Lehrstuhl für Systemsimulation

Sommersemester 2025

Prof. Dr. Harald Köstler, Frederik Hennig, Michael Zikeli

Algorithmik kontinuierlicher Systeme
Aufgabenblatt 5 — Quantisierung und Downsampling

- Zu diesem Übungsblatt steht im StudOn-Kurs ein Zip-Verzeichnis mit Material bereit. Laden Sie dieses herunter und entpacken Sie es, bevor Sie mit den Aufgaben beginnen.
- Zu jeder Aufgabe gehört ein Python-Modul, dessen Name im Aufgabentitel gelistet ist. Füllen Sie die darin enthaltenen Funktions-Stubs mit Ihren Lösungen und geben Sie die Dateien über StudOn ab. Es handelt sich hierbei um **Einzelabgaben**. Sie können Ihre abgegebenen Dateien beliebig oft aktualisieren – nur die letzte abgegebene Version wird gewertet. **Laden Sie ihre Lösungen einzeln, oder als Zip-Archiv hoch, und ändern Sie nicht die Namen der Python-Dateien.**
- Das Material zum Übungsblatt enthält außerdem ein Jupyter Notebook zur interaktiven Entwicklung Ihrer Lösungen, sowie eine automatisierte Test-Suite. Jede Teilaufgabe wird anhand von einer Reihe an Tests automatisch bewertet. Die *öffentlichen* Testfälle können Sie entweder in dem mitgelieferten Notebook, oder durch das Skript `blatt**_tests.py` auf der Konsole ausführen. Denken Sie daran, dass das Bestehen der öffentlichen Tests keine Garantie für Korrektheit ist.

Auf diesem Aufgabenblatt werden wir uns mit der Farbquantisierung von RGB-Bildern mithilfe des *Median Cut*-Algorithmus, sowie *K-Means Clustering* befassen.

Aufgabe 1 — Hilfsfunktionen (6 Punkte) `imgutils.py`

Für die beiden Quantisierungsalgorithmen in Aufgaben 2 und 3 werden wir ein paar Hilfsfunktionen benötigen. Diese zu implementieren ist Thema dieser Aufgabe.

a) Pixelliste erzeugen: (1.5 Punkte) Schreiben Sie eine Funktion `image2pixels`, die ein Bild in eine Liste von Pixeln konvertiert. Das übergebene Bild ist ein dreidimensionales Array der Form `[Zeile, Spalte, RGB]` mit den Abmessungen $(y_{\max}, x_{\max}, 3)$. Die drei Farbkanäle werden für jeden Pixel durch eine natürliche Zahl zwischen 0 und 255 dargestellt. In der Pixelliste soll jeder Pixel als Tupel `(R, G, B, X, Y)` repräsentiert werden; hier sind `R`, `G` und `B` die drei Farbkanäle, und `X` und `Y` die Position des Pixels im Bild. Die Pixel sollen zeilenweise in der zurückgeggebenen Liste liegen.

b) Pixelliste zurückkonvertieren (1.5 Punkte) Schreiben Sie nun die Funktion `pixels2image`, welche aus einer Liste von `(R, G, B, X, Y)`-Tupeln das dazugehörige Bild rekonstruiert. Dieses soll als dreidimensionales NumPy-Array mit dem Elementdatentyp `uint8` zurückgegeben werden. Die Reihenfolge der übergebenen Pixel in der Liste darf dabei keine Rolle spielen.

c) Kleinster umgebender Quader (1.5 Punkte) Schreiben Sie die Funktion `bounding_box`, welche den kleinsten umgebenden Quader *um die Farbwerte* einer Liste von `(R, G, B, X, Y)` Tupeln berechnet. Der Quader soll in einem geschachtelten Tupel der Form $((R_{\min}, R_{\max}), (G_{\min}, G_{\max}), (B_{\min}, B_{\max}))$ gespeichert werden.

d) Durchschnittsfarbe (1.5 Punkte) Schreiben Sie die Funktion `color_average`, welche eine Liste aus `(R, G, B, X, Y)` Tupeln übergeben bekommt, den Mittelwert aller Farben darin bestimmt und diese Mittelwerte als `(R, G, B)` Tupel zurückgibt. Verwenden Sie dabei die Python Funktion `round`, um den Mittelwert in jedem Farbkanal von einer Gleitkommazahl in eine Ganzzahl zwischen 0 und 255 zu konvertieren.

Aufgabe 2 — Median Cut (7 Punkte) **mcut.py**

In dieser Aufgabe implementieren Sie den rekursiven Median-Cut-Algorithmus, um den Farbraum eines Bildes zu quantisieren. Arbeiten Sie dazu die Teilaufgaben der Reihe nach durch, und verwenden Sie insbesondere Ihre Funktionen aus Aufgabe 1 wieder. In dem zu diesem Übungsblatt gehörigen Jupyter Notebook finden Sie zudem ein Setup, um den Algorithmus an echten Bildern zu testen.

Abbildung 1: Originalbild.

Abbildung 2: Median-Cut-Quantisierung mit 16 Farben im RGB-Farbraum.

a) Schnittdimension (2 Punkte) Implementieren Sie eine Hilfsfunktion `cut_dimension`, welche den Index der längsten Kante des umgebenden Farbquaders zurückgibt. Der Quader wird in dem Format aus Aufgabe 1 c) erwartet. Der zurückgegebene Index soll bei Rot den Wert 0, bei Grün den Wert 1 und bei Blau den Wert 2 liefern. Bei Gleichstand wird Rot und danach Grün bevorzugt.

b) Rekursive Unterteilung (4 Punkte) Implementieren Sie nun die Funktion `recursive_median_cut`, die den Hauptteil des Median-Cut-Algorithmus rekursiv implementiert. Die Funktion erwartet eine Liste von `(R, G, B, X, Y)` Tupeln und eine nicht-negative Ganzzahl `N`, die angibt, wie viele rekursive Schritte noch durchgeführt werden sollen. Ist `N` $= 0$, oder der Farbraum bereits vollständig unterteilt, wird die Rekursion abgebrochen und alle Elemente auf die Durchschnittsfarbe gesetzt. Optional darf der maximale umgebende Farbquader übergeben werden, falls dieser bereits bekannt ist.

Die Unterteilung des Farbraumes wird in jedem rekursiven Schritt entlang des Indexes von Aufgabe a) durchgeführt, d.h. die Liste der Tupel wird bzgl. der längsten Seite der Bounding Box sortiert und dann in der Mitte geteilt (Berechnung des Median). Bei einer ungeraden Länge wird beim abgerundeten Index geteilt. Der Datentyp `list` verfügt in Python über eine Methode `sort(key=func)`, die die Elemente aufsteigend sortiert. Eine Lambda-Funktion kann ihnen dabei helfen den richtigen `key` auszuwählen.

c) Anwendung auf NumPy-Arrays (1 Punkt) Die Funktion `recursive_median_cut` aus der vorherigen Teilaufgabe arbeitet auf einer Pixelliste; wir wollen den Median-Cut-Algorithmus aber auf Bilder anwenden, die als NumPy-Arrays vorliegen.

Implementieren Sie die Funktion `median_cut`, welche ein Bild als NumPy-Array im Format $M \times N \times 3$ erhält. Konvertieren Sie das Bild in eine Pixelliste; wenden Sie `recursive_median_cut` auf die Pixel an, und konvertieren Sie das Ergebnis zurück in ein NumPy-Array.

Aufgabe 3 — K-Means Clustering (7 Punkte) **kmeans.py**

In dieser Aufgabe implementieren Sie den Lloyd-Algorithmus zur Bestimmung eines K-Means Clustering. Der Ablauf des Algorithmus wird schematisch in Abbildung 3 gezeigt. Der Algorithmus wird mit einer initialen Quantisierung aufgerufen. Basierend auf dieser wird jeder Farbwert des Bildes einem Repräsentanten der Quantisierung zugewiesen. Somit wird der Farbraum in k Bereiche unterteilt. Um den Fehler der Quantisierung zu verringern, wird nun ein neuer Repräsentant für einen Bereich berechnet, indem der Mittelwert der Farbwerte bestimmt wird. Dieses Vorgehen (Zuweisen zu Repräsentanten, Neuberechnung der Repräsentanten) wird solange wiederholt, bis die Segmentierung des Farbraums konvergiert ist.

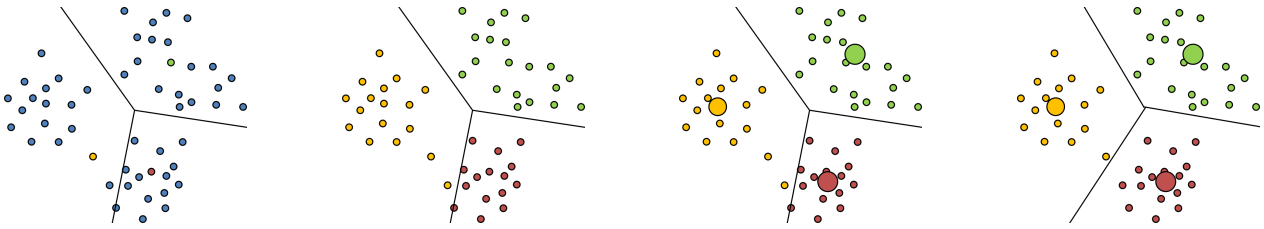

Abbildung 3: Ablauf des k-Means-Algorithmus für $k = 3$

Die zwei zentralen Datenstrukturen dieses Algorithmus sind Cluster und Mittelwerte. Cluster werden durch eine Liste von Pixellisten repräsentiert. Mittelwerte werden durch eine Liste von `(R, G, B)` Tupeln repräsentiert. Alle Funktionen die Pixel und Cluster verarbeiten müssen, soweit möglich, die Reihenfolge der Eingabedaten beibehalten.

a) Durchschnitte (2 Punkte) Schreiben Sie eine Funktion `compute_means`, die für gegebene Cluster die zugehörigen Mittelwerte berechnet. Vewenden Sie hierzu die Funktion `imgutils.color_average`.

b) Cluster Berechnen (3 Punkte) Schreiben Sie eine Funktion `compute_clusters`, die für eine gegebene Pixelliste und gegebene Mittelwerte die zugehörigen Cluster zu jedem Mittelwert berechnet.

Für gegebene Mittelpunkte $\mathbf{m}_1, \ldots, \mathbf{m}_k$ ergeben sich die zugehörigen Cluster $C_1, \ldots, C_k$ folgendermaßen:

$$C_i = \left\{ \mathbf{x} : \left\| \mathbf{x} - \mathbf{m}_i \right\|^2 \leq \left\| \mathbf{x} - \mathbf{m}_j \right\|^2 \text{ für alle } j \neq i \right\} \tag{1}$$

Falls ein Pixel zu mehreren Clustermittelpunkten den gleichen Abstand besitzt, wird er dem Cluster mit dem niedrigeren Index zugeordnet.

c) Gemittelte Pixelliste (2 Punkte) Schreiben Sie eine Funktion `averaged_pixels`, die gegebene Cluster und Mittelwerte in eine Pixelliste umwandeln, bei der jeder Pixel eines Clusters die Farbe des zugehörigen Mittelwerts annimmt.