

Kursrapport

ET2584

App-utveckling med bildtillämpningar

Nenad Cuturic

cnesko@gmail.com

necu20

v3.1.0

September 13, 2021

Läsperson 5

Våren 2021

Institutionen för matematik och naturvetenskap

(TIMN)

Blekinge Tekniska Högskola (BTH)

Sverige

Innehåll

1 Litteraturstudie	3
1.1 Bakgrund	3
1.2 Bildförbättringstekniker	4
1.3 Utvärdering av Android apps	6
1.4 Egen bildförbättringsalgoritm	7
2 Matlabimplementation och utvärdering	9
2.1 LIME	9
2.2 Diskret 2D Haar Wavelet transform för brusreducering	10
2.2.1 2D Haar DWT	11
2.2.2 Inverse 2D Haar DWT	12
2.2.3 Tröskelvärde	12
2.2.4 Implementation	13
2.3 Utvärdering	14
3 Java implementation för Android	17
3.0.1 Java kod och utvecklingsverktyg	17
3.0.2 Utmaningarna	18

<i>INNEHÅLL</i>	2
-----------------	---

4 Slututvärdering	19
--------------------------	-----------

Kapitel 1

Litteraturstudie

1.1 Bakgrund

Sinnen är hos varelser, såväl djur som människor, de neurologiska och psykologiska system för perception av omvärlden och sig själva.[\[1\]](#)

Mänskans förmåga att, med ögat, uppfatta det synliga ljuset dvs. en elektromagnetisk strålning med våglängderna mellan 380 nm och 780 nm, kallas för syn.

Sedan urminnes tid har människan försökt fånga och föreställa sin omvärld i form av bilder. Dessa bilder har med tiden blivit alltmer sofistikerade men fram tills nyligen har bilderna varit uteslutande analoga. Med datorernas ankomst och moderna digitaliseringstekniker har den digitala bilden nästan helt ersatt den analoga.

Detta har dessutom öppnat upp för olika typer av automatiserade bild- och signalbehandlingstekniker. Begreppet signal är mer korrekt benämning för att den omfattar även elektromagnetisk strålning inom hela dess frekvensområde samt signaler skapade av eller med hjälp av (ultra)ljud medan bild associeras traditionellt med synligt ljus.

Bildbehandling kan delas upp i olika operationer på många olika sätt. Vi kommer använda följande uppdelning: **förbättring**, **återställning** och **bildanalys**. I denna rapport kommer vi koncentrera oss enbart på synligt ljus och bildförbättringstekniker.

1.2 Bildförbättringstekniker

Vi kan grovt dela upp bildförbättringar i två grupper:

- **Kvalitativ**, som syftar till att förbättra bildens kvalitet så att den blir mer tilltalande för en människa. Den tekniken är ganska subjektiv för olika personer har olika uppfattningar av vad som är mer eller mindre tilltalande.
- **Kvantitativ**, som underlättar extrahering av relevant information vilket till exempel underlättar segmentering och separation av objekt. Den här tekniken är mer objektiv för man kan oftast kvantitativt mäta resultatet.

Huvudmålet för en bildförbättring är en förändring av bildens attribut för att bättre anpassa bilden till den givna uppgiften och den specifika betraktaren.

Det finns två grundvillkor för en bildförbättringsprocess:

- Den förväntade informationen måste redan existera i bilden exempelvis om vi skall förbättra den blå färgen måste den redan existera i bilden.
- Den förväntade information måste kunna urskiljas från bruset dvs. signal-brusförhållande (SNR = signal-noise ratio) måste vara tillräckligt högt.

En digital bilds spatiella (rumsliga) egenskaper kan representeras av en tvådimensionell (2D) matris där kolumner och rader motsvarar de diskreta koordinaterna för varje pixel. Varje element i matrisen är en- eller flerdimensionell vektor som representerar varje pixels färgegenskaper.

Det finns flera olika färgmodeller och de 3 viktigaste är:

- **Monokrom-/Gråskalemodell**, där varje pixelvärdet motsvarar intensitet (GRAY) på en gråskala oftast $n = k * 8\text{bitar}$ ($k = 1, 2, 4, \dots$) nummer eller $(2^n + 1)$ -nivåer av grå.
- **RGB**, där varje pixel har tre intensitetsvärden (3-dimensionell vektor), ett för varje färg: röd (R), blå(B) och grön(G). Den här modellen efterliknar ögats sätt att tolka bilder (3 typer av koner för dessa tre färger). Omvandling av RGB till en monokrom bild sker enligt följande formel:

$$GRAY = 0.30R + 0.59G + 0.11B$$
- **HSV**, där varje pixel, förutom spatiella koordinater, har ytterligare tre värden.

- Färgton: (H)ue = 0 - 360°, representeras av intervallet [0, 360]
- Mättnad: (S)aturation = 0 – 100%, representeras av intervallet [0, 1]
- Intensitet: (V)alue = 0 – 100%, representeras av intervallet [0, 1]

Det finns en uppsjö olika tekniker för bildförbättring bl.a. kontrastförbättring, filtrering, erosion, utvidgning, bakgrundskorrektion, tröskelvärdering, kantdetektering, vattendelare osv.[2]

För att uppnå något av dessa mål används olika metoder och dessa kan i stort sett delas upp i följande grupper:

- Metoder i den **spatiala domänen** där vi direkt manipulerar bildens pixlar.
- Metoder i **frekvensdomän**: Med hjälp av transformfunktion omvandlar vi bilden till frekvensdomänen där förbättringsprocessen sker (oftast används någon filter) för att sedan omvandla bilden tillbaka till spatial domän igen med hjälp av invers transformfunktion. Olika källor gör olika klassificeringar där ena grupperar Wavelats transformer och Furier transformer tillsammans medan andra separerar dem i olika grupper.
- **Maskinlärning (ML)** där man tränar en ML modell som, baserat på en mängd bildsampel, i sin tur på egen hand lär sig att utföra bildförbättringsfunktioner.

Om man begränsar området till bilder tagna under svagt/dåligt ljus kan man, enligt en granskning[3] från 2020, dela upp algoritmer i följande grupper:

- **Linjära och icke linjära Transformationer** av gråskalebilder, t.ex. förstoring/förminskning av pixelvärdet, spegling osv.
- **Histogramutjämningsmetoder**, syftar på att förbättra kontrasten genom förändringar av intensitet.
- **Retinex** metoder, där man separerar belysnings- och reflektionskomponenter i en bild
- Metoder inom frekvensdomän (se ovan),
- **Bildfusion** metoder, man använder sig av flera bilder med till exempel olika exponeringar, som kombineras till en förbättrad bild
- Metoder för att avlägsna dimman från bilder, man skulle kunna säga att det är en undergrupp till brusreduceringsmetoder
- Metoder som använder sig av maskinlärning (se ovan).

1.3 Utvärdering av Android apps

Numera finns det en stor mängd mobila appar som hävdar att de har någon form av bildförbättringsfunktionalitet. Urvalet av de analyserade apparna baseras på betyget i Google Play, antal nedladdningar, snabb koll på annonserad funktionalitet samt i Photoshop-fallet baserad på dess dominerande marknadsposition.

Dessutom har jag valt att ta med ett par appar som hävdar att de använder sig av någon form av maskinlärning.

Så den slutliga listan blev:

1. **AI Image Enlarger**[\[4\]](#)
2. **AirBrush**[\[5\]](#)
3. **EnhanceFox**[\[6\]](#)
4. **Lumii**[\[7\]](#)
5. **Photoshop Express**[\[8\]](#)
6. **Remini**[\[9\]](#)

Jag har valt att testa enbart de funktionaliteter som är fria vilket uteslöt Photoshop Express helt för det kräver att man måste lämna kreditkortsuppgifter för att kunna utnyttja 3-dagars testperiod.

Utvärderingen gav tyvärr ett magert resultat. Majoritet av applikationerna har lagt tyngdpunkten på olika typer av bildeffekter exempelvis, inramning, bakgrundsförändring, kombinering av två bilder i en osv. som är nog tänkt att locka yngre målgrupp och är rätt så svaga i området som traditionellt uppfattas som bildförbättring.

Applikationerna riktar sig oftast mot förändringar i selfiebilder utan att användare behöver förstå sig på tekniska detaljer. I de fall det även finns bildförbättringsfunktionalitet finns det oftast valet att antingen bara slå på eller slå av utan att kunna finjustera parametrar.

De mer avancerade funktionerna om de finns är låsta och enbart finns i en betalversion.

Av de två appar som hävdar AI-funktionalitet är den ena appen bara ett skal som tillåter uppladdning av bilden till någon server för att sedan i efter hand kunna ladda ner den färdigbehandlade bilden.

Den andra appen ger enbart valet att förbättrabilden utan att kunna justera parametrar. Ytterligare en funktionalitet den har är att färga svartvita bilder.

1.4 Egen bildförbättringsalgoritm

Sedan intåg av smarta telefoner med kameror har jag upplevt problem med bilder tagna under svagt ljus. Så det är ett område som jag har särskilt intresse av och lagt mitt fokus på i denna del av rapporten.

Ett annat område som jag tycker är intressant är förbättring av bilder tagna under vattnet. Här finns ännu fler parametrar som påverkar bildens kvalitet som ljusbrytning i vattnet, vatten som ger effekten av en förstoringsglas samt en annan typ och mycket större brus orsakad av olika partiklar i vattnet.

Mitt val föll på **LIME** (eng. Low-light Image Enhancement via Illumination Map Estimation)-algoritmen[10]. Enligt en färsk experimentell utvärdering [3] av algoritmer för förbättring av bilder under svaga ljusförhållanden ligger LIME i topp i samtliga test och i ett par av dem ligger best. Dessutom är resultat visuellt tilltalande jämfört med andra algoritmer.

LIME-algoritmen baseras på s.k. Retinex modelleringen. Dess syfte är att uppskatta belysningskartan genom att bevara den framstående strukturen av bilden, medan man avlägsnar de redundanta struktur detaljerna (eng. texture). Lime-metoden baseras på följande modell:

$$L = I \odot T \quad (1.1)$$

där: L = originalbild tagen under svagt ljus, I = förbättrad bild, och T = belysningskarta

1. Uppskatta initial belysningskarta genom att beräkna $\max L^c$ över samtliga färgkanaler (R/G/B)

$$\hat{T}(x) \leftarrow \max_{c \in \{R,G,B\}} L^c(x) \quad (1.2)$$

2. Lös följande optimeringsproblem

$$\min_T \|\hat{T} - T\|_F^2 + \alpha \|W \odot \nabla T\|_1 \quad (1.3)$$

där F = Frobenious norm, 1 står för $l1 - norm$, W = viktmatris, samt ∇ består av en horisontell $\nabla_h T$ och vertikal $\nabla_v T$ komponent dvs. (x,y) koordinater

3. Utifrån 1.3 och 1.2 kan man få en förbättrad bild I men den bilden har fortfarande en bruskomponent
4. Beräkna en förbättrad bild I_f utan brus med hjälp av:

$$I_f \leftarrow I \odot T + I_d \odot (1 - T) \quad (1.4)$$

där I_d = brusreducerad bild med hjälp av BM3D-algoritmen, och
 $1 - T$ är en invers (negativ) av belysningskartan T

Som en andra algoritm har jag använt Diskret 2D Haar wavelet transform för brusreducering. Haar wavelet transform brukar fungera bra för ett smalare brusområde (Gaussiskt brus).

Kapitel 2

Matlabimplementation och utvärdering

All kod för detta projekt hittas på Github under <https://github.com/neskoc/apputveckling> och Matlab-implementationen under mappen matlab.

2.1 LIME

Implementationen av LIME algoritmen baseras på papperet och implementationen gjorda av Vasudera och Dora [11]. Själva LIME algoritmen har visat sig vara väldigt avancerad och kräver god kunskap i hur man löser ett optimeringsproblem med Langrange multiplikatorer, avancerad differentiell kalkyl + matriskalkyl för en approximativ lösning av gradientproblem och fast Fourier transform för att underlätta matriskalkylen.

Mitt bidrag består av kodanpassningen för att fungera med gråskale- och kompositbilder, en del omskrivningar och förenklingar av koden, byte av algoritmen för brusreducering och anpassning av gränssnittet. Även den förenklade koden är rätt så stor och komplex och därfor kommer det bara ges en översiktlig förklaring av koden.

Filen **exec_lime_enhancement.m** är huvudfilen för implementationen. Den i sin tur anropar funktionen **exec_full_haar_main_module** som finns i en separat fil som i sin tur anropar:

- **lime_enhance(orig_img, alpha, mu, rho, gamma, iter)** funktion där alla lime-beräkningar sker/styrs samt,

- `exec_full_haar_main_module(enh_img, level, th_type)` för brusreducering.
- I slutet ritas resultatet.

Kodstrukturen följer de tre delproblem detaljbeskrivna i II-A(1,2,3)[11] (T/G/Z delproblem) och samtliga steg är sammanfattade i II-C algoritm 1 på sidan 4[11]. Det finns en hel del parametrar som kan ställas in för att finjustera algoritmen

1. Parametrar som är gemensamma för samtliga bilder (konfigureras i `exec_full_haar_main_module.m`)
 - $\alpha = 0.08$ - används i s.k. "krympningsoperation" (shrinkage operator) G-delproblem" II-A2[11], $S_e[\alpha] = \text{sign}(\alpha) \max(|\alpha - e|, 0)$ originalpapper föreslår värdet 0.15
 - $\gamma = 0.8$ - parametern för gamma korrigering av belysningskartan $T \leftarrow T^\gamma, \gamma < 1$ för att öka visuell uppfattning av resultatet, tillhör post-processing delen av algoritmen II-C[11]
2. Parametrar som är specifika (optimerade) för varje bild. I början av huvudfilen `exec_lime_enhancement.m` anges en matris med optimerade parametrar för 9 testbilder som brukar användas för utvärdering av de olika bildförbättringsalgoritmerna. Nya bilder kan använda dessa som initiale parametrar för att sedan finjusteras för den specifika bilden.
 - μ, ρ - konstanter (parametrar) för "lösaren" (solver) av optimeringsproblemet. Se papparet[11] för detaljerade förklaringar.
 - $iter$ - antal iterationer för lösaren. I koden används 30 men papparet föreslår 50. Jag har inte kunnat konstatera någon större skillnad (det är en subjektiv bedömning) efter 30 och just den delen är mest resursintensiv. Lägre värden sparar tid/beräkningsresurser och om algoritmen uppfattas som trög är denna parameter som påverkar mest.

2.2 Diskret 2D Haar Wavelet transform för brusreducering

LIME algoritmen för bildförbättring förstärker även brus. Därför behöver man ha ett efterbehandlingssteg där man tillämpar någon algoritm för brusreducering. I det första papparet om LIME[10] från 2016 föreslås *Block-matching and 3D filtering (BM3D)* algoritm för brusreducering.

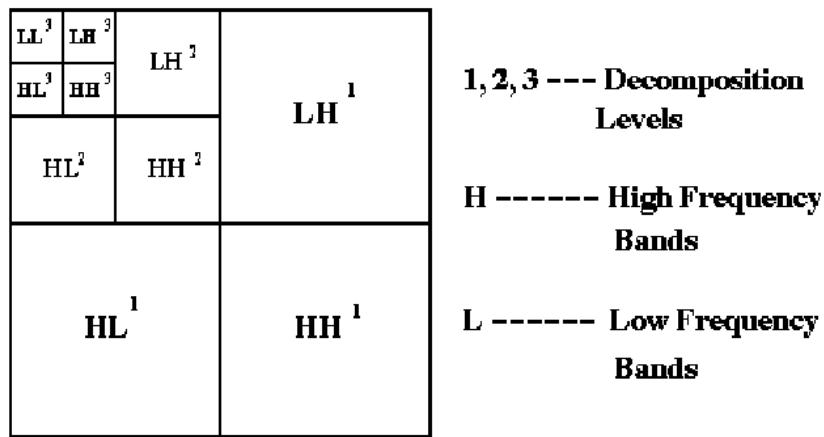
Jag har i stället valt Diskret 2D Haar Wavelet. Anledningen är att jag, för ett tag sedan, som en del av en grundläggande maskinlärningskurs stött på s.k. inplace Haar Wavelet transform (jag skrev ett program för bildkomprimering) och tyckte det vore intressant att undersöka hur den kan användas för brusreducering.

Kursboken[12] förklrar Haar DWT grunderna men utan att ta upp 2D transform och brusreducering. Generellt har jag inte kunnat hitta något lämpligt papper om 2D Haar DWT utan jag har samlat in information från en del olika källor utan att kunna citera någon specifik.

2.2.1 2D Haar DWT

Figur 2.1 illustrerar hur en framåtriktad (forward) Haar DWT fungerar (3 nivåer). Varje transform skapar 4 kvadranter där första kvadranten (vänster upp) är komprimerad bild genom att ta medelvärde av 4 punkter, se ekvation (2.1), som också representerar lågfrekvensband (LL).

Resterande 3 kvadranter är har wavelet koefficienter med antingen bara högfrekvensband (HH) eller både hög- och lågfrekvensband (LH/HL), ekvationer 2.2, 2.3 och 2.4.



Figur 2.1: 2D Haar DWT, 3 nivåer [13]

Följande ekvationer demonstrerar hur nivå 1 av 2D Haar DWT koefficienter beräknas.

$$y_{LL} = \frac{1}{4}(x_{11} + x_{12} + x_{21} + x_{22}) \quad (2.1)$$

$$y_{LH} = \frac{1}{4}(x_{11} + x_{12} - x_{21} - x_{22}) \quad (2.2)$$

$$y_{HL} = \frac{1}{4}(x_{11} - x_{12} + x_{21} - x_{22}) \quad (2.3)$$

$$y_{HH} = \frac{1}{4}(x_{11} - x_{12} - x_{21} + x_{22}) \quad (2.4)$$

Värdena x_{11}, x_{12}, x_{21} och x_{22} representerar intensitetsvärden för 4 närliggande punkter i en 2×2 matris där det första indexet representerar rader och andra kolumner. Till exempel i en enkel bild med 4×4 punkter delar man upp bilden i 4 sådana områden (1), (2), (3) och (4), se tabellen nedan.

$x_{11}^{(1)}$	$x_{12}^{(1)}$	$x_{11}^{(2)}$	$x_{12}^{(2)}$
$x_{21}^{(1)}$	$x_{22}^{(1)}$	$x_{11}^{(2)}$	$x_{12}^{(2)}$
$x_{11}^{(3)}$	$x_{12}^{(3)}$	$x_{11}^{(4)}$	$x_{12}^{(4)}$
$x_{21}^{(3)}$	$x_{22}^{(3)}$	$x_{11}^{(4)}$	$x_{12}^{(4)}$

Sedan ”faltas” (det är inte faltning enligt en strikt definition utan vi använder formler (2.1) till (2.4) i stället) denna 2×2 matrix över hela bilden med steg (eng. ”stride”) 2 dvs. varje punkt tas med en och bara en gång. Värdena/beräknade HWT koefficienterna läggs i en ny matris enligt nedan:

$y_{LL}^{(1)}$	$y_{LL}^{(2)}$	$y_{LH}^{(1)}$	$y_{LH}^{(2)}$
$y_{LL}^{(3)}$	$y_{LL}^{(4)}$	$y_{LH}^{(3)}$	$y_{LH}^{(4)}$
$y_{HL}^{(1)}$	$y_{HL}^{(2)}$	$y_{HH}^{(1)}$	$y_{HH}^{(2)}$
$y_{HL}^{(3)}$	$y_{HL}^{(4)}$	$y_{HH}^{(3)}$	$y_{HH}^{(4)}$

Upprepas man processen på den första kvadranten får man nivå 2, nivå 3 osv. De första 3 nivåerna är demonstrerade i figur 2.1.

2.2.2 Inverse 2D Haar DWT

Om man inte förändrar wavelet koefficienterna inverstransformen återställer ursprungsbilden fullständigt dvs. transformen är fullt reversibel.

$$x_{11} = y_{LL} + y_{LH} + y_{HL} + y_{HH} \quad (2.5)$$

$$x_{12} = y_{LL} + y_{LH} - y_{HL} - y_{HH} \quad (2.6)$$

$$x_{21} = y_{LL} - y_{LH} + y_{HL} - y_{HH} \quad (2.7)$$

$$x_{22} = y_{LL} - y_{LH} - y_{HL} + y_{HH} \quad (2.8)$$

2.2.3 Tröskelvärde

Bruset är vanligtvis koncentrerad i högfrekvensband och ett sätt att minska brus är att tillämpa någon form av lågpassfilter.

I Haar DWT fallet beräknar man fram ett tröskelvärde för koefficienterna från de 3 kvadranterna med högfrekvensband.

Koefficienter som är mindre än tröskelvärdet sätts till noll.
Det finns två strategier för behandling av koefficienterna som är större än tröskelvärdet ($|Y| \geq t$)[\[14\]](#)

1. **hard** - ingen ändring
 2. **soft** - nya koefficienter beräknas genom att subtrahera tröskelvärdet från koefficientens värde.
- $$T_{soft}(Y, t) = sgn(Y)(|Y| - t)$$

På olika ställen föreslås olika metoder för beräkning av tröskelvärdet och jag har valt att implementera algoritmen som föreslogs i diskussionsforumet på Mathworks webbsida[\[15\]](#).

$$T_{threshold} = \frac{\left(\frac{median|M|}{0.6745}\right)^2 \sqrt{\log \frac{length(M)}{level}}}{std(M_{HH})} \quad (2.9)$$

där M står för hela bilden, M_{HH} står för HH-kvadrant för den valda nivån och level är transformens nivå.

Denna formel är mest lik s.k universal threshold formel som nämns i några papper.

$$T_{univ_thresh} = \frac{median|M| \sqrt{2\log(length(M))}}{0.6745} \quad (2.10)$$

Jag har först testat den här formeln men tröskelvärdet var alldeles för högt så det hade ingen effekt.

En implementation använde också en skalningsfaktor vilket skulle kunna läggas till för att bättre styra beräkningen av tröskelvärdet.

2.2.4 Implementation

Huvudfunktionen `exec_full_haar_main_module(orig_img, level, th_type)` hittas i filen `exec_full_haar_main_module.m`.

- **orig_img** - bilden som skall behandlas
- **level** - nivå för Haar transformen
- **th_type** - tröskelstrategi, tillåtna värden är (soft, hard)

Haar transform kräver att bildstorleken (bredd och höjd) är exponenter av 2. Därför anpassas (expanderas) bildens storlek till närmaste exponenter av 2.

Färgbilder omvandlas till hsv format och bara intensitetskomponenten behandlas av transformen. Efter filtrering och invers transform omvandlas hsv till rgb igen och bilden beskärs till ursprungsstorleken.

Huvudfunktionen i sin tur anropar följande funktioner

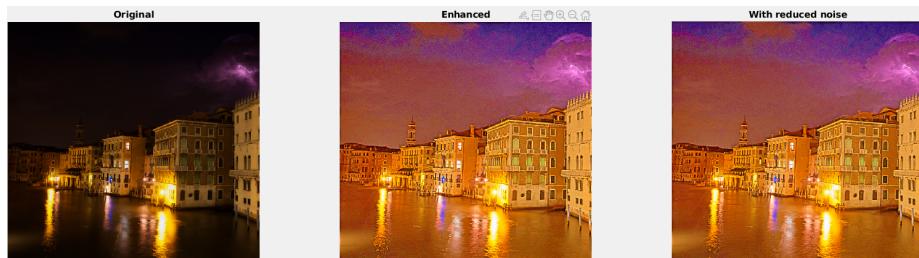
- **full_haar2d(X, level)** - framåt 2D Haar DWT
- **full_ihaar2d(X, level)** - invers 2D Haar DWT
- **apply_haar_filter(M, level, th_type, HH_matrix)** - lågpassfilter

Vill man testa enbart brusreducering kan man köra **test_full_haar.m**. I den filen genereras Gaussiskt brus via Matlab funktionen
`imnoise(Vbf_norm,"gaussian", 0, 0.003)`

2.3 Utvärdering

LIME algoritmen ger väldigt bra resultat för bilder tagna med svagt ljus se figur 2.2. Däremot är algoritmen rätt så komplicerad med många parametrar så en bra resultat kan fås enbart om man är noga med finjustering av parametrarna.

Den bästa strategin för justering av parametrarna är att utgå ifrån de 9 förvalda parametrarna från parametermatrisen. När man hittat de som fungerar best kan fortsätta finjustera dessa.



Figur 2.2: Originalbild, förbättrad samt efter brusreducering

2D Haar DWT ger med en del justering synlig förbättring för Gaussiskt brus vilket illustreras i figur 2.3 (förstora dokumentet tills den vänstra halvan börjar visa brus).



Figur 2.3: Bild med Gaussiskt brus (vänster halva) och efter brusreducering nivå=5 och soft tröskelstrategi

Jag har haft väldigt svårt att få någon märkbar brusreducering av bilderna efter Lime-förbättringen vilket demonstreras med bilden först från höger i figur 2.2.

LIME vs. Histogramutjämning

Figur 2.4 visar resultat av LIME och inbyggd histeq funktion för histogramutjämning utan brusreducering. Utifrån bilderna är det uppenbart att histeq överexponerar bilden och LIME förbättringen känns mycket mer naturlig samt man kan se betydligt fler detaljer.

Visserligen har vi tittat bara på en bild och LIME parametrar är optimerade just för den bilden men man kan med stor säkerhet påstå att LIME algoritmen presterar mycket bättre än histogramutjämning av bilder tagna under dåliga ljusförhållanden.

Om man tittar på Median Absolute Deviation (MAD) parametern
 $MAD = \text{median}(|X - \text{median}(X)|)$, X=den analyserade bilden
 har vi $MAD_{LIME} = 48.4$ och $MAD_{histeq} = 68.0$

Vanligtvis önskar man MAD så hög som möjligt för att det indikerar bättre kontrast (större avvikelse från median) och bättre spridning över hela spektra men i vårt fall är MAD värden missvisande dvs. den objektiva utvärderingen går emot den subjektiva bedömningen.



Figur 2.4: Jämförelse mellan LIME (mitt) och histeq (höger) bildförbättring

2D Haar DWT vs. Imbilatfilter

Det är ganska svårt att jämföra effekten av brusreducering på en bild utan rätt så uppenbara problem med brus så jag har valt att göra jämförelsen av brusreduceringsalgoritmer på ett Matlab-genererat Gaussiskt brus på en bild.

Som jämförelsemетод har jag valt standard Imbilatfilt-funktion (Matlab: Bilateral filtering of images with Gaussiska kernels) för att den används i pappelet[11] samt 2D Haar DWT nivå 5 med soft tröskling.

Resultatet demonstreras i Figur 2.5.



Figur 2.5: Jämförelse mellan Haar (mitt) och imbilatfilt (höger) brusreducering

Det är uppenbart att bilateralt filter presterar mycket bättre än Haar. Haar skulle nog kunna prestera bättre med en mer avancerad tröskelmetod.

Intuitivt borde MAD inte vara särskilt nyttigt mått för att mäta effektivitet av brusreducering. Hur som helst är MAD värdena följande:

$$MAD_{Haar} = 3.00 \text{ och } MAD_{bilatfilt} = 4.05$$

Med andra ord skiljer sig värdena väldigt lite och då är det svårt att dra någon slutsats.

Kapitel 3

Java implementation för Android

Jag har valt att implementera 2D Haar DWT som min Android-bildförbättringsmetod. Den viktigaste anledningen var att Lime-metoden är väldigt komplex och skulle ta betydligt mer tid.

3.0.1 Java kod och utvecklingsverktyg

Implementationen är gjort utifrån ImageEnhancer exempelapplikationen där jag lagt till två klassmoduler:

- **HaarDWTEnhancer.java** - innehåller klassen HaarDWTEnhancer som är huvudklass för implementation av 2D Haar DWT och i stort sett följer namnkonventionen och strukturen vilka användes i Matlab-implementationen. Den klassen realiseras ImageEnhancer interface samt använder sig av Matrix-klassen för matrix-operationer.
- **Matrix.java** - Den här klassen är en utökning av grunläggande Matrix-klassen gjord av Sedgewick and Wayne[16] där jag lagt till stödfunktioner för Haar DWT. I och med att Java inte har ett inbyggt stöd för matriser och dess operationer samt saknar Matlabs rika bibliotek för signalbehandling skiljer sig koden en del från Matlab-implementationen. I början av filen finns en lista över samtliga funktioner (15) jag lagt till.

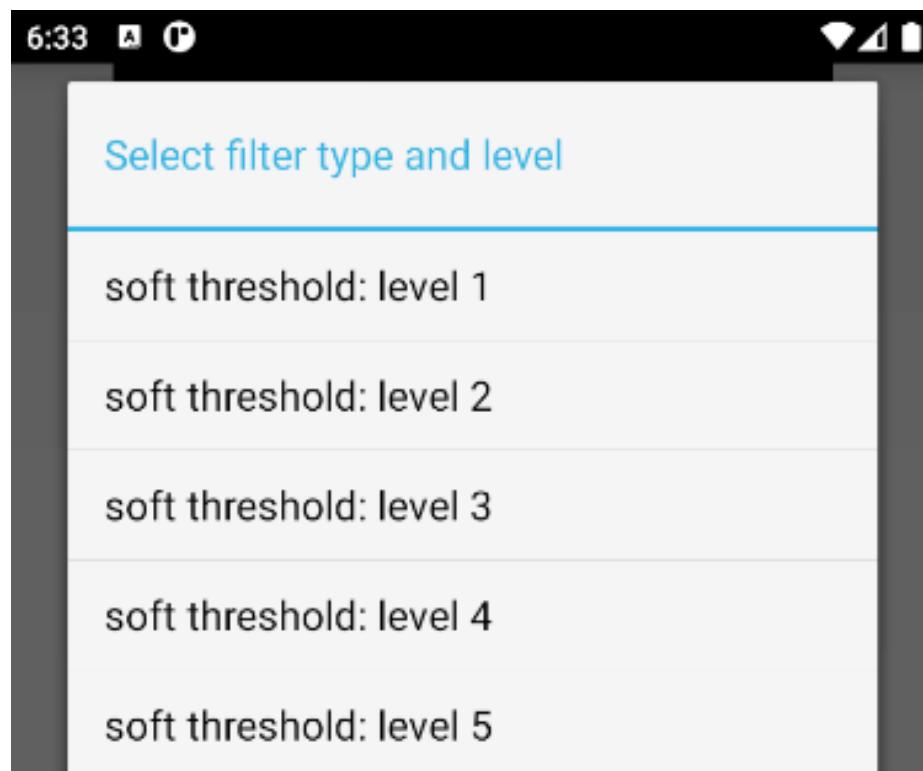
Jag har använt Eclipse som verktyg för att utveckla och testa Matrix-klassen medan själva Android-applikationen utvecklades och avelsades i Android Studio

och emulatorn av Pixel XL med API 30.

3.0.2 Utmaningarna

En av de största utmaningarna, förutom att utveckla matrisoperationer som skulle fungera i Java, var att få till ett gränssnitt för hantering av parametrarna. Detta var min första Android app och jag har valt att återanvända samma gränssnitt (AlertDialog) som användes i exempelapplikationen.

Det blev en lång lista där man samtidigt väljer både typ av tröskel samt transformens nivå: först samtliga möjliga soft alternativ (5 visas i Figur 3.1) och därefter möjliga nivåer med hard tröskelstrategi (visas inte på bilden).



Figur 3.1: Konfiguration av tröskelvärden

De nivåer som erbjuds är beräknade dynamiskt utifrån den expanderade bildens storlek så att applikationen inte skulle krascha om man valt för hög nivå.

Kapitel 4

Slututvärdering

Enligt Mohideen m.fl. [13] uppnås ett optimalt resultat om man väljer soft tröskelstrategi på nivå 3 eller 4 vilket har också visat sig stämma överens med hur applikationen presterar.

Det går att välja väldigt höga nivåer av transformen men ju högre man kommer upp desto större chans är att det börjar uppstå artefakter vilket också nämns i [13] och demonstreras i Figur 4.5.

Anledningen är att varje förändring av koefficienterna (minskning på grund av att beräknad tröskel överstigs) fortplantar sig för varje steg då man inverstransformerar bilden tillbaka till den ursprungliga storleken. Det inträffar på sätt och vis en kumulativ effekt.

I ett fall med nivå 3, vilket demonstreras med Figur 2.1 påverkas koefficienterna i kvadranten längst upp till vänster. När man sedan ”expanderar”koefficienterna sker det i kvadranter vars yta ökar i storlek med faktor 4 så en liten förändring på få koefficienter på en hög nivå fortplantar sig över ett stort område i ett kvadratformat mönster som till slut täcker hela bilden.

Något som inte nämndes i kapitlet [Utmaningarna](#) är att jag lagt en skalningsfaktor: `final float scaling_factor = (float) 1.2` i klassens **HaarDWTEnhancer.java** medlemsfunktionen `sigtthresh` för tröskeln.

Denna skalningsfaktor kan användas för att proportionellt justera tröskelns värde upp eller ner. I och med att Androids AlertDialog inte är så flexibel är denna skalningsfaktor hårdkodad dvs. den kan inte ändras av applikationens användare.

Min implementation av Haar DWT brusredusering använder sig av en matris på 2x2 för att beräkna transformens koefficienter. Mohideen [13] har testat med matriserna 3x3, 5x5 och 7x7 och även med dessa är resultatet sämre än andra

testade brusreduceringsmetoder. Därför var det förväntat att resultaten skulle bli lite sämre än i papperet. De experimentella resultatet bekräftar det.

Figur 4.1 visar ett exempel på brusreducering i en android-hårdvara med min implementering gjord av läraren Benny Lövström.

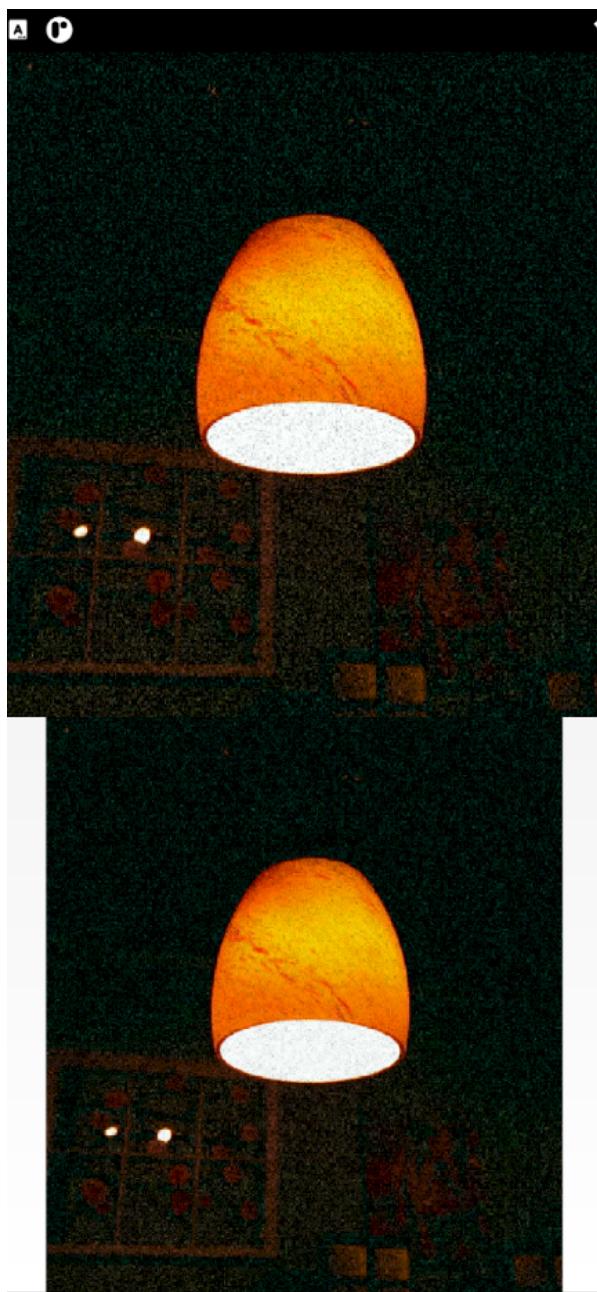


Figur 4.1: Lena med Gaussiskt brus (upp) samt
efter brusreducering med soft tröskel, transform nivå 3 (ner)

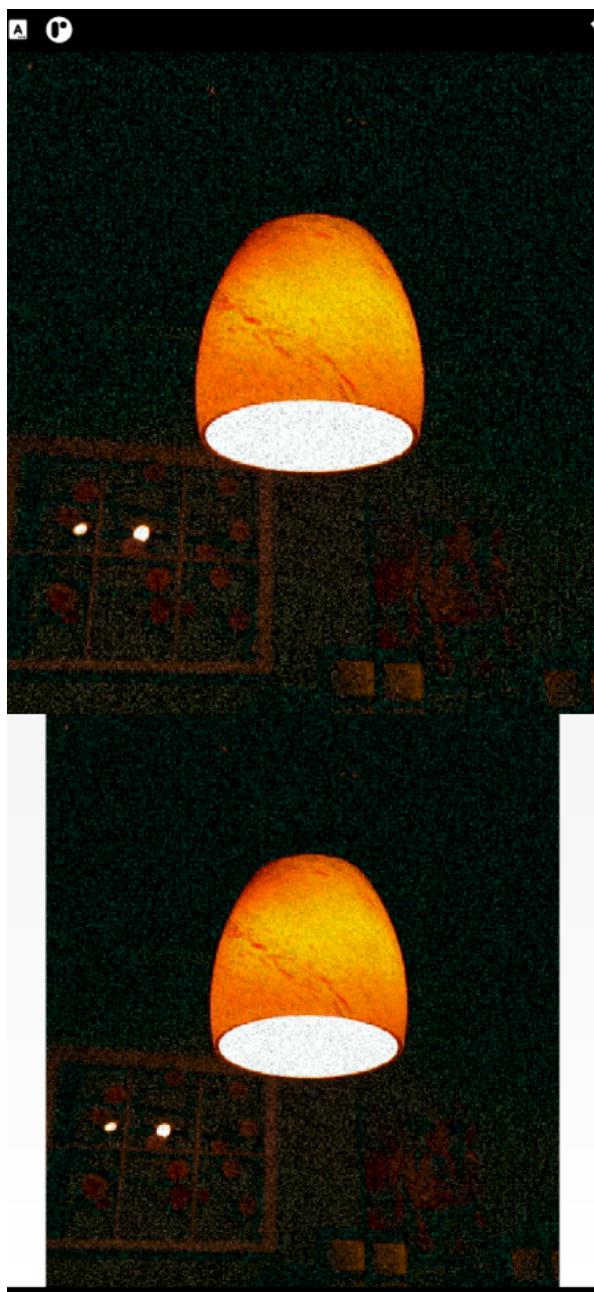
Testet med en bild av lampan skadad av Gaussiskt brus med följande parametrar: $mean = 0.01$, $variance = 0.01$ på en Android emulator visas i Figur 4.2. Brusreduceringen i båda exemplen är synlig men andra metoder skulle nog ge ett bättre resultat.



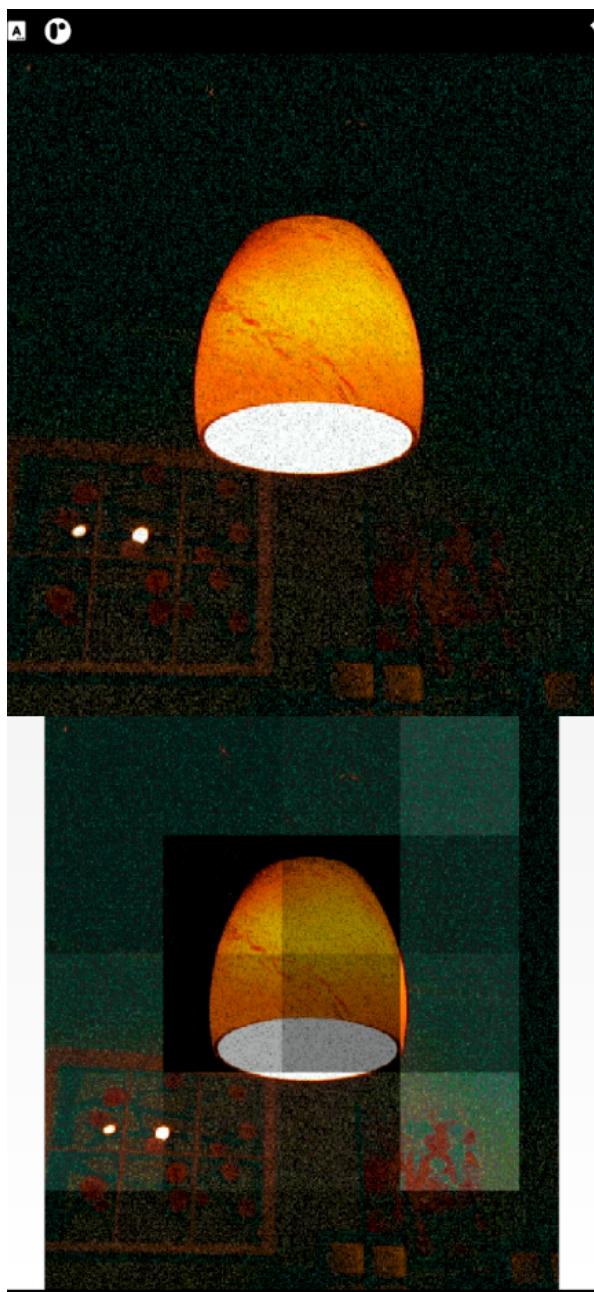
Figur 4.2: Lampan med Gaussiskt brus (0.01, 0.01) (upp)
och efter brusreducering soft tröskel nivå 3 (ner)



Figur 4.3: Lampan med Gaussiskt brus (0.01, 0.01) (upp)
och efter brusreducering hard tröskel nivå 3 (ner)



Figur 4.4: Lampan med Gaussiskt brus (0.01, 0.01) (upp)
och efter brusreducering hard tröskel nivå 1 (ner)



Figur 4.5: Lampan med Gaussiskt brus (0.01, 0.01) (upp)
och efter brusreducering soft tröskel nivå 9 med synliga artefakter (ner)

Litteratur

- [1] Wikipedia. "Sinne". I: (2020). URL: <https://sv.wikipedia.org/wiki/Sinne>. (besökt: 29.06.2021).
- [2] Anders A.; Lamis A.; Alvaro G. "DATORTOMOGRAFI SOM UNDER-SÖKNINGSMETOD FÖR UNG OCH GAMMAL, SPRUTAD OCH GJUTEN BETONG FÖR TUNNLAR". I: (2016). ISSN 1104 – 1773, ISRN BEFO-R—165—SE, BeFo Rapport 165. URL: https://www.befoonline.org/UserFiles/Archive/706/BeFo-Rapport_165_webb.pdf. (besökt: 01.07.2021).
- [3] Wencheng Wang m. fl. "An Experiment-Based Review of Low-Light Image Enhancement Methods". I: *IEEE Access* 8 (2020), s. 87884–87917. DOI: [10.1109/ACCESS.2020.2992749](https://doi.org/10.1109/ACCESS.2020.2992749).
- [4] Vertexshare Software Ltd. *AI Image Enlarger*. Android app. URL: <https://play.google.com/store/apps/details?id=com.app.aiimglarger>. (besökt: 04.07.2021).
- [5] PIXOCIAL TECHNOLOGY (SINGAPORE) PTE. LTD. *AirBrush*. Android app. URL: <https://play.google.com/store/apps/details?id=com.magicv.airbrush>. (besökt: 04.07.2021).
- [6] risingcabbage. *EnhanceFox*. Android app. URL: <https://play.google.com/store/apps/details?id=com.changpeng.enhancefox>. (besökt: 04.07.2021).
- [7] InShot Inc. *Lumii*. Android app. URL: <https://play.google.com/store/apps/details?id=photo.editor.photoeditor.filtersforpictures>. (besökt: 04.07.2021).
- [8] Adobe. *Photoshop Express*. Android app. URL: <https://play.google.com/store/apps/details?id=com.adobe.psmobile>. (besökt: 04.07.2021).
- [9] ernata derli. *Remini*. Android app. URL: https://play.google.com/store/apps/details?id=com.guide_enhance_old_photos_and_low_quality_photos_enhance_photos_taken_with_old_cameras.guia_remini_repairing_blurred_videos. (besökt: 04.07.2021).

- [10] Xiaojie Guo. "LIME: A Method for Low-light Image Enhancement". I: (2016). URL: <https://arxiv.org/pdf/1605.05034.pdf>. (besökt: 05.07.2021).
- [11] Puneesh Deora Bhavya Vasudeva. "Low-light Image Enhancement". I: (Okänt år). URL: https://drive.google.com/file/d/1aph-GUsr_Br2dMLTR3e0kYqAM5aThmj1/view. (besökt: 09.08.2021).
- [12] Yves Nievergelt. *Wavelets Made Easy*. Birkhäuser, 2013. ISBN: 978-1-4614-6005-3.
- [13] S.Kother Mohideen och et al. "Image De-noising using Discrete Wavelet transform". I: In *Proceedings of International Journal of Computer Science and Network Security IJCSNS* 8.1 (January 2008), s. 213–216. URL: https://www.researchgate.net/publication/251938085_Image_Denoising_using_Discrete_Wavelet_transform. (besökt: 09.08.2021).
- [14] Anke Meyer-Baese och Volker Schmid. *Pattern Recognition and Signal Analysis in Medical Imaging*. Academic Press, 2014. ISBN: 978-0-12-409545-8. DOI: <https://doi.org/10.1016/C2012-0-00347-X>.
- [15] MaTlaB Adda. *Thresholding*. Mathworks forum kommentar. URL: <https://se.mathworks.com/matlabcentral/answers/498939-image-denoising-using-wavelet-transform>. (besökt: 09.08.2021).
- [16] Robert Sedgewick och Kevin Wayne. *Base Matrix class*. Java class. URL: <https://introcs.cs.princeton.edu/java/95linear/Matrix.java.html>. (besökt: 30.08.2021).