codegreenrgb0,0.6,0 codegrayrgb0.5,0.5,0.5 codepurplergb0.58,0,0.82 backcolourrgb0.95,0.95,0.92

mystyle backgroundcolor=backcolour, commentstyle=codegreen, keywordstyle=magenta, numberstyle=codegray, stringstyle=codepurple, basicstyle=, breakatwhitespace=false, breaklines=true, captionpos=b, keepspaces=true, numbers=left, numbersep=5pt, showspaces=false, showstringspaces=false, showtabs=false, tabsize=4

style=mystyle

Mini Project 1: Filtering and image restoration

Jeppe Matzen

September 2020

1 Aim:

Målet med mini projekt 1 er at overfører teori til $C++$ kode ved at implementere funktioner og anvende funktionerne på virkelige medicinske billeder.
For selv at kunne konstruere $C++$ funktioner fra bunden er det nødvendigt at forstå formålet og fremgangsmåden detaljeret.

2 Method:

2.1 Opgave 2

Opgaven to bad om at få udregnet og tegnet et gray-level-histogram over gray-values.
Det gråtoner billede der arbejdes med i denne opgave, er et 2^8 bits bilede, bestående af pixel værdier fra 0-255. 0 svarer til sort og 255 til hvid.
For at kunne udregne histogrammet tælles frekvensen af hver pixel. Frekvensen normaliseres herefter ved at dividere med det totale antal pixel (Rækker * Kolonner). Dette giver et "probability density function" (PDF), som indeholder sandsynligheden fra 0-1 for at observere en bestemt pixel. Det normaliserede histogram er et godt værktøj til at danne et overblik over fordelingen af pixel i et billede.

2.2 Opgave 3

I opgave tre skulle der laves et stretch på histogrammet fra opgave to. Dette gøres for at udnytte hele den dynamiske pixel range. Et stretch udvider range fra at gå fra billedets min til max til at gå fra, i det her tilfælde 0 til 255. Dette ændrer ikke i udformningen af histogrammet, men flader det ud. Til dette benyttes formlen

$$g_i = \frac{2^n - 1}{a_{max} - a_{min}} * a_i$$

Hvor n er antallet af bits, a_{max} er den største pixel værdi og a_{min} er den mindste

pixel værdi. Histogram stretch bruges til at øge kontrasten af et billede, der er forskellen mellem den højeste og den mindste pixel. Ofte vil der sættes en grænseværdi for at undgå at outlier i billedet har indflydelse på hvor meget histogrammet kan strækkes.

2.3 Opgave 4

,

I opgave fire skulle billedet inverteres. I dette tilfælde var det et billede med pixels i rækkevidden 0-255. For at invertere billedet blev der udregnet en ny værdi for hver pixel i billedet med formlen

$$invertedPixelValue = 255 - oldPixelValue$$

Ved at invertere billedet bliver de lyse pixels mørke og omvendt. Dette kan gøre det muligt at fremhæve ting i billedet.

2.4 Opgave 5

I opgave fem skulle der køres et tre gange tre filter på billedet. For at benytte et filter er det nødvendt at "padde" eller tilføje en ramme rundt om billedet. Padding kan enten være lavet med nuller eller gennemsnitsværdien af de nærmeste naboer. Det gøres for at billedet ikke bliver mindre hver gang filteret bruges på billedet.
Et tre gange tre gennemsnits filter, som er brugt i denne opgave, fungere ved at det stiller sig på en pixel, og regner summen af de otte pixels, der ligger omkring og den selv, hvorefter der divideres med størrelsen af filteret. På denne måde bliver hver pixel udskiftet med gennemsnittet af den selv og de omkring liggende. Dette bruges ofte til at udglatte et billede for at fjerne støj. Det vil få billedet til at se mere "grumset" ud.

2.5 Opgave 6

I opgave seks skulle størrelsen på filteret ændre til et fem gange fem filter og efterfølgende skulle det ændres fra et gennemsnits filter til et gausian filter.
Ved at øge filter størrelsen mindsker man sensitivitet over før støj men på bekostning af at man udglatter skarpe overgange som f.eks. kanter i billedet.
Gausian filteret er forskellige fra gennemsnits filteret ved at hver pixel er vægtet efter en gausian fordeling i stedet for at alle har vægten 1. Den højst vægtet pixel er den midterste og dets længere man kommer fra centrum dets lavere vægtet er pixelen.
Gausian filteret og gennemsnits filteret bruges begge at fjerne støj.

3 Results: describe the outcome of applying the tools on the images and reflect on how these tools could be used to improve the usage of the images

Billedet der bliver arbejdet med i opgave to til opgave seks er dette billede af en lunge **??** **??**.

3.1 Opgave 2

Histogrammet viser fordelingen af pixels i billedet. Det ses at største delen af pixels ligger i den venstre del af histogrammet hvilket betyder at det hovedsageligt består af lyse gråtoner. Ud fra histogrammet kan det vurderes om det ville give mening af strække billedet og hvilken thredshold værdi der ville være god for billedet. Forholdet i histogrammet er skaleret til vindue størrelsen.

[b]0.49 [width=]Image/lung.png

Figure 1: Billede af lunge.

[b]0.49 [width=]Image/Histogram.png

Figure 2: Histogram af lunge billede

Figure 3:

3.2 Opgave 3

Efter histogrammet er blevet strækket, er Det første Spike rykket helt til venstre og histogrammet er blevet fladere. Dette resulterer i at hele den dynamiske range er blevet udnyttet. Det giver et mere tydeligt billede fordi der er stører forskel i kontrasten.

3.3 Opgave 4

På figur **??** ses det inverterede billede af en lunge. Dette fremhæver nogen farve forskelle som ellers ikke var så tydelige i det originale billede.

3.4 Opgave 5

På figur **??** ses på billede (a) det originale billede af en lunge og på billede (b) lugen med et 3X3 filter. På det filtrerede billede ses en større forskel i farve. Dette skaber et mere tydeligt billede og gør det nemmere at identificere strukturer i billedet.

Ulempen er at kanter bliver mere udtværet, men med et 3X3 filter er dette minimalt.

3.5 Opgave 6

På figur **??** ses på billede (b) et lunge billedet med et 5X5 gausian filter og til højre lungebilledet med et 3X3 gennemsnits filter. Det er svært at se en forskel på de to filter. Gaussian filteret er dog bedre, hvis der skal arbejdes med frekvenser i billedet. 5X5 filteret udtværer kanter i billedet yderligere end et 3X3 filter.

3.6 Opgave 7

Første skridt i behandlingen af billedet af hjertet var at kigge på billedet af hjertet og histogrammet for at danne et overblik over hvilke værktøjer der ville være godt at bruge og i hvilken rækkefølge.
Ud fra histogrammet kan det ses at man ikke ville få noget ud af at strække histogrammet. Dette skyldes at der er et meget stort Spike ved 0 og en høj frekvens af 255.
Billedet af hjertet viser meget støj. Dette forsøges fjernet ved at bruge et 5X5 Gaussian filter. Dette vil påvirke skrapheden af kanterne men ikke signifikant.

Næste skridt er at invertere billedet for at fremhæve hjertevægen, som vi gerne vil fremhæve, med sort i stedet for hvid. Dette skidt ændrer ikke billedet.
Næste skridt er at thredshold billedet i to dele. Billedebehandlingsprogrammet GIMP er blevet brugt til at identificere (Row, Col) værdier i forskellige områder i billedet. Disse er blevet brugt til at thredsholde billedet med forskellige thredshold værdier. Øverste venstre halvdel af billedet er thredsholded med en værdi på 176 og resten af billedet en thredshold værdi på 135. Efterfølgende er det binære billede blevet Eroder og dialated for at optimere på tidligheden af hjertevægen.
Det færdige resultat ses i **?? ??**, hvor hjertevægen og hjerteklapperne er markerede i hvid.

[width=0.5]Image/StrechedHistogram.png

Figure 4: Stretched histogram af lunge billede

[width=0.48]Image/InvertedImage.png

Figure 5: Inverterede af lunge billede

[b]0.49 [width=]Image/lung.png

Figure 6: Billede af lunge.
[b]0.49 [width=]Image/Image3X3.png

Figure 7: Filter med 3X3 lunge billede

Figure 8:

[b]0.49 [width=]Image/Image3X3.png

Figure 9: Filter med 3X3
[b]0.49 [width=]Image/Image5X5.png

Figure 10: Filter med 5X5 Gausian

Figure 11:

[b]0.49 [width=]Image/heart.png

Figure 12: Original billede af hjerte
[b]0.49 [width=]Image/HistHeart.png

Figure 13: Histogram af hjerte

Figure 14:

[b]0.49 [width=]Image/HeartInvert.png

Figure 15: Inverted hjerte billede
[b]0.49 [width=]Image/HeartFinal.png

Figure 16: Hjerte billede færdig behandlet.

Figure 17:

4 Decribtion of inverseDFT

Funktionen tager som input en string, som er navnet på en fil. Først oprettes objekter: et $Mat < float >$ objekt "IMG" som billedet indlæses til som One channel grayscale. Et Rect objekt der er et rektangel der er samme størrelse som billedet. Et tomt Mat objekt "padded" til at kopiere input billedet over i og tilføjer en ramme af nuller rundt om billedet.
[language=c++]

void inverseDFT(const std::string filename)

// A gray image cv::Mat$_<float > img = cv :: imread(filename, cv :: IMREAD_GRAYSCALE); cv :: RectorigSize(0, 0, img.cols, img.rows)$;

cv::Mat padded; // expand input image to optimal size int m = cv::getOptimalDFTSize(2 * img.rows); int n = cv::getOptimalDFTSize(2 * img.cols); copyMakeBorder(img, padded, 0, m - img.rows, 0, n - img.cols, cv::BORDER$_CONSTANT, cv$ :: $Scalar :: all(0))$;

cv::Mat$_<float > imgs[] = padded.clone(), cv :: Mat_<float > (padded.rows, padded.cols, 0.0f)$;

Der oprettes herefter et Mat objekt med to channels "$imgdft$" hvor "img" meres over i. Dette gøres ved et 1-channel billededе for at gøre det muligt at arbejde med komplekse tal.
dtf funktionen laver en Discrete Fourier transform og overskriver imgdft med de nye værdier.
Efter DFT spilttes over i imgs 2-channels. Så den relle del af DFT ligger i imgs[0] og den imaginære del i imgs[1].
Der oprettes to elementer, magnitude og phase. Efterfølgende ændres imgs[0] og imgs[1] fra cartesian koordinater til polare som gemmes i hhv. magnitude og phase.
dtfshift funktionen flytter rundt på kvadranterne så top-venstre og bund-højre bytter plads og top-højre og bund-venstre bytter plads.
Herefter tages logaritmen af magitude + 1 og imgout oprettes.

[language=c++] // Merge cv::Mat$_<cv :: Vec2f > img_dft; cv :: merge(imgs, 2, img_dft)$;

// Compute DFT cv::dft(img$_dft, img_dft)$;

// Split cv::split(img$_dft, imgs)$;

// Compute magnitude/phase cv::Mat$_<float > magnitude, phase; cv :: cartToPolar(imgs[0], imgs[1], mag$

// Shift quadrants for viewability dftshift(magnitude);

cv::Mat$_<float > orig_magnitudel = magnitude+1.0f; cv :: log(orig_magnitudel, orig_magnitudel)$;

// Output image cv::Mat$_<float > imgout$;

ILP finder halvdelen af den mindste længde af billedet og definere en cutoff frekvens. fs sættes til halvdelen af den mindste side (rows, cols) og ganges med cutoff frekvensen.
Herefter oprettes et mat objekt "filter" bestående af nuller med et rektangel med origo i midten af filter. Rektanglet har højden og bredden fs*2.

[language=c++]

// // ILP float halfOfSmallestImageSide = std::min(img$_dft.rows, img_dft.cols)/2.0; floatD0 = 0.3; intfS = halfOfSmallestImageSide*D0; cv :: Matfilter(img_dft.rows, img_dft.cols, img_dft.type(), cv ::$

$Scalar(0.0, 0.0))$; $filter(cv :: Rect(filter.cols/2 - fS, filter.rows/2 - fS, fS * 2.0, fS * 2.0)) = cv :: Scalar(1.0, 0.0)$;

I næste frekvens splittes filter som tidligere til imgs to kanaler. Der laves en omregning fra cartesian koordinater til polare og de gemmes i magnitude og phase. dtfshift funktionen flytter rundt på kvadranterne og mulSpectrums tager imgdft, plusser den med filter og gemmer den i imgdft.
Herefter splittes imgdft igen ind i imgs. Der sker igen en omregning fra cartesian koordinater til polare og de gemmes i magnitude og phase.
Der oprettes et Mat¡float¿ objekt final magnitudel som får værdien magnitude + 1. Herefter bytter dftshift om på kvadranterne og final magnitudel laves om til log skala.

[language=c++] cv::split(filter, imgs); cv::cartToPolar(imgs[0], imgs[1], magnitude, phase); cv::imshow("Filter", magnitude);

dftshift(filter);

cv::mulSpectrums(img$_dft$, $filter, img_dft, 0); cv :: split(img_dft, imgs); cv :: cartToPolar(imgs[0], imgs[1], magnitude, phase)$;

cv::Mat$_{<} float > final_magnitudel; final_magnitudel = magnitude + 1.0f$;

dftshift(final$_m agnitudel); cv :: log(final_magnitudel, final_magnitudel)$;

Den sidste del af koden udregner den inverse DFT med dft funktion. dft funktion kan regne begge vej. imgdft er input og imgout er output. Funktionen omregner fra frekvensdomænet tilbage til pixels og skalere derefter resultatet.
Herefter normaliseres img, orig magnitudel, final magnitudel og phase mellem 0 og 1 og billederne vises.
imgout ændres til samme størrelse som orig, normaliseres mellem 0 - 1 og vises.
orig magnitudel normaliseres mellem 0 - 255 og gemmes som png.

[language=c++]

// Do inverse DFT cv::dft(img$_dft, imgout, cv :: DFT_INVERSE + cv :: DFT_SCALE + cv :: DFT_REAL_OUTPUT)$;

// Show cv::normalize(img, img, 0.0, 1.0, cv::NORM$_M INMAX); cv :: normalize(orig_magnitudel, orig_magnitudel, 0.0, 1.0, cv :: NORM_M INMAX); cv :: normalize(final_magnitudel, final_magnitudel, 0.0, 1.0, cv :: NORM_M INMAX); cv :: normalize(phase, phase, 0.0, 1.0, cv :: NORM_M INMAX); cv :: imshow("Input", img); cv :: imshow("Originalmagnitude", orig_magnitudel); cv :: imshow("Newmagnitude", final_magnitudel)$;

imgout = imgout(origSize); cv::normalize(imgout, imgout, 0.0, 1.0, cv::NORM$_M INMAX); cv :: imshow("Output", imgout); cv :: waitKey(); cv :: normalize(orig_magnitudel, orig_magnitudel, 0.0, 255.0, cv :: NORM_M INMAX); cv :: imwrite("orig_magnitude.png", orig_magnitudel)$;

Opsummeret tager funktionen et billede:

- Forbereder det til en Discrete Fourier transform
- Laver transformationen
- Laver et filter som transformeres og ligges til billedets magnitude og phase.
- Magnitude plusses med 1 og omregnes til log.

- Billedet invers transformeres.
- Billede magnituderne og phasen normaliseres mellem 0 -1 og vises
- Output normaliseres mellem 0 -1 og vises
- Orig gemmes efter den er blevet normaliserede mellem 0 - 255 .