Labo 2: Helderheidtransformaties en Filtering in het spatiale domein

Gianni Allebosch, Martin Dimitrievski

1 Helderheidtransformaties

1.1 Histogram

In een histogram wordt voor elke intensiteitswaarde aangeduid hoeveel pixels er met die intensiteitswaarde voorkomen. Bekijk bijvoorbeeld eens het histogram van het beeld `lena.jpg` met onderstaande scripts.

```
import numpy as np
from matplotlib import pyplot as plt
import cv2
image = cv2.imread("lena.jpg",cv2.IMREAD_GRAYSCALE)
hist = cv2.calcHist([image],[0],None,[256],[0,256])
plt.plot(hist)
plt.show()
```

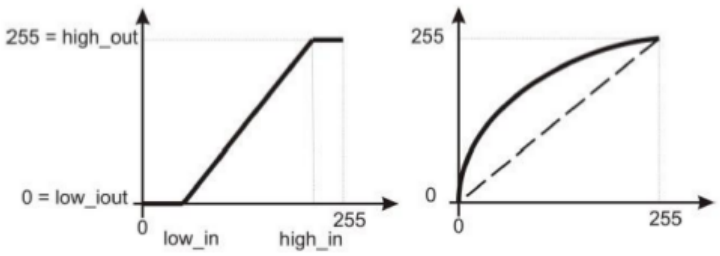
of

```
import numpy as np
from matplotlib import pyplot as plt
import cv2
image = cv2.imread("lena.jpg",cv2.IMREAD_GRAYSCALE)
plt.hist(image.ravel(),256,[0,256])
plt.show()
```

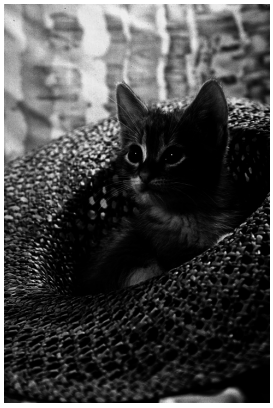
1.2 Intensiteit, contrast en gamma

Door bewerkingen op de intensiteitswaarden van de pixels, kan je de intensiteit en het contrast van het beeld verhogen of verlagen. Deze bewerking kan zowel een lineaire als een niet-lineaire transformatie van de intensiteitswaarden zijn. Door een bepaald bereik van intensiteitswaarden op een groter bereik af te beelden, kan je het contrast in dat bereik vergroten. Door niet-lineaire functies (zoals bij gammacorrectie) toe te passen, kan je het contrast in donkere of lichtere zones in je beeld vergroten. Enkele voorbeelden worden gegeven in de Fig. 1.

De **gammatransformatie** ($\mathbf{I}' = C\mathbf{I}^{\gamma}$) doet een mapping van een bereik van inputpixelwaarden $\mathbf{I}$ naar een bereik van outputpixelwaarden $\mathbf{I}'$. Het effect hangt sterk af van de waarde van γ. **Vraag:** ***voor welke waarden van γ vergroot je het contrast in donkere zones, respectievelijk lichte zones? Bespreek dit in je README.txt.***



Figuur 1. Links wordt intensiteitstransformatie getoond die een 'smaller' intensiteitsbereik (`low_in` tot `high_in`) mapt op het volledige bereik. Rechts staat een transformatiecurve voor gammacorrectie.



Figuur 2. `cat.jpg`

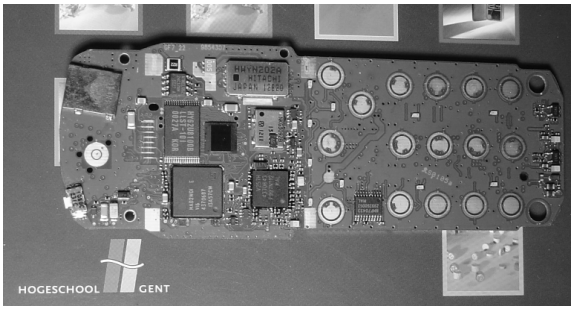
1.3 Histogramegalisatie en histogram matching

De informatie uit het histogram kan gebruikt worden voor een contrastverbetering van een gegeven beeld. ***Histogram matching*** is een techniek waarbij het mogelijk is om een transformatie te berekenen zodat de spreiding een vooropgestelde verdeling benadert, meestal een histogram dat overeenkomt met een contrastrijk beeld. In zijn eenvoudigste vorm wordt er gestreefd naar een uniforme verdeling van de grijswaarden, wat **histogramegalisatie** heet.

1.4 Opdrachten

Programmeer de nodige Pythoncode om onderstaande opdrachten uit te kunnen voeren.

1. Probeer het beeld `cat.jpg` (zie Fig. 2) te verbeteren door het contrast te verhogen in de donkere zones met behulp van een gammatransformatie. Produceer de histogrammen van de afbeelding voor en na de transformatie (zie sectie 1.5). Noem de functie `gamma_transformation`.



Figuur 3. `gsm.jpg`

2. Test histogramegalisatie (zie OpenCV API) uit op het beeld `lowcontrast .jpg`. Bekijk zeker ook het histogram van het beeld voor en na deze operatie. Noem de volledige functie `histogram_equalization`.
3. Laad het beeld `gsm.jpg` (zie Fig. 3) van de binnenkant van een gsm. Dit is een goed, contrastrijk beeld met een inhoud, vergelijkbaar met die van het eerder gegeven beeld (ook een printplaat). Transformeer `lowcontrast.jpg` zodat zijn histogram dat van het gsm-beeld benadert. Concreet willen we dus dat er in het outputbeeld (procentueel gezien) evenveel pixels met dezelfde intensiteiten voorkomen als in het gegeven hogeresolutiebeeld. Maak hiervoor een implementatie van histogram matching als volgt:
   - Produceer van beide beelden eerst de histogrammen en bepaal hieruit de cumulatieve distributies.
   - Construeer een *lookup table* (LUT) met 256 ingangen, zodat alle intensiteitswaarden van het lageresolutiebeeld zoals gewenst gemapt worden op de intensiteitswaarden in het hogeresolutiebeeld, gebruik makend van de informatie uit hun cumulatieve distributies. (Zie ook `Histogram.pdf` voor meer informatie.)
   - Gebruik de LUT om de conversie voor het volledige beeld uit te voeren (je kunt hiervoor de OpenCV-functie `cv2.LUT` gebruiken).

   Noem de functie `histogram_matching`.

1.5 Hulpcode

Gebruik onderstaande code om de cumulatieve distributie van de histogrammen te berekenen en om de afbeeldingen en hun histogrammen voor en na de transformatie te weergeven.

```
def cumulative_histogram(image):
    hist = cv2.calcHist([image],[0],None,[256],[0,256])
    cumhist = np.cumsum(hist).astype(np.float64)
    cumhist = cumhist/cumhist[-1]
    return cumhist
```

```
def show_results(image, result_image):
    fig = plt.figure(figsize=(20,15))
    ax = fig.add_subplot(321)
    ax.set_title("Original Image")
    ax0 = fig.add_subplot(322)
    ax0.set_title("Image after Transformation")
    ax1 = fig.add_subplot(323)
    ax1.set_title("Histogram of Original")
    ax2 = fig.add_subplot(324, sharex=ax1)
    ax2.set_title("Histogram after Transformation")
    ax3 = fig.add_subplot(325, sharex=ax1)
    ax3.set_title("Cumulative Histogram of Original")
    ax4 = fig.add_subplot(326, sharex=ax1)
    ax4.set_title("Cumulative Histogram after Transformation")
    ax1.set_xlim([0,255])
    ax.imshow(image, cmap=plt.get_cmap("gray"))
    ax0.imshow(result_image, cmap=plt.get_cmap("gray"))
    ax1.plot(cv2.calcHist([image],[0],None,[256],[0,256]))
    ax2.plot(cv2.calcHist([result_image],[0],None,[256],[0,256]))
    ax3.plot(cumulative_histogram(image))
    ax4.plot(cumulative_histogram(result_image))
    plt.show()
```

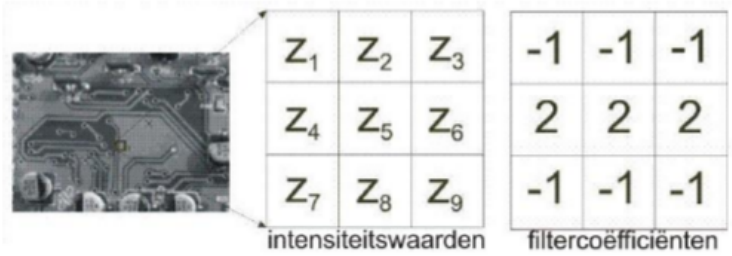
2 Spatiale filters

In beeldverwerking gebeurt spatiale filtering door de interactie van een filtermasker met het beeld. Het masker glijdt over het beeld en in elke pixel die overeenkomt met het centrum van het masker wordt het resultaat van de bewerking bijgehouden (= de respons van het filter).

2.1 Lineaire ruisfilters

Bij lineaire filters wordt de som van de producten van de filtercoëfficiënten met de pixelwaarden berekend. In Fig. 4 vind je een voorbeeld met een 3x3 filtermasker.

Het **averaging** filter is een bekend voorbeeld van een lineair filter, dat meestal gebruikt wordt om ruis in een beeld te onderdrukken (*smoothing*). Experimenteer op de beelden `lena_gaussn.jpg` en `lena_spn.jpg` met de functie `cv2.filter2D` of `scipy.ndimage.filters.correlate`. Gebruik als filtermasker een 5x5 array van elementen met gelijke waarde waarvan de som gelijk is aan 1. Dit masker zorgt voor een uitmiddeling van de ruis in het filtervenster. Het is duidelijk dat de ruis vermindert, maar je krijgt ook andere effecten. **Vraag:** ***geef aan wat de nadelen van deze methode zijn en wat de invloed van de grootte van het filter is. Bespreek ook waarom de som van alle elementen gelijk moet zijn aan 1.***



Figuur 4. Illustratie van de berekening van de respons van een 3x3 filter voor één beeldpixel. Voor deze ene pixel (in het voorgestelde geval is dat de centrale pixel in het venster, dus die met intensiteit z_5) is de informatie van alle pixels in een 3x3 regio rond die pixel in het beeld nodig, dus $z_1 \dots z_9$. De filterrespons kan dan berekend worden als $\sum_{i=1}^{9} w_i z_i$ met w_i de gewichten van de filter en z_i de intensiteitswaarden (waarbij z_5 overeenkomt met de centrale pixel). Het resultaat van deze operatie wordt toegewezen aan de centrale pixel. Deze operatie wordt herhaald voor elke pixel in het beeld.

Een ander populair smoothing filter is een **Gaussiaans** laagdoorlaatfilter. De kernel van deze filter verschilt van de averaging filter: het stelt een Gaussiaanse verdeling voor in 2 dimensies, dus er wordt meer gewicht toegekend aan het centrale pixel van de kernel. De grootte van het filter kan aangepast worden, net als de standaard deviatie van de Gaussiaan. Experimenteer met verschillende instellingen voor deze parameters in `cv2.GaussianBlur` of `scipy.ndimage.filters.gaussian_filter` en bekijk de effecten. **Vraag:** ***vermeld wat de voordelen en nadelen zijn ten opzichte van een averaging filter. Let opnieuw op de grootte van het filter.***

Lineaire filters zoals de **Laplaciaan** laten toe om beelden scherper te maken. Filter het beeld 'gsm.jpg' met de Laplaciaan, die de tweede afgeleide van het beeld benadert in het masker, met de functie `cv2.Laplacian` of `scipy.ndimage.filters.laplace`. Bekijk zowel het masker dat gegenereerd wordt, als het resulterende gefilterde beeld. Dit beeld toont eigenlijk alleen de randen. Buiten die randen is ook nog veel 'ruis' te zien. Om het beeld scherper te maken, moet je dit resultaat dus op de juiste manier combineren met de originele afbeelding.

Een alternatief voor de Laplaciaan is het **Sobelmasker**. Dit filter benadert de eerste afgeleide in een beeld, maar dan standaard in één richting (verticaal of horizontaal). Je kunt hiervoor de functie `cv2.Sobel` gebruiken.

2.2 Niet-lineaire filters

Niet-lineaire filters werken fundamenteel anders dan lineaire filters. Een voorbeeld hiervan zijn ***order statistic filters***. Binnen de kernel van een order statistic filter worden de pixelwaarden gerangschikt volgens grootte. We kiezen vervolgens welk percentiel van de pixels genomen wordt om de centrale pixel te vervangen.

(a) (b) (c)

Figuur 5. (a) Origineel beeld `lena.jpg`; (b) met gaussiaanse ruis en (c) *salt and pepper noise.*

Stel dat we een 3x3 kernel kiezen met allemaal enen, dan komt order = 9 overeen met een **maximumfilter** (centrale pixel wordt het maximum uit de kernel) (`scipy.ndimage.filters.maximum_filter`). Order = 1 komt op zijn beurt overeen met een **minimumfilter** (`scipy.ndimage.filters.minimum_filter`). Als de order gelijk aan de middelste pixel van het filter gekozen wordt (5 in ons geval), dan bekomen we een **mediaanfilter** (`scipy.ndimage.filters.median_filter` of `cv2.medianBlur`).

Fig. 5 (c) bevat '*salt and pepper noise* (zout en peperruis)'. Deze ruis is anders dan de Gaussiaanse ruis in Fig. 5 (b). Bij Gaussiaanse ruis wordt de grijswaarde van elke pixel in meer of mindere mate beïnvloed. Salt and pepper noise heeft slechts invloed op een gedeelte van de pixels: deze random over het beeld verdeelde pixels zijn zwart of wit. De ruisdensiteit is bij het voorbeeld gelijk aan 0.2, wat wil zeggen dat ongeveer 20 % van de pixels veranderd en dus ruis zijn. Het mediaanfilter kijkt naar de pixels uit de kernel en vervangt de centrale pixel door de mediaan. Het mediaanfilter kan dergelijk beeld spectaculair verbeteren, aangezien de extreme zwart/wit pixels weinig invloed hebben op een mediaan.

2.3 Opdrachten

4. Bewerk de file `lena_spn.jpg` met de tot nu toe geziene functies zodat je een beeld bekomt dat zoveel mogelijk op een potloodtekening lijkt. Je moet dus ook de ruis zo goed mogelijk wegwerken. Doe dit in een eerste stap, en toon het tussenresultaat. Noem de functie `pencil`. Een voorbeeld vind je in Fig. 6.

3 Indienen

Indienen van het labo gebeurt via Opdrachten op Ufora. De code dien je in als 1 pythonscript genaamd **labo2.py** waarin alle opdrachten sequentieel worden uitgevoerd. Voeg een **README.txt** bestand toe waarin je programma beschreven

Figuur 6. Voorbeeld van een potloodtekening.

is en waarin de vragen beantwoord worden. Alle gevraagde outputafbeeldingen geef je mee in een .zip-file genaamd **output.zip**.