

SoSe 2023

# AG Verteilte Systeme (Tag 1)

## Programmierpraktikum

Daniel Schneider, Markus Vogelbacher, Hicham Bellafkir

25.08.2023

Die folgenden Aufgaben sollen Ihnen eine Einführung in die Themen Bilderverarbeitung (Aufgabe 1), Netzwerkprogrammierung (Aufgabe 2) sowie Machine Learning (Aufgabe 3) geben. Sie finden zu allen Aufgaben Quellcode-Vorlagen im Ilias.

### Aufgabe 1 - Bildfilterung (10 Punkte)

Digitale Bilder können auf vielfältige Weise verarbeitet werden. Eine verbreitete Methode ist die Filterung. Sie erzeugt ein neues Bild basierend auf einer pixelbasierten Manipulation des Originalbildes. Jedes Pixel im neuen Bild wird als Funktion von Pixeln des Originalbildes, normalerweise aus der Umgebung des Zielpixels, berechnet. In Abbildung 1 ist die Anwendung eines Gauß-Filter auf das Originalbild zu sehen. Dabei wird für jedes Pixel im Ergebnisbild eine Filtermatrix auf einen Bereich um das entsprechende Pixel im Originalbild angewendet. Diese Operation wird als Faltung (engl. *convolution*) bezeichnet. Weitere Beispiele für Bildfilter sind Kantendetektion, Schärfung oder Weichzeichner.

In dieser Aufgabe soll zunächst ein Bild eingelesen werden und dieses anschließend in Graustufen konvertiert werden. Auf dem resultierenden 1-Kanal 2D-Array sollen dann quadratische Filter angewendet werden. Die entsprechenden Funktionen (Faltung und Konvertierung in Graustufen) sollen dabei selbst implementiert werden, d.h. es sollen keine Libraries verwendet werden.

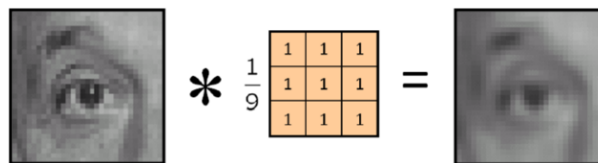


Abbildung 1: Glättung durch Gauß-Filter

- a) (1 Punkt) Implementieren Sie die Methode `loadImage` der Klasse `ImageProcessing`, die ein Bild von einem gegebenen Dateipfad als `BufferedImage` einliest und laden Sie mit der Methode das Eingabebild `example.jpg`.
- b) (1,5 Punkte) Implementieren Sie die Methode `convertToGrayscaleArray`, die ein `BufferedImage` in ein 2-dimensionales `int`-Array konvertiert, wobei jeder Wert im Array einen Grauwert repräsentiert. Es gibt verschiedene Möglichkeiten, ein RGB-Bild in Graustufen zu konvertieren. Nutzen Sie hier folgende Formel:

$$G = 0,299 \times R + 0,587 \times G + 0,114 \times B$$

Diese Verteilung der Farben spiegelt die entsprechende Farbempfindlichkeit der menschlichen Wahrnehmung wieder.

*Hinweis:* Jedes Bildpixel ist in `BufferedImage` als `int` gespeichert, hat also 32 Bit zur Verfügung. In Abbildung 2 ist die Aufteilung der Bit-Repräsentation eines RGB Pixel dargestellt. Jeweils ein Byte ist dem Alpha-, Rot-, Grün und Blaukanal zugeordnet und kann Werte zwischen 0 und 255 annehmen<sup>1</sup>. Mit `getRGB(x,y)` kann der RGB Wert eines Pixels im Bild ausgelesen werden. Z.B kann mit `getRGB(0,0)` auf den RGB Wert des Pixel links oben im Bild zugegriffen werden.

ALPHA								RED								GREEN								BLUE							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Abbildung 2: Zuordnung der Bit Positionen in `BufferedImage`

- c) **(1,5 Punkte)** Implementieren Sie die Methode `convertToBufferedImage`, die aus einem Integer-Array mit Werten für Graustufen ein `BufferedImage` vom Typ `BufferedImage.TYPE_INT_RGB` erzeugt. Implementieren Sie anschließend die Methode `saveImage`, die das konvertierte Bild in eine Datei schreibt.

Als nächstes sollen Sie auf das Graustufen-Bild Filtermasken (engl. *kernel*) anwenden. Filtermasken sind meist quadratische Matrizen in unterschiedlichen Größen. Für digitale Bilder wird die Filtermaske  $K$  mit einer diskreten 2-dimensionalen Faltung auf das Bild  $I$  angewendet<sup>2</sup>:

$$I^*(x, y) = \sum_{i=1}^n \sum_{j=1}^n I(x + a - i, y + a - j) K(i, j), \quad (1)$$

Dabei ist  $I^*(x, y)$  das Ergebnispixel und  $K(i, j)$  ist ein Element der Filtermaske.  $n$  bezeichnet die Breite bzw. Höhe der Filtermaske und  $a$  den Größenunterschied zwischen Eingabe- und Ausgabebild ( $a = n - 1$ ) Abbildung 3 veranschaulicht die Anwendung der Filtermaske auf eine bestimmte Position im Eingabebild.

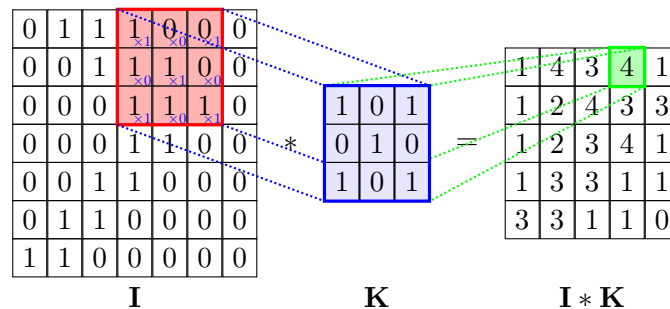


Abbildung 3: Faltung von Filtermaske  $K$  mit Eingabebild  $I$

<sup>1</sup>Der Alphakanal, der die Transparenz des Pixel festlegt, soll hier nicht berücksichtigt werden, d.h.  $A = 11111111_2 = 255_{10}$ .

<sup>2</sup>In der Praxis würde die Faltung durch eine Fourier-Transformation erfolgen, sodass sie in ein Produkt überführt wird. Dadurch reduziert sich der Aufwand zur Berechnung einer diskreten Faltung von  $\mathcal{O}(n^2)$  auf  $\mathcal{O}(n \cdot \log n)$ .

- d) (2 Punkte) Implementieren Sie in der Klasse `Kernel` die Funktion `convolve` die ein Eingabebild als 2D-Array (`int[][]`) entgegennimmt, die Faltung durchführt und das Ausgabebild ebenfalls als 2D-Array vom Typ `int[][]` zurückliefert. Beachten Sie, dass die Größe des Ausgabebildes nach der Faltung entsprechend der Filtergröße kleiner ist als die des Eingabebildes. Die Breite des Ausgabebildes ist dann  $w_{I_{out}} = w_{I_{in}} - w_K + 1$ , analoges gilt für die Höhe. Sowohl beim Array des Eingabebildes als auch beim Array des Kernel steht die erste Dimension für die `x`-Koordinate und die zweite für die `y`-Koordinate. Der Konstruktor der Klasse `Kernel` ist bereits implementiert: Um die Eingabe der Filtermatrix zu erleichtern wird die übergebene Filtermatrix dort transponiert. So ist es möglich einen Kernel, z.B.

$$k = \begin{bmatrix} 0 & 0 & 3 \\ 0 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

mit

```
new Kernel ( new double[][] {{0, 0, 3},
                             {0, 2, 0},
                             {1, 0, 0}} );
```

zu initialisieren. Das Transponieren dient hier also lediglich dazu, den Kernel intuitiver eingeben zu können, da wir die erste Dimension des Bild-Arrays (und des Kernels) als X-Achse und die zweite als Y-Achse definiert haben.

- e) (1 Punkt) Erstellen Sie JUnit Tests für die Klasse `Kernel`. Testen Sie ob die Funktion `convolve` korrekte Ergebnisse liefert, indem Sie prüfen, ob die in Abbildung 3 dargestellte Faltung auf dem rot gekennzeichneten Bildausschnitt tatsächlich den Wert 4 liefert und ob die Breite und Höhe des Ausgabebildes korrekt sind. Erstellen Sie bei Bedarf weitere Tests.
- f) (1,5 Punkte) Implementieren Sie die Methode `filter` der Klasse `ImageProcessing`, die eine Filtermatrix auf ein Bild anwendet.  
Wenden Sie verschiedene Filtermatrizen aus der Klasse `Kernels` auf das in Graustufen konvertierte Beispielbild an, konvertieren Sie es zurück in ein `BufferedImage` und schreiben Sie das Ergebnisbild jeweils in eine Datei. Überlegen Sie sich mindestens zwei weitere Filtermatrizen und wenden Sie diese auf das Bild an.
- g) (1,5 Punkte) Die Bilder sind nach der Faltung entsprechend der Kernelgröße kleiner als das Eingabebild. Überlegen Sie sich, wie dieses Problem behoben werden könnte. Implementieren Sie eine Lösung, bei der die Breite und Höhe des Eingabebildes erhalten bleiben. Erweitern Sie dazu die Methode `filter`.

## Aufgabe 2.1 - Sockets (5 Punkte)

Das Konzept eines Datenstroms erlaubt es Programmierer\*innen, sich bei der Programmierung auf die eigentliche Datenverarbeitung zu konzentrieren und von den Implementierungsdetails der Input-/Output-Operationen zu trennen. Somit können Programmierer\*innen, ihren Quellcode für unterschiedliche Arten von Datenströmen wiederverwenden, zum Beispiel für Lese- und Schreibzugriffe auf Dateien, aber auch für Netzwerkkommunikation.

Eine Verbindung über ein Netzwerk wird über so genannte Sockets hergestellt. Ein Socket stellt dabei ein Ende einer Verbindung zwischen zwei Endpunkten dar. Über den `InputStream` und den `OutputStream` des `Socket` werden Bytes von der Gegenstelle empfangen bzw. an diese gesendet.

In dieser Aufgabe sollen Sie mit Hilfe von Sockets ein einfaches, bidirektionales, interaktives und text-orientiertes Netzwerk-Protokoll<sup>3</sup> implementieren.

- a) **(5 Punkte)** Implementieren Sie den `TextClient`. Bauen Sie in der `main`-Methode eine Verbindung zu einem Server auf. Ihr Programm soll in einer Endlosschleife Benutzereingaben von der Konsole lesen, sie zum Server schicken und anschließend die Antwort des Servers auf der Konsole ausgeben.

Sie können zum Testen Ihres Clients unseren Echo-Server auf `dsgw.mathematik.uni-marburg.de` und Port 32823 verwenden. Der Echo-Server ist ein einfacher Server basierend auf Sockets, der Verbindungen akzeptiert und der die empfangenen Textnachrichten an den Sender zurückschickt.

---

<sup>3</sup>Kommunikation über Sockets sollte in der Praxis keinesfalls wie hier ungesichert verwendet werden und dient hier nur zur Veranschaulichung.