Aufgabe 2.1
(_) Aufgabe 2.2
(_) Aufgabe 2.3
(_) Aufgabe 3.1
(_) Aufgabe 3.2
(_) Aufgabe 3.3
(_) Aufgabe 3.4
(_) Aufgabe 3.5
(_) Aufgabe 3.6
(_) Aufgabe 4.1
(_) Aufgabe 4.2
(_) Aufgabe 4.3
(_) Aufgabe 4.4
(_) Aufgabe 4.5
(_) Aufgabe 4.6
(_) Zusatzaufgabe 5.1
(_) Zusatzaufgabe 5.2
(_) Zusatzaufgabe 5.3
(_) Zusatzaufgabe 5.4
(_) Zusatzaufgabe 5.5
```

0	0	0
0	1	0
0	0	0

(a) Bild f

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

(b) Bild g

Abbildung 1: Bilder für Aufgabe 1 mit Pixelwerten als Zahlen.

Aufgabe 1 — Faltung und Korrelation berechnen - 2 + 3 + 2 + 3 + 2 + 3 = 15 Punkte - Theorieaufgabe
Gegeben seien die beiden Bilder f und g in Abbildung 1a und 1b. Führt die folgenden Faltungen ($\star$) und Korrelationen ($\star$) der Filterkerne

$$k_1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad k_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{und} \quad k_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \tag{1}$$

mit den Bildern f und g händisch durch. Normiert dazu die Filterkerne zunächst so, dass die Summe über sie jeweils 1 ergibt, und umrahmt die Bilder mit einer ausreichenden Anzahl an Nullen (padding). Beschreibt zudem kurz die Wirkung des jeweiligen Filterkerns auf die Bilder. Zusammengefasst sollen also die folgenden Operationen ausgeführt werden:

1. $k_1 \star f$ und $k_1 \star\!\!\star f$
2. $k_1 \star g$ und $k_1 \star\!\!\star g$
3. $k_2 \star f$ und $k_2 \star\!\!\star f$
4. $k_2 \star g$ und $k_2 \star\!\!\star g$
5. $k_3 \star f$ und $k_3 \star\!\!\star f$
6. $k_3 \star g$ und $k_3 \star\!\!\star g$

Aufgabe 2 — Eigenschaften von Faltung und Korrelation - 5 + 10 + 5 = 20 Punkte - Theorieaufgabe

1. Meist werden Glättungs-Filterkerne so normiert, dass die Summe über sie 1 ist. Welchen Hintergrund hat das? Welche Konsequenzen hat bspw. die Anwendung eines Box-Filters auf ein Bild, dessen Gewichte zusammen 0.5 oder 2 ergeben? Beantwortet diese Fragen kurz schriftlich und gebt Beispiele (auch in 1D möglich), die eure Thesen untermauern.
2. Wie in der Vorlesung vorgestellt, ist die Faltung kommutativ. Beweist dies!
3. Im Gegensatz zur Faltung ist die Korrelation i.A. nicht kommutativ. Zeigt dazu ein Gegenbeispiel, gerne auch in 1D.
   Tipp: Berechnet die Korrelation für ein 1×5 Bild mit einem 1×3 Filterkern.

Aufgabe 3 — Rauschreduktion - 5 + 10 + 5 + 5 + 5 + 10 = 40 Punkte - Programmieraufgabe
Erlaubte (Sub-)Pakete: `numpy`, `skimage.io`, `skimage.filters`, `skimage.util`, `matplotlib`, `scipy.ndimage`
Die Faltung kann genutzt werden, um das Rauschen in einem Bild zu mindern. Dazu sollen verschiedene Filterkerne in Kombination mit verschiedenen Typen von Rauschen im Rahmen dieser Aufgabe erprobt werden.

1. Ladet zunächst das Bild `einstein.png` aus dem Moodle und verrauscht es mit Gaußschem Rauschen. Wandelt dafür zunächst den Wertebereich des Bildes von $0 \ldots 255$ auf $0 \ldots 1$ um. Nutzt zum Verrauschen entweder eure eigene Funktion von Aufgabenblatt 2 Aufgabe 5 oder nutzt die Funktion `skimage.util.random_noise()` mit dem Parameter `mode='gaussian'` und einer Varianz von 0.01. Zeigt das verrauschte Bild an.
2. Schreibt eine Funktion, die den mittleren absoluten Unterschied zwischen den korrespondierenden Pixeln eines Bildes berechnet. Dazu müsst ihr durch alle Koordinaten der Bilder iterieren und jeweils die absolute Differenz der Pixelwerte beider Bilder an der entsprechenden Koordinate berechnen. Die Differenzen müssen schließlich aufsummiert und durch die Anzahl der Pixel geteilt werden. Wie groß ist die mittlere Differenz zwischen dem Originalbild (`einstein.png`) und eurer verrauschten Variante?
   Tipp: Ihr könnt hier auch ohne Schleifen auskommen. Nutzt dazu bspw. die NumPy-Funktionen `numpy.abs` und `numpy.mean` und den Operator `-` zwischen Arrays.
3. Wendet nun verschiedene Varianten eines Box-Filters auf das verrauschte Bild an. Die Varianten sollen sich in der Größe des $n \times n$ Filterkerns unterscheiden mit $n \in \{3, 5, 7, 9, 11\}$. Erstellt zunächst einen Filterkern entsprechender Größe und normiert diesen, sodass die Summe über den Filterkern 1 ist. Nutzt nun die Funktion `scipy.ndimage.convolve()`, um das Bild mit dem Filterkern zu falten. Evaluiert, welche Größe n zur geringsten mittleren Differenz zum Originalbild führt. Visualisiert zudem das beste Ergebnis.
4. Wendet nun den Gauß-Filter mit unterschiedlichen Varianzen auf dasselbe verrauschte Bild an. Testet Varianzen im Bereich $0.1, \ldots, 2$ in Schritten von 0.1 und vergleicht das Ergebnis erneut mit Hilfe der mittleren Differenz mit dem Originalbild. Welcher Filter liefert das bessere Ergebnis (geringere mittlere Differenz zum Originalbild)? Visualisiert zudem das beste Ergebnis mit dem Gauß-Filter. Sieht man einen Unterschied im Vergleich zum besten Ergebnis des Box-Filters?

5. Verrauscht nun das Originalbild `einstein.png` mit Salt and Pepper Noise. Nutzt dazu erneut die Funktion `skimage.util.random_noise()` mit dem Parameter `mode='s&p'` oder eure Lösung von Aufgabenblatt 2 Aufgabe 5. Die Wahrscheinlichkeit für das Auftreten von Salt und Pepper soll bei 0.1 liegen. Wendet nun den Gauß-Filter mit verschiedenen Varianzen (erneut im Bereich $0.1, \ldots, 2$ in Schritten von 0.1) auf das mit Salt and Pepper Noise verrauscht Bild an. Wie groß ist die geringste mittlere Distanz zum Originalbild? Visualisiert auch das Ergebnis.
6. Filtert das mit Salt and Pepper Noise verrauschte Bild nun mit dem sog. Median-Filter. Dies bedeutet, dass jeder Pixel durch den Median seiner 3×3-Nachbarschaft ersetzt wird. Implementiert dies händisch (doppelte `for`-Schleife) und nutzt dabei die Funktion `numpy.median()`. Ist das Ergebnis in Bezug auf die mittlere Distanz besser als die Variante mit dem Gauß-Filter? Vergleicht zudem das Ergebnis visuell.

1	2	5	9	8	3	6	7	9	9

Abbildung 2: Bildzeile für Aufgabe 4 mit Pixelwerten als Zahlen.

Aufgabe 4 — Padding am Bildrand - 3 + 8 + 3 + 3 + 4 + 4 = 25 Punkte - Programmieraufgabe
Erlaubte (Sub-)Pakete: `numpy`, `matplotlib`
In der Vorlesung wurden verschiedene Möglichkeiten der Behandlung des Bildrands im Rahmen der Faltung vorgestellt. Diese sollen nun erprobt und ihre Effekte an einem kleinen Beispiel ermittelt werden. Dazu wird im Rahmen dieser Aufgabe nur eine einzige Bildzeile (s. Abbildung 2) betrachtet.

1. Setzt die Bildzeile aus Abbildung 2 in Python mit Hilfe einer Liste um. Plottet die Bildzeile anschließend als Funktion, wie in Aufgabe 4 von Aufgabenblatt 6 vorgestellt. Der Index eines Listeneintrags soll dabei der x-Wert sein und der Listeneintrag selbst der y-Wert.
2. Definiert nun den 1D-Filterkern
$$k = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$
ebenfalls als Liste und normiert ihn, sodass die Summe der Einträge 1 ist. Implementiert nun selbst die 1D-Faltung zwischen der Bildzeile und dem Filterkern k. Umhüllt dafür zunächst die Bildzeile mit einer ausreichenden Anzahl Nullen an beiden Seiten (*zero-padding*). Fügt das Ergebnis ebenfalls in den oben erstellten Plot ein. Welche Auswirkung hat das zero-padding auf die Randbereiche?
Tipp: Mit dem Operator + kann man zwei Listen zu einer neuen Liste konkatenieren.
3. Ändert das Padding nun so, dass statt Nullen jetzt Neunen zum Padding genutzt werden. Faltet erneut die Bildzeile mit k und plottet das Ergebnis. Wie verändert es sich?
4. Variiert das bisherige konstante Padding nun so, dass jeweils der Wert am Rand der Bildzeile der Wert für das Padding ist (*replicate*). Wie verändert sich das Ergebnis der Faltung von Bildzeile und k hier und im allgemeinen Fall? Plotte zudem das Ergebnis.
5. Implementiert nun ein Padding der Variante *mirror*, wobei auch das Randelement der Bildzeile über den Rand gespiegelt wird:
$$cba|abcde|edc.$$
Plottet auch das Ergebnis der Faltung zwischen der Bildzeile und k mit diesem Padding und beschriebt kurz die Veränderung.
6. Nutzt zuletzt die Variante *reflection* des Paddings, die wie die Variante mirror funktioniert, wobei das Randelement der Bildzeile nicht über den Rand gespiegelt:
$$dcb|abcde|dcb.$$
Faltet nun die Bildzeile erneut mit k und plottet das Ergebnis. Diskutiert auch hier kurz die Auswirkungen des Paddings.

Zusatzaufgabe 5 — Bilder abstrahieren - 3 + 3 + 10 + 8 + 6 = 30 Punkte - Programmieraufgabe
Erlaubte (Sub-)Pakete: `numpy`, `skimage.io`, `skimage.filters`, `skimage.segmentation`, `matplotlib`
Durch die Faltung eines Bildes mit einem Gauß-Filter wird der Bildinhalt weichgezeichnet. Dies führt dazu, dass das Rauschen gemindert werden kann, wie in Aufgabe 3 erprobt. Ein weiterer Effekt ist das Verschwinden von Details im Bild. Diese Abstraktion des Bildes hat zwar oft negative Folgen, kann aber auch ausgenutzt werden, um Aufgaben zu vereinfachen.

1. Ladet erneut das Bild `einstein.png` aus dem Moodle herunter. Wendet auf das Bild einen Gauß-Filter an und visualisiert das Ergebnis. Wie weit könnt ihr das σ des Gauß-Filters erhöhen, bis ihr Einstein nicht mehr identifizieren könnt? Welche erkennbaren Elemente des Kopfes bleiben in der letzten identifizierbaren Stufe noch über? Welche Details des Kopfes gingen verloren?
2. Im Moodle findet ihr das Bild `ballon.png`. Ladet dieses Bild ebenfalls herunter und lasst es euch anzeigen. Welche Elemente und Strukturen im Bild fallen euch zunächst auf? Zeichnet nun das Bild ebenfalls mit Hilfe eine Gauß-Filters weich. Benutzt dabei hohe σs $(10, 20, \dots)$ und beschreibt, welche Elemente und Strukturen nach Anwendung des Filters noch erkennbar sind. Da es sich um ein Farbbild handelt, setzt bei der Funktion `skimage.filters.gaussian()` den Parameter `multichannel=True`.
3. Ladet zuletzt das Bild `gletscher.png` aus dem Moodle in Python und visualisiert es. Das Bild zeigt einen Teil eines Gletschersees umgeben von Moränen und Bergen. Im Gletschersee schwimmen zudem einige Eisblöcke. Nun soll versucht werden, die Pixel des Bildes in drei Bereiche zu unterscheiden. Pixel, die zu den Moränen und Bergen zu beiden Seiten des Sees gehören, sollen schwarz eingefärbt werden. Pixel, die zum See gehören, sollen in einem einheitlichen mittleren Grauton eingefärbt werden, wobei alle Objekte, die im See schwimmen, zum See gehören sollen. Zuletzt sollen die Pixel, die den Himmel bilden, weiß eingefärbt werden. Das Bild beinhaltet in der Folge nur noch drei Grauwerte. Setzt dies zunächst um, indem ihr die aus Vorlesung 4 und Aufgabenblatt 4 Aufgabe 1 bekannte Intensitätstransformation *intensity-level slicing* mit drei verschiedenen Intensitätsniveaus auf die Pixel anwendet. Sucht dazu zwei Grenzwerte, die Himmel und See bzw. See und Berge ungefähr voneinander trennen. Visualisiert das Ergebnis. Welche Probleme ergeben sich?
4. Zeichnet nun zuerst mit einem Gauß-Filter das Originalbild `gletscher.png` weich, um es zu abstrahieren, wie im Falle des Heißluftballons aus Aufgabenteil 2. Wendet jetzt eure Intensitätstransformation erneut auf das Ergebnis an. Wie verändert sich das Ergebnis? Experimentiert mit verschiedenen größeren σs und verschiedenen Grenzwerten für die Intensitätstransformation, um ein visuell gutes Ergebnis zu bekommen. Welche Fehler bleiben bestehen?
5. Zur besseren Einschätzung der Güte eures Ergebnis, könnt ihr die ermittelten Grenzen zwischen den Bereichen über das Originalbild legen. Nutzt dazu die Funktion `skimage.segmentation.mark_boundaries()` mit dem Originalbild als Parameter `image` und dem Ergebnis eurer Intensitätstransformation als Parameter `label_img`. Wie genau stimmen die von euch ermittelten Grenzen mit den tatsächlichen Grenzen im Bild überein? Findet eine Erklärung für mögliche Diskrepanzen.
   Hinweis 1: Das Ergebnis von `skimage.segmentation.mark_boundaries()` ist ein RGB-Bild. Ihr könnt dies ganz normal über `matplotlib.pyplot.imshow()` ohne den Parameter `cmap` anzeigen lassen.
   Hinweis 2: Die Funktion `skimage.segmentation.mark_boundaries()` erwartet in dem Parameter `label_img` ein Bild mit einem ganzzahligen Datentypen (bspw. `numpy.int`).