Zusammenfassung Image Processing & Computer Vision

Stephan Stofer
Hochschule Luzern - Informatik
FS2020
stephan.stofer@hslu.ch
SW01 von Adrian Franz Willi, adrian.willi@stud.hslu.ch

February 27, 2020

Contents

1 Punktbildoperation

1.0.1 Bildverarbeitung

- **Computergrafik:** Erzeugung von realistischen Bildern aus der Beschreibung von Objekten. Beispielsweise werden realistische Bilder erzeugt und dann als Trainingsdaten für Machine Learning Applikationen verwendet.
- **Bildverarbeitung:** Erkennung von Objekten oder Szenen aus Bildern.

1.0.2 Bildverarbeitung auf verschiedenen Stufen

- Computer Vision (High Level Vision): Interpretation des Bildes, Erkennung von Objekten.
- Image Processing (Low Level Vision): Bildverbesserung (Kontrastkorrektur, Rauschunterdrückung), Erkennung von Low Level Features (Kanten, Linien,...)

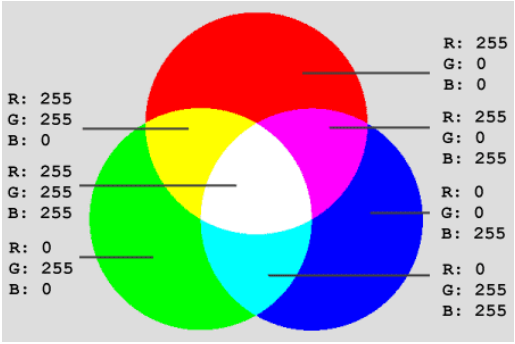
1.1 Bildverarbeitungs-Methoden

Bevor Methoden angewendet werden, werden Bilder oft vorverarbeitet. D.h. Informationen verstärken oder Störungen reduziert.

Unterscheidung zwischen:

- **Punktoperatoren:** Operation auf einem Punkt. Nur der Grau- (oder Farb-) wert wird transformiert.
- **Nachbarschaftsoperatoren:** Ergebnis hängt von der Umgebung des Punktes ab. zB. 3x3 Umgebung, Filteroperationen, etc.

RGB Übersicht R(Rot)G(Grün)B(Blau). Werte gegen 0 werden dunkler, Werte gegen 255 hingegen heller.



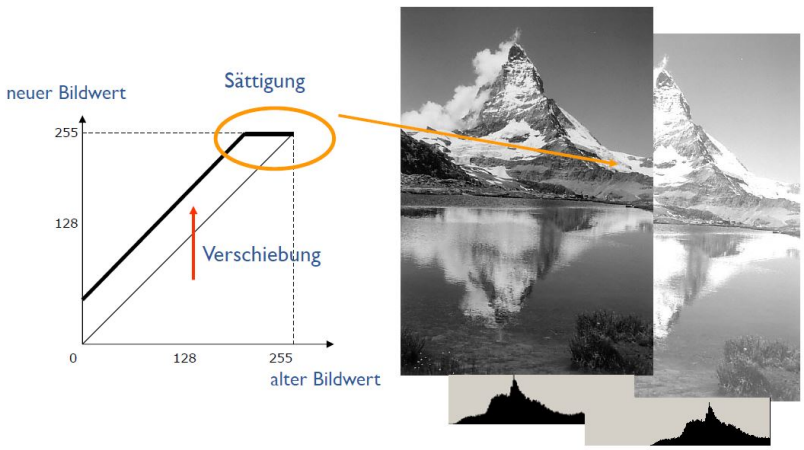
OpenCV liest Bilder in BGR-Format. Mit image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) können Bilder in das RGB-Format konvertiert.

1.2 Punktbildfunktionen

Veränderung der Intensitätswerte durch eine Funktion. Beispiele: Schwellwert, Helligkeitsveränderung, Kontrast Korrektur, Gamma Korrektur.

1.2.1 Helligkeitsveränderung

Im Beispiel wird Helligkeit erhöht, erkennbar durch die Verschiebung der Kurve nach oben. Die Werte im Histogramm verschieben sich somit nach rechts.



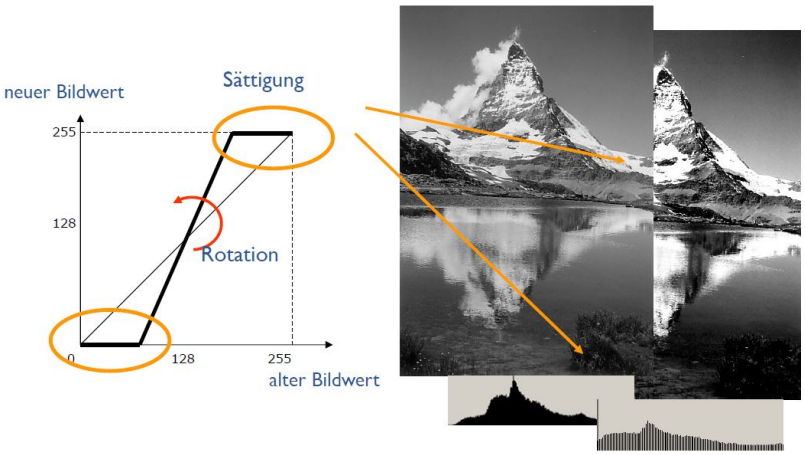
OpenCV: Methode zum verändern der Helligkeit

```
In [43]:
def f(image, gamma):
    # change function to make pixesl brighter (or darker)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    image[:,:,2] = image[:,:,2] + gamma
    image = cv2.cvtColor(image, cv2.COLOR_HSV2BGR)


    return image
```

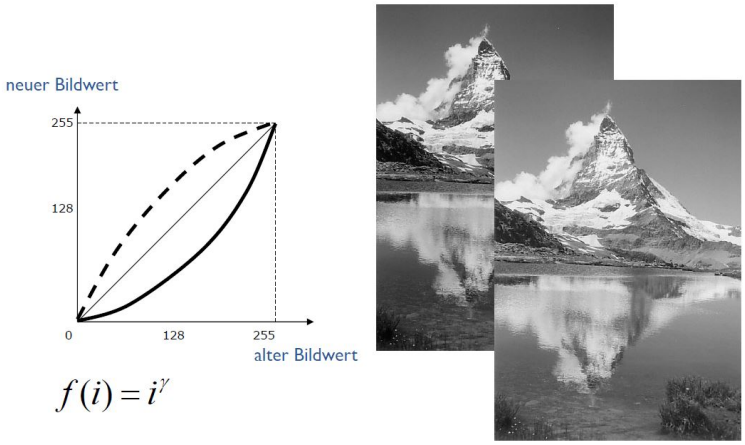
1.2.2 Kontrastveränderung

Wenn die Steigung > 1 ist, dann nimmt die Kontrastveränderung zu. Falls nicht, nimmt sie ab. Optimal ist eine Kurve mit einer S-Form, da so am wenigsten Informationen verloren gehen.



1.2.3 Gamma-Korrektur

Die vom Menschen empfundene Helligkeit steigt in dunklen Bereichen steiler und in hellen weniger steil an. Die Stevenssche Potenzfunktion ordnet dem menschlichen Auge dabei ein Gamma von ca. 0,3 bis 0,5 zu. Soll das Helligkeitssignal eines Anzeigegerätes, beispielsweise eines Monitors, linear wahrgenommen werden, muss es daher mit dem reziproken des obigen Gammawerts (ca. 3,3 bis 2) vorverzerrt werden, damit sich beide Nichtlinearitäten für den Betrachter am Ende wieder aufheben. Ein typischer Wert für Bildschirme ist etwa ein Gamma von 2,2



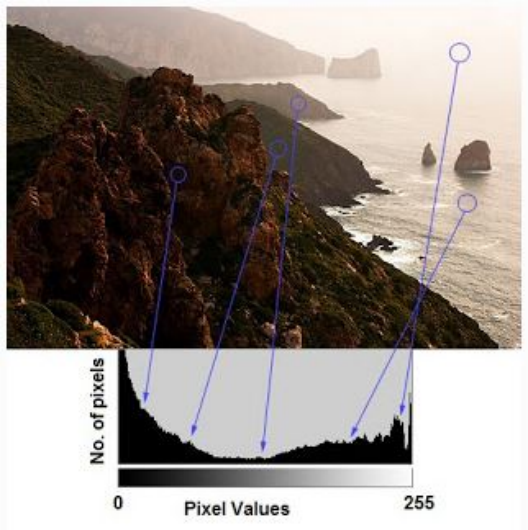
Je nach Grösse des Exponenten γ(gamma) unterscheidet man 3 Fälle:

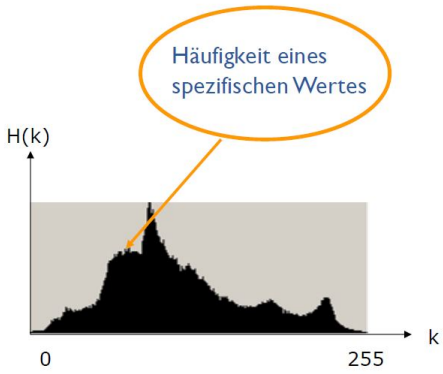
- $\gamma = 1$: die Abbildung ist linear.

- $\gamma < 1$: Die Abbildung ist konkav. Kleine Eingabewerte werden stark gespreizt, grosse gestaucht. (gepunktete Linie)
- $\gamma > 1$: Die Abbildung ist konvex. Kleine Eingabewerte werden gestaucht, grosse gespreizt. (durchgezogene Linie)

1.3 Histogramm

Farbverteilung des Bildes: Wie häufig kommt welche Farbe/Helligkeit vor. So kommen im ersten Beispiel unten häufig sehr dunkle Werte vor, was auch gut erkennbar im Histogramm ist.



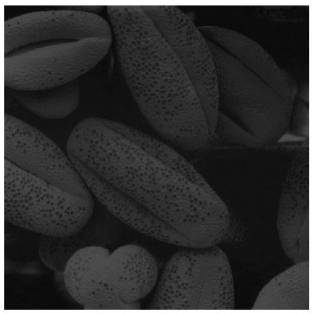


OpenCV cv2.calcHist(images/sw01, channels, mask, histSize, ranges[, hist[, accumulate]])

- **1.** images/sw01 : it is the source image of type uint8 or float32. it should be given in square brackets, ie, "[img]".
- **2.** channels : it is also given in square brackets. It the index of channel for which we calculate histogram. For example, if input is grayscale image, its value is [0]. For color image, you can pass [0],[1] or [2] to calculate histogram of blue,green or red channel respectively
- **3.** mask : mask image. To find histogram of full image, it is given as "None". But if you want to find histogram of particular region of image, you have to create a mask image for that and give it as mask. (I will show an example later.)
- **4.** histSize : this represents our BIN count. Need to be given in square brackets. For full scale, we pass [256].
- **5.** ranges : this is our RANGE. Normally, it is [0,256]

1.3.1 Histogrammausgleich

Ziel eines Histogrammausgleich ist die Transformation der Grauwerte, so dass das Histogramm des neuen Bildes möglich ausgeglichen ist. Die Funktion wird in Bildverarbeitung-Tools oft "Auto Contrast" genannt.



original

nach Histogrammausgleich

Histogrammausgleich Berechnung

H_{neu} = konstant

$f'(i) = \frac{1}{c} H_{alt}(i)$

$f(i) = \frac{1}{c} \int_0^i H_{alt}(j)\, dj$

0 1 1 2 2
2 1 1 2 2
4 5 6 7 7
8 8 7 7 7
8 8 9 9 9

Pixel Intensität	Histogramm alt / No. of pixels	$\sum_{0}^{i} H_{alt}(i)$ / cdf(i) Kumulierte Häufigkeit	Ausgleich $f(i) = \frac{9}{c} \cdot cdf(i)$	$H_{neu}(i)$ gerundet
0	1	1	0.36	0
1	4	5	1.8	2
2	5	10	3.6	4
3	0	10	3.6	4
4	1	11	3.96	4
5	1	12	4.32	4
6	1	13	4.68	5
7	5	18	6.48	6
8	4	22	7.92	8
9	3	25	9	9

Anzahl Zahlen: $c = 25$
Höchster Wert: $x = 9$
Normalisierung: $\frac{x}{c} = \frac{9}{25} = 0.36$

1.4 Morphologische Operationen

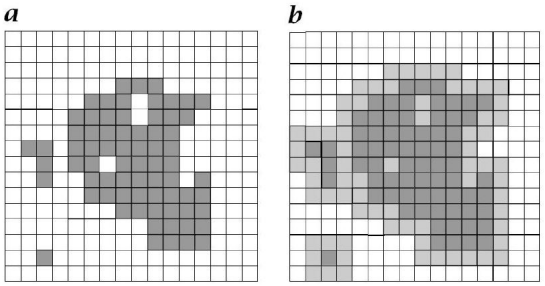
- Morphologische Operation werden zur Bearbeitung von Binärbildern verwendet
- Binärbilder können zum Beispiel mit dem Setzen eines Schwellwerts (engl. Threshold) erzeugt werden

- Die entstehenden Regionen müssen jedoch häufig weiterverarbeitet werden.
- Morphologische Operationen benötigen zwei Inputs. Das original Bild und ein Strukturelement (kernel)
- Morphologische Operationen können durch Mengenoperationen dargestellt werden. Dabei wird das Bild mit einem Strukturelement verknüpft, dieses ist als Matrix definiert.

1.4.1 Dilate

Verdickt das Objekt. Wenn mind. 1 Pixel vom Bild unter dem Strukturelement ist. Es vergrössert das Objekt im Vordergrund und entfernt so beispielsweise weisse Löcher.

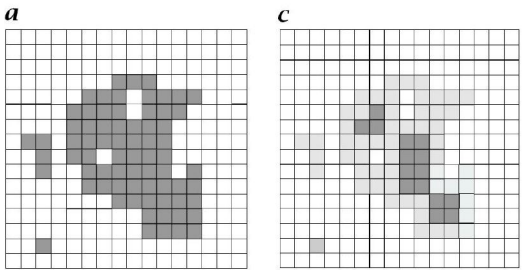
$$A \oplus S = \{x \mid A \cap S_x \neq \varnothing\}$$



1.4.2 Erode

Verdünnt das Objekt. Bei Erode muss das ganze Strukturelement über den Pixel des Bildes sein. Ansonsten werden diese entfernt.

$$\begin{aligned} A \ominus S_x &= \{x \mid \bar{A} \cap S_x = \emptyset\} \\ &= \{x \mid A \supseteq S_x\} \end{aligned}$$



1.4.3 Closing/Opening

Closing Dilate gefolgt von Erode. Schliesst das Objekt (eliminiert Löcher)

Opening Erode gefolgt von Dilate. Öffnen (separiert) Objekte. Ist sinnvoll um Störungen aus Bilder zu entfernen

1.5 Connected Component Labeling

Auch bekannt als connected component analysis, blob extraction, region labeling, blob discovery oder region extraction. Dabei geht um eine Anwendung aus der Graphentheorie, wo Teilmengen von "connected components" eindeutig markiert (labeled) aufgrund einer gegebenen Heuristik.

- Wie viele Objekte sind auf einem (binären) Bild zu sehen?
- Wie "gross" (Anzahl Pixel) sind Objekte?
- Welche Regionen sind zusammenhängend?

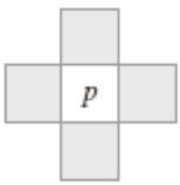
1	1	1	0	0	0	0	0
1	1	1	0	1	1	0	0
1	1	1	0	1	1	0	0
1	1	1	0	0	0	1	0
1	1	1	0	0	0	1	0
1	1	1	0	0	0	1	0
1	1	1	0	0	1	0	0
1	1	1	0	0	0	0	0

Ausgangsbild (binär)

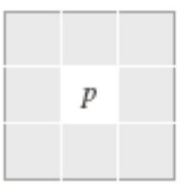
1	1	1	0	0	0	0	0
1	1	1	0	2	2	0	0
1	1	1	0	2	2	0	0
1	1	1	0	0	0	4	0
1	1	1	0	0	0	4	0
1	1	1	0	0	0	4	0
1	1	1	0	0	3	0	0
1	1	1	0	0	0	0	0

Resultat: gefundene Regionen mit Labels

4er/8er Zusammenhang



4-er Nachbarschaft eines Pixels



8-er Nachbarschaft eines Pixels

1.6 Übungen

1.6.1 Spiegelung

Spiegelung mit OpenCV

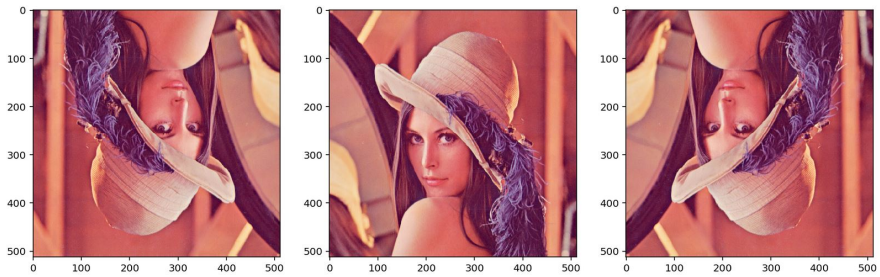
Spiegeln sie das Bild an der vertikalen und horizontalen Achse unter Benutzung von Funktionen aus OpenCV und stellen Sie die Bilder dar.

Bemerkungen
Die Funktion cv2.flip(image, number) nimmt zwei Parameter entgegen. Nebst dem Bild können mit folgenden Angaben die Richtung der Spiegelung angegeben werden:

0, for flipping the image around the x-axis (vertical flipping)
> 0 for flipping around the y-axis (horizontal flipping)
< 0 for flipping around both axes

```
image_mirror_vert = cv2.flip(image, 0)
image_mirror_hor = cv2.flip(image, 1)
image_mirror_both = cv2.flip(image, -1)
plt.figure(figsize=(15,  10))
plt.subplot(1, 3, 1)
plt.imshow(image_mirror_vert)
plt.subplot(1, 3, 2)
plt.imshow(image_mirror_hor)
plt.subplot(1, 3, 3)
plt.imshow(image_mirror_both)


plt.show()
```



Alternative Darstellung mit numpy
plt.subplot(1,2,1)
plt.imshow(np.flip(image, 0))
plt.subplot(1,2,2)
plt.imshow(np.flip(image,1))
plt.show()

1.6.2 Kugeln zählen

Aufgabe 2.3: Kugeln zählen

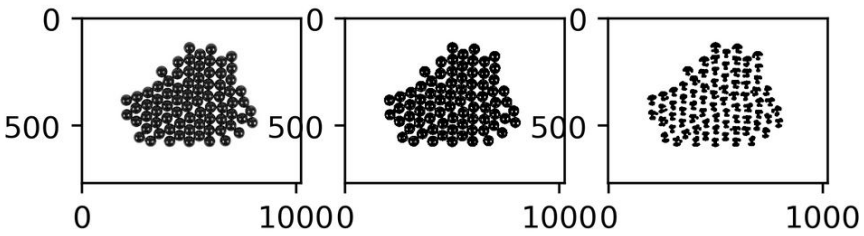
Finden Sie eine Sequenz von OpenCV Operationen die die Kugeln in folgenden Bild zählt? Als Ausgabe sollten sie dann also schlussendlich eine Zahl (zu Speichern in `nr_kugeln`) erhalten.

Hinweis: Beachten Sie dass die Kugeln schwarz auf hellem Hintergrund dargestellt sind.

```
.subplot(1, 4, 1)
eln = cv2.imread('images/Kugeln.jpg', 0)
.imshow(kugeln, cmap="gray")

.subplot(1, 4, 2)
  kugeln_binary = cv2.threshold(kugeln, 127, 255, cv2.THRESH_BINARY)
.imshow(kugeln_binary, cmap="gray")

.subplot(1, 4, 3)
eln_closed = cv2.morphologyEx(kugeln_binary, cv2.MORPH_CLOSE, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (15, 15)))
.imshow(kugeln_closed, cmap="gray")
.show()
```




```
from skimage import measure

labels = measure.label(kugeln_closed, connectivity=None, background=255)
nr_kugeln = len(np.unique(labels))-1

print("Anzahl Kugeln:", nr_kugeln)

plt.imshow(labels, cmap='gray')
plt.show()
```

```
Anzahl Kugeln: 77
```

Btw. die Lösung 77 ist korrekt