Berliner Hochschule für Technik

Prof. Dr.-Ing. Kristian Hildebrand

Maschinelles Sehen
Filter, OpenCV Basics and K-Means (10 Points)

SoSe 2025

Ziel dieser Übungsaufgabe ist eine tiefere Einarbeitung in NumPy und OpenCV und Bildverarbeitung. Laden Sie den vorgegebenen Skelett-Sourcecode von Moodle und implementieren Sie die notwendigen Funktionen. Erweitern und verändern Sie den Sourcecode nach Belieben. Das ist nur eine anfängliche Unterstützung.

Wichtig: Für die erfolgreiche Bearbeitung der Aufgabe ist es unbedingt erforderlich, dass Sie sich selbst mit OpenCV, Python und Numpy vertraut: z.B. https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html oder https://www.stavros.io/tutorials/python/ oder https://numpy.org/devdocs/user/quickstart.html. Datentypen und -formate sind bei der Benutzung von NumPy und OpenCV sehr wichtig. OpenCV Bilder werden häufig im *uint8: 0 - 255* geladen, bearbeitet und gespeichert. Viele Berechnungen finden dann aber im *float* Format statt. Umrechnungen können u.a. mit NumPy *np.float32(image)* durchgeführt werden.

Aufgabe 1: Numpy Basics (3 Punkte)

Die Bibliothek *Numpy* ist sehr umfangreich und wir werden auch nicht jede Funktionalität brauchen, aber es ist wichtig mit den grundlegenden Datenstrukturen vertraut zu sein. Ich habe hier eine Liste von kleinen Aufgaben zusammengestellt, die Sie in der Datei *numpy_basics.py* beantworten sollen. **Suchen Sie sich insgesamt 16 Sterne zusammen und implementieren Sie diese**. Das ist häufig ein Einzeiler oder Zweizeiler. Lesen Sie sich selbstständig in die Dokumentation ein und versuchen Sie die Fragen mit den notwendigen Numpy-Funktionen zu beantworten. Sollten Sie keine Funktion finden, implementieren Sie Lösung selbst in einer kleinen Funktion. Nutzen Sie dafür die vorgegebene Struktur der Datei. **Bitten geben Sie die Lösung jeweils auf der Konsole aus (mittels print), z.B. '(a) [...]' . Sonst werde ich die Aufgabe mit 0 Punkten bewerten.**

a) (*) Erzeugen Sie ein 1D-Numpy-Array mit Ganzzahlen von 5 bis 100 (einschließlich 100), die durch 5 teilbar sind (geht in einer Zeile).

b) (*) Kehren Sie die Reihenfolge der Elemente in einem gegebenen Vektor um (verwenden Sie Array-Slicing. Lesen Sie sich bitte ein, was das ist - https://numpy.org/doc/2.2/user/basics.indexing.html).

c) (*) Erzeugen Sie eine 5x5-Matrix mit Ganzzahlen von 0 bis 24, wobei die Zahlen zeilenweise gefüllt werden (links oben bis rechts unten).

d) (*) Erzeugen Sie eine 6x6-Matrix mit Zufallswerten im Bereich [0, 100). Finden Sie das Minimum, das Maximum und normalisieren Sie die Matrix, sodass alle Werte im Bereich [0, 1] liegen.

e) (*) Multiplizieren Sie eine 5x3-Matrix mit einer 3x4-Matrix. Was ist die Form (Shape: row:column) der resultierenden Matrix?

f) (*) Erstellen Sie ein Numpy-Array mit Werten von -5 bis 15. Erzeugen Sie ein neues Array, das nur die positiven geraden Zahlen enthält. Verwenden Sie boolesche Indizierung.

g) (*) Erzeugen Sie eine 6x6-Matrix mit den Werten 1 bis 36 und extrahieren Sie daraus eine 2x2-Untermatrix aus der Mitte der Matrix (Array-Slicing again).

h) (*) Erzeugen Sie ein 1D-Numpy-Array mit Werten von 0 bis 30. Ersetzen Sie alle Werte zwischen 10 und 20 (einschließlich) durch ihr Negatives.

i) (*) Erzeugen Sie ein zweidimensionales Array mit Zufallswerten und berechnen Sie die Summe aller Elemente sowie die Summe entlang der Achsen (Zeilen- und Spaltensummen).

j) (*) Erzeugen Sie eine Matrix M der Größe 3x5 und einen Zeilenvektor v der Größe 5. Multiplizieren Sie v mit M (Achten Sie auf die richtige Form!).

k) (**) Erzeugen Sie eine 6x6-Matrix mit fortlaufenden Werten von 1 bis 36. Geben Sie die Werte aller ungeraden Spalten (Index 1, 3, 5) aus.

l) (**) Erzeugen Sie eine 2D-Punktwolke mit 100 Punkten und setzen Sie dafür einen festen Zufalls-Seed (*np.random.seed(42)*). Berechnen Sie den Mittelpunkt und anschließend die Kovarianzmatrix der zentrierten Daten.

m) (**) Erzeugen Sie 50 zufällige 2D-Koordinaten im kartesischen System (10x2 Matrix). Wandeln Sie diese Koordinaten in Polarkoordinaten um (Radius r und Winkel θ).

n) (**) Erzeugen Sie zwei passende Arrays für eine Matrixmultiplikation mit dem `@`-Operator und führen Sie die Operation durch. Überprüfen Sie das Ergebnis mit `np.dot()`.

Aufgabe 2: Computer Vision Basics + OpenCV (3 Points)

Vorbemerkung: OpenCV verwendet standardmäßig den BGR-Farbraum statt RGB - beim Einlesen von Bildern mit cv2.imread() sind die Farbkanäle also vertauscht. Farbwerte liegen typischerweise als uint8 im Bereich 0-255 vor; bei Verarbeitung als float (z.B. für Normalisierung oder Machine Learning) werden sie meist auf 0-1 skaliert. **Wichtig:** cv2.imshow() erwartet Bilder im uint8-Format - bei float-Bildern (z.B. 0-1) kann die Anzeige fehlschlagen oder schwarz wirken, daher sollten sie vor dem Anzeigen zurück in uint8 (z.B. durch Multiplikation mit 255 und astype(np.uint8)) umgewandelt werden.

2.1. (2 Punkt + 1 Zusatzpunkt) Implementieren Sie die Faltung eines Bildes mit einer Filtermaske (Convolution) auf einem Graustufenbild **ohne die Funktionen in OpenCV**. Nutzen Sie als Filtermaske ein Gaussianblur und die in der Vorlesung besprochenen Sobelfilter (x- und y-Richtung) zur Berechnung der Bildgradienten. Berechnen Sie daraus ein Bild, dass die Gradientstärken darstellt (Magnitude of Gradients). Beachten Sie bitte das Padding und, dass die Größe des Eingabe- und Ausgabebilders gleich ist und die Ränder nicht schwarz sind 1. Hinweis: Versuchen Sie die Implementierung so effizient wie möglich zu machen. Die Implementierung von OpenCV läuft dann in Echtzeit - das ist nicht notwendig. **Zusatzpunkt:** Wenn Sie die Faltung als separierbare Faltung implementieren bekommen Sie einen Zusatzpunkt. Allerdings müssen Sie mir das auch erklären können.

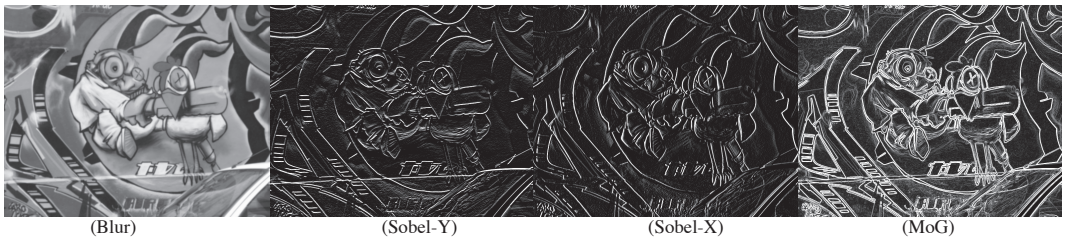

Abbildung 1: Ergebnisse der Faltung.

2.2. (1 Punkte) Experimentieren Sie mit folgenden - in der Vorlesung angesprochenen - Bildverarbeitungsalgorithmen in OpenCV:

a) Wechsel von Farbräumen (HSV, LAB, YUV)

b) Recherchieren Sie die Theorie hinter *Histogram Equalization* und implementieren Sie dann diesen Algorithmus zur Kontrastverbesserung, um die visuelle Qualität und Detailgenauigkeit zu erhöhen.

c) Adaptives Thresholding bitte in den beiden folgenden Varianten Gaussian-Thresholding und Otsu-Thresholding.

d) Canny-Kantenextraktion

e) Verwenden Sie das Numpy-Modul *fft*, um eine Fast Fourier-Transformation auszuführen und die Frequenzmerkmale des Bildes zu analysieren. Verwenden Sie dazu das Ergebnis des Aufrufs *fft*. Beachten Sie, dass sich die Null-Frequenz-Komponente in der linken oberen Ecke befindet. Um diese Komponente zu zentrieren, verschieben Sie das Ergebnis sowohl in horizontaler als auch in

vertikaler Richtung. Verwenden Sie dazu die Funktion *np.fft.fftshift()*. Durch die zentrale Positionierung wird die Analyse vereinfacht. Bestimmen Sie anschließend das Magnitudenspektrum der Frequenztransformation, indem Sie nur die absoluten Werte verwenden und diesen Bereich mit *np.log* komprimieren und mit 20 multiplizieren. Nun können Sie das Bild filtern, indem Sie Teile der Frequenzamplituden entfernen und es vom Frequenzbereich zurück in den räumlichen Bereich transformieren. Speichern Sie Ihre Bilder in separaten Dateien. *Wichtig: Diese Teilaufgabe muss nicht mit ihrer Webcam laufen, nutzen Sie ein Bild Ihrer Wahl!*

Nutzen Sie dafür die vorgegebene Datei *opencv_experiments.py* und *fourier.py* sodass man auf Tastendruck die oben genannten Funktionen wechseln kann.

Aufgabe 3: K-Means for color quantization (4 Punkte)

Abbildung 2: k-means based color clustering. Color quantization with growing $k = 8...64$

Methoden des Maschinellen Lernens spielen eine wichtige Rolle in vielen Anwendungen der Computer Vision. Während des gesamten Kurses werden wir einige neuere Algorithmen diskutieren. Viele *klassische* Probleme wie Feature Matching, Bildsegmentierung oder Farbquantisierung (und viele andere mehr) hängen jedoch stark von gut verstandenen ML-Methoden ab. Ein Beispiel ist der k-Means-Algorithmus. In dieser Übung müssen Sie ein einfaches Schema zur Farbquantisierung (Clustering) auf der Grundlage von k-Means implementieren. Dabei wird eine vorher bekannte Anzahl von k-Gruppen aus einem Satz ähnlicher Datenpunkte gebildet. Diese Standardmethode ist eine der am häufigsten verwendeten Techniken zur Gruppierung von Datenpunkten, z.B. in unserem Fall RGB-Pixelwerte (siehe Abbildung 2 für weitere Einzelheiten. **Bitte lesen Sie zum besseren Verständnis der Darstellung die Bildunterschrift.**).

Die Idee der *Farbbasierte Segmentierung*, die auf dem k-means-Algorithmus basiert, besteht darin, die Farben im Bild zu quantisieren / segmentieren, um ähnliche Farben im Bild zu einer Gruppe zusammenzufassen. Die Ähnlichkeit zwischen Pixelwerten kann durch den euklidischen Abstand zwischen RGB-, LAB- oder HSV-Werten bestimmt werden.

Implementieren Sie auf der Grundlage des gegebenen Quellcodes und der Kommentare den k-means-Algorithmus und testen Sie ihn auf verschiedene k und verschiedene Farbräume, wie in Abbildung 2 gezeigt. **Berechnen Sie den Gesamtfehler für jedes Ergebnis und drucken Sie ihn auf der Kommandozeile aus.**

Bitte interpretieren Sie Ihre Ergebnisse des Clustering und beantworten Sie die folgenden Fragen **(bitte reichen Sie Ihre Antwort als zusätzliche .txt- oder .md-Datei ein)**:

a) Was sind die Probleme dieses Clustering-Algorithmus?

b) Wie kann ich die Ergebnisse verbessern?

Implementieren Sie eine Algorithmusverbesserung. (0,5 von 4 Punkten) Hinweis: Wenn Sie es für nötig halten, können Sie den Standard-Quellcode gerne ändern. Bitte verwenden Sie für Ihre Implementierung die Datei kmeans.py.

Abgabe Dies ist *Übungsaufgabe 1*; die Bearbeitungszeit der Aufgabe ist für ca. 3 Wochen ausgelegt. **Die Abgabe soll via Moodle zum angegebenen Zeitpunkt erfolgen**. Geben Sie bitte jeweils nur eine einzige .zip-Datei mit den Quellen Ihrer Lösung ab. **Bitte fügen Sie alle notwendigen Bilder ein, sodass jede Aufgabe direkt ausführbar ist. Andernfalls behalte ich mir Punktabzüge vor.** Verspätete Abgaben werden mit einem Abschlag von 3 Punkten je angefangener Woche Verspätung belegt.

Bitte seien Sie vorbereitet, dass ich stichprobenartig nach einer Demonstration und Erläuterung Ihrer Lösung in der nächsten Übung nach dem Abgabetag frage. Die Qualität Ihrer Demonstration ist, neben dem abgegebenen Code, ausschlaggebend für die Bewertung. Viel Spaß und Erfolg bei der Bearbeitung.