INF2310 - Oblig1

Andreas Tisland (andretis)

Mars 2019

Oppgave 1

I denne oppgaven skal bildet *portrett.png* klargjøres for en ansiktgjennkjennings-algoritme. Kontrasten skal standardiseres og bildet skal mappes til å passe over en maske.
Implementasjonen er gjort i *oppgave1.py*

Figur 1: Bildet som skal preprosesseres

Standardisering av kontrast

Bildet skal gjennomgå en lineær gråtonetransform som er spesifisert til å endre middelverdien til 127 og standardavviket 64.

Implementasjonen bruker formlene:

$$a = \frac{\sigma_T}{\sigma}$$
$$b = \mu_T - a\mu$$

Der μ og σ er inn-bildets middelverdi og standardavvik, og μ_T og σ_T er ny spesifisert middelverdi og standardavvik.

Deretter utføres transformen på inn-bildet:

$$g(x, y) = af(x, y) + b$$

Denne transformen kan gi intensitetsverdier utenfor 8-bits intervallet $[0 - 255]$, derfor gjøres en *lineær normalisering* av det transformerte bildet slik at det bare har verdier på intervallet $[0 - 255]$.

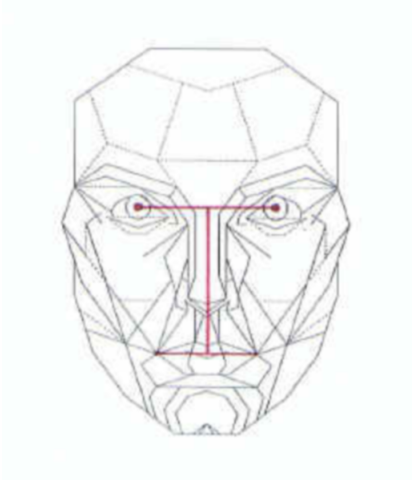
Mellom-resultat etter gråtonetransformasjon:

Figur 2: Resultat av gråtonetransformasjon

Gråtonetransformasjonen er implementert metoden ***kontrast_standardisering()***

Standardisering av geometrien

Skal matche øyne og munn til input-bildet og følgende maske:



Figur 3: Masken som skal matche øyne og munn

Dette ble gjort ved å plukke ut 3 pikselpunkter (øyner og midten av munn) i begge bilder og deretter løse likningssystemet:

$$AX = Y$$
$$A = YX^{-1}$$

Der X består av de 3 pikselpunktene fra inn-bildet og Y består av de 3 punktene fra masken.

$$X = \begin{bmatrix} 88 & 68 & 109 \\ 84 & 120 & 129 \\ 1 & 1 & 1 \end{bmatrix} \quad Y = \begin{bmatrix} 260 & 260 & 443 \\ 170 & 341 & 257 \\ 1 & 1 & 1 \end{bmatrix}$$

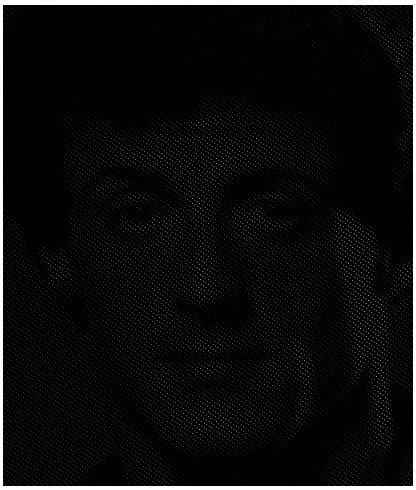
Dette gir transformasjonskoeffisientene $(a_0, a_1, a_2, b_0, b_1, b_2)$ i matrisen A som transformerer piksler fra inn-bildet til å matche geometrien til masken.

$$A = \begin{bmatrix} a_0 & a_1 & a_2 \\ b_0 & b_1 & b_2 \\ 0 & 0 & 1 \end{bmatrix}$$

Dette er implementert i metoden ***finn_koeffisienter()***

Forlengstransformasjon

Resultat av forlengstransformasjon for standardisering av kontrast og geometri (implementert i metoden ***affin_transform()***):



Figur 4: Bildet etter forlengstransformasjon

Baklengstransformasjon

Resultat av nærmeste-nabo-interpolasjon og bilineær interpolasjon:

Figur 5: Nærmeste-nabo-interpolert bilde

Figur 6: Bilineært interpolert bilde

Nærmeste-nabo-interpolasjon er implementert i metoden ***baklengs_transform_narmeste_nabo()***

Bilineær interpolasjon er implementert i metoden ***baklengs_transform_bilinear()***

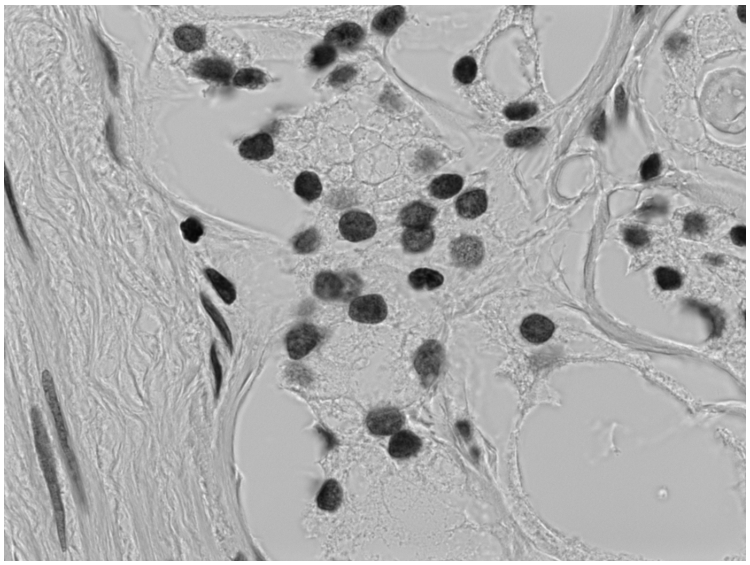
Kommentar til transformasjonene

I forlengsmappingen får vi et bilde som er veldig mørkt og har mange sorte mellomrom, men man kan tyde at det avbilder det originale portrettet. Det er veldig mørkt fordi ut-bildet sin ramme er en del større en inn-bildet, så vi transformerer færre piksler enn totalt antall piksler i ut-bildet.

I baklengsmappingen får vi et mye bedre resultat fordi vi mapper alle pikslene i ut-bildet tilbake i inn-bildet og henter en intensitetsverdi som passer. Dermed får vi fylt de mørke mellomromene i ut-bildet med passende intensiteter.
Resultatet blir litt glattere og finere for den bilineære interpolasjonen, mens nærmeste-nabo-interpolasjon ser litt mer kornete ut.

Oppgave 2

Skal i denne oppgaven lage et program som detekterer kanten til cellekjerner. Oppgaven er implementert i ***oppgave2.py***



Figur 7: Programmet skal detektere kanter i dette bildet

Konvolusjon

Konvolusjonen er implementert ved at det lages en kopi av inn-bildet som paddes utifra størrelsen på filteret. Antall rader som legges til oppe og nede gis av $a = \frac{m-1}{2}$, der m er antall rader i filteret. Antall kolonner som legges til på begge sider gis av $b = \frac{n-1}{2}$, der n er antall kolonner i filteret.

Det paddes med nærmeste pikselverdi fra inn-bildet.
Deretter loopes det over pikslene i inn-bildet slik at vi får origo til filtret på inn-bildets piksler når vi legger det over det paddede bildet. For hver av dem regnes den vektede summen av filteret med det paddede bildet. For at filteret skal snus 180^o legges det til $2a$ og $2b$ til rad og kolonne når vi henter verdier i det paddede bildet.
Implementert i metoden ***konvolusjon()***

Cannys algoritme

Implementeringen av Cannys algoritme starter i hovedmetoden ***canny()***. Her settes først parameterne *sigma*, T_l,T_h og så lastes bildet inn.

Deretter kjøres metoden ***lag_gauss_filter(sigma)*** som lager et Gauss-filter som passer til valgt standardavvik (*sigma*). Filteret får størrelse $n\mathbf{x}n$ der $n = 1 + 8 * sigma$. For verdiene i filteret brukes formelen:

$$h(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Filteret normaliseres deretter slik at summen av vektene blir 1.
Deretter filtreres bildet med Gauss-filteret i metoden ***konvolusjon(bilde,filter)***

Utreging av gradient-magnitude og gradient-retning skjer så i metoden ***gradient_magnitude_vinkel()***. Der brukes den symmetriske 1D-operatoren for h_x og h_y:

$$h_x = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \quad h_y = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix}$$

g_x og g_y fås så ved å konvolvere h_x og h_y med inn-bildet. Nå regnes magnituden og retningen ut:

$$M = \sqrt{{g_x}^2 + {g_y}^2}$$
$$\theta = \tan^{-1}(g_y/g_x)$$

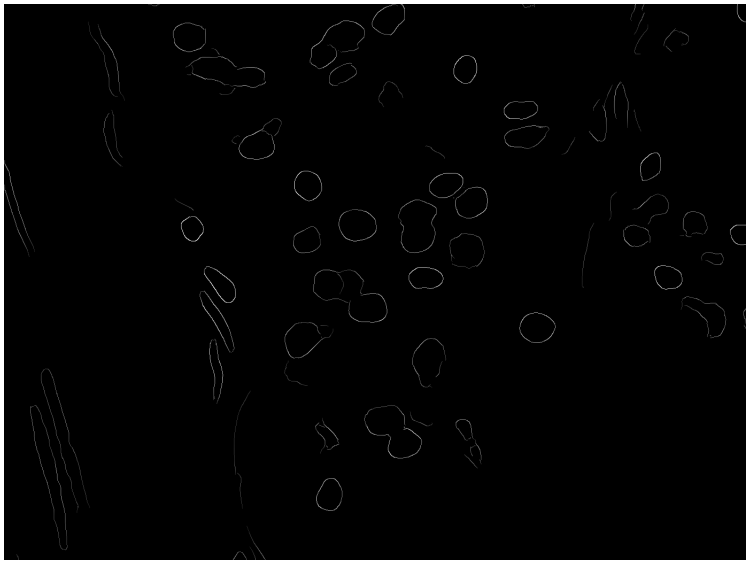
Etter at magnituden og retningen er regnet ut kjøres ***kant_tynning()***. Den sjekker gradientvinkelen til hver piksel og nuller den hvis den er nabo med en sterkere intensitet i gradientretningen. Dette gjør kantene tynnere ved at kun de sterekste intensitetene langs kanten bevares.

Siste del er å kjøre ***hysteresetersking()***.
Parameterne T_l og T_h er terskler for hvilke piksler som blir merket som kant. Alle piklser med intensitet over T_h blir med en gang merket og lagret i g_{NH}. Pikslene mellom T_l og T_h lagres i g_{NL}, mens de under T_l ignoreres.

I implementasjonen lagres indeksene til de merkede pikslene i en liste (*merket*) og en loop går gjennom disse og sjekker om de har 8-naboer i g_{NL} større enn null. 8-naboene blir merket og loopen kjører på nytt med de nye pikslene. Dette gjentar seg til det ikke lenger er nye merkede piksler.
Til slutt ligger alle merkede piksler i g_{NH} som returneres fra metoden.

Resultat av Canny

Det følgende resultatet hadde parameterverdiene $\sigma = 5$, $T_h = 40$, $T_l = 80$



Figur 8: Resultat av Canny-algoritmen

Resultatet er ikke optimalt, men har de fleste av kantene til cellekjernene og det er lite støy i bildet.

Ved å øke standardavviket vil man fjerne mer av lavpass-støy, men økes den for mye blir kantene for glatte og brede og vi mister informasjon.

T_h og T_l må tilpasses slik at T_h er stor nok til at bare tydelige kanter blir merket først og ikke støy. T_l må ikke være for lav slik at hysteresetersklingen kun merker piksler som ligger langs kanter og ikke merker støy.