NTNU
Det skapende universitet

Automatisk visuelt inspeksjonssystem

Per Gunnar Bårdsen

Master i datateknikk
Oppgaven levert: Juni 2006
Hovedveileder: Ketil Bø, IDI

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

Problembeskrivelse

Oppgaven går ut på å utvikle en generell modul for automatisk, videobasert kvalitetskontroll av en klasse objekter. Sytemet må kunne settes opp ved hjelp av templates og/eller parametre og være i stand til å plukke ut objekter som ikke fyller kvalitetskravene med hensyn til form, farge, mangler, skader etc.

Oppgaven gitt: 2006-01-20
Hovedveileder: Ketil Bø, IDI

Forord

Denne rapporten presenterer resultatet av min hovedoppgave i TDT4900 Datateknikk og informasjonsvitenskap ved Norges teknisk-naturvitenskapelige universitet, NTNU. Arbeidet er rettet mot et automatisk visuelt inspeksjonssystem, og har gått over vårsemesteret 2006. Som et fundament for dette, ligger teori og bakgrunn fra mitt fordypningsprosjekt i TDT4725 Bildebehandling, *Automatisk visuelt inspeksjonssystem for generelle objekter* [1], som gikk over høstesemesteret 2005.

Jeg vil takke min veileder, Ketil Bø, som har vært tilgjengelig og behjelpelig gjennom hele perioden.

Trondheim, juni 2006

Per Gunnar Bårdsen

Sammendrag

Denne hovedoppgaven følger opp mitt fordypningsprosjekt [1] med en praktisk implementasjon av et automatisk visuelt inspeksjonssystem. På bakgrunn av en serie av treningsbilder er målsetningen at systemet skal kunne klassifisere objekters avbildninger som godkjent eller underkjent. Arbeidet har lagt stor vekt på at systemet skal virke på generelle objekter. Systemet er implementert i Microsoft Visual Studio .NET 2003 C++, og viktige elementer tilknyttet arbeidet beskrives i denne rapporten. Resultatene virker lovende, da inspeksjonssystemet gjennomsnittlig klassifiserer 91% riktig på de 8 bildesett som systemet er testet med. Videre planer gir imidlertid håp om å utbedre systemet betydelig. Disse planene presenteres som videre arbeid i slutten av rapporten.

Innhold

Figurer

Tabeller

Kapittel 1

Innledning

Dette kapittelet beskriver oppgaven, og gir en oversikt over rapportens innhold og oppbygning.

Oppgaven går essensielt ut på å utarbeide et system, som kan trekke ut egenskaper i bilder av defekte og ikke-defekte objekter på generelt basis. Disse egenskapene skal så anvendes til å lære opp systemet om hva som kjennetegner et defekt og et ikke-defekt objekt. Målsetningen er at systemet skal være i stand til å klassifisere objektavbildninger som godkjent eller underkjent. Her tenkes det anvendt metoder innenfor kunstig intelligens. Hovedproblemstillingen ved prosjektet vil være å finne en fremgangsmåte som velger ut tilstrekkelig gode egenskaper fra bildene. For å begrense oppgaven antas gode forhold (tilstrekkelig lys, lite støy, osv.) i avbildningsfasen. Basert på dette utelukkes forhåndsprosessering som et nødvendig steg i forkant av segmenteringsfasen. I en typisk industriell sammenheng anser vi mulighetene til å kontrollere forholdene i avbildingsfasen som gode.

Rapporten er foruten dette kapittelet bygd opp av 10 kapitler. Kapittel 2 til 6 er hentet fra fordypningsprosjektet. Disse er gjentatt her for at leseren skal få en komplett oversikt over bakgrunn og "state-of-the-art" metoder. Alle disse utgjør viktig bakgrunnsinformasjon for denne hovedoppgaven.

Kapittel 2 beskriver flere velkjente segmenteringsmetoder fra litteraturen. Disse metodene er ordnet på en kategorisk måte, og kapittelet er tenkt som et oversiktskapittel over noen av de aktuelle metodene som kan anvendes i dette arbeidet. Det er verdt å merke at kapitlet er utvidet med en "ny" segmenteringsmetode (*Richard Blake's fargesegmentering*, se 2.3.5) i forhold til fordypningsprosjektet.

Kapittel 3 omhandler egenskapsbeskrivelse. Kapittelet beskriver flere måter å representere karakteristiske trekk i objekter på. Samtlige teknikker som

presenteres her er hentet fra litteraturen. Dette er i likhet med kapittel 2 tenkt som et oversiktskapitttel. Alle disse metodene er derimot aktuelle for hovedoppgaven.

Kapittel 4 tar for seg egenskapsuttrekking. Kapittelet gir en typisk oppskrift på hvordan seleksjon av egenskaper kan gjøres. Et stikkord her er datagruvedrift. Oppskriften er hentet fra litteraturen. Inspeksjonssystemet har riktignok ikke implementert metoder med bakgrunn i dette kapittelet. Kapittelet fortjener til tross for dette sin plass i rapporten, da strategien og metodene er attraktive i forbindelse med en videre utvikling av inspeksjonssystemet.

Kapittel 5 presenterer fire "state-of-the-art" strategier for mer eller mindre automatisert visuell inspeksjon av generelle objekter. Disse strategiene baseres på forskjellige innfallsvinkler, og er alle hentet fra litteraturen.

Kapittel 6 beskriver mitt eget løsningsforslag for fleksibel automatisk visuell inspeksjon. Løsningsforslaget presenteres som en modul, der diagrammer brukes som et hovedverktøy for modulbeskrivelse. Kapitlet er oppdatert med viktige elementer som ikke var tatt høyde for i fordypningsprosjektets rapport.

Kapittel 7 presenterer inspeksjonssystemets konstruksjon hovedsakelig i form av klasse og sekvensdiagrammer. Hensikten er å få frem oppbygning og logisk flyt i inpseksjonssystemet. Diagrammene er støttet opp med tekstlig beskrivelse.

Kapittel 8 tar for seg implementasjonen av noen utvalgte deler av systemet. Tanken er at beskrivelsen skal gjøre det lettere for utenforstående å forstå og ta i bruk systemet.

Kapittel 9 presenterer tester og resultater som er foretatt på inspeksjonssystemet. Dette inkluderer beskrivelse av blant annet bilder, konfigurasjonsinnstillinger, eksekveringstider og klassifiseringer.

Kapittel 10 beskriver viktige analyser av systemet. Analysen omfatter blant annet treningssett, egenskapskandidater, konfigurasjonsinnstillinger, kjøretider og nytteverdi for inspeksjonssystemet.

Kapittel 11 gir en beskrivelse av videre arbeid som tar sikte på å utbedre inspeksjonssystemet.

Kapittel 2

Segmentering

Målet med segmentering er å dele inn et bilde i forskjellige regioner. Disse regionene representerer de objektene i bildet som vi er interessert i, og segmenteringen må derfor være i stand til å skille objekter fra andre objekter, og fra bakgrunnen. Det er ønskelig at regionene oppfyller følgende kriterier [2]:

1. Alle piksler i bildet tilordnes en region.
2. Piksler innenfor en region er i kontakt med hverandre.
3. En region består av piksler som deler en eller annen felles egenskap.
4. Ingen eller minst mulig overlapp mellom forskjellige regioner og deres egenskaper.

Det eksisterer hundrevis av segmenteringsteknikker i litteraturen [3]. Mange av disse er gode for visse typer bilder, men ingen er derimot gode for alle typer bilder. Strategier for segmentering baseres generelt på konseptet om diskontinuitet eller konseptet om likhet [2, 4]. I konseptet om diskontinuitet ønsker man å dele bilder der hvor intensitetsverdier endrer seg plutselig. Dette er typisk for kantbaserte segmenteringsteknikker som beskrives i avsnitt 2.2. I konseptet om likhet ønsker man å samle piksler med felles egenskaper, og regionbaserte segmenteringsteknikker som beskrives i avsnitt 2.3 er et godt eksempel på dette. Terskelbasert segmentering utgjør en tredje variant, og denne beskrives i avsnitt 2.1. Avsnitt 2.4 beskriver deformerbare konturer som en fjerde variant, før kapittelet avslutter med å nevne noen andre segmenteringsteknikker i avsitt 2.5. Dette kapittelet beskriver bare et lite utvalg av metoder og kategorier for segmentering. Strukturingen av disse er ikke absolutt, og flere teknikker går på tvers av kategoriene de er delt inn i.

2.1 Terskelbasert segmentering

Det finnes en hel rekke tersklingsmetoder for å segmentere bilder. Felles for mange er at de er beregningsmessig billige i forhold til mange andre segmenteringsmetoder. Terskelbasert segmentering [5] er spesielt godt egnet i tilfeller der objekter ikke er i kontakt med hverandre, og deres intensitesverdier er ulik bakgrunnen. Metodene er også egnet når de interessante objektene i bildet har tilstrekkelig forskjell i intensitetsverdier. I de neste avsnittene beskrives et lite utvalg av terskelbasert segmenteringsmetoder, som er en av mange måter å strukturere dette utvalget på.

Fundamental terskling beskrives i avsnitt 2.1.1, histogrambasert terskling i avsnitt 2.1.2, flerspektral og klusterbasert terskling i avsnitt 2.1.3, og hierarkisk terskling i avsnitt 2.1.4.

2.1.1 Fundamental terskling

Den enkleste formen for terskling er binær segmentering [6]. En typisk anvendelse er å skille objekter fra bakgrunn, og matematisk kan denne beskrives på følgende måte:

$$g(x,y) = \begin{cases} 1 & \text{hvis } f(x,y) \geq T \\ 0 & \text{hvis } f(x,y) < T \end{cases} \tag{2.1}$$

der

$f(x,y)$ er intensiteten i bildet,
T er terskelverdi,
$g(x,y)$ er det binære segmenteringsresultatet

Dersom man ønsker å skille flere objekter fra hverandre i tillegg til bakgrunn, kan man innføre flere terskelverdier:

$$g(x,y) = \begin{cases} 0 & \text{hvis } 0 \leq f(x,y) \leq t_1 \\ 1 & \text{hvis } t_1 < f(x,y) \leq t_2 \\ \dots & \\ N & \text{hvis } t_N < f(x,y) \leq G-1 \end{cases} \tag{2.2}$$

der

$f(x, y)$ er intensiteten i bildet,
$t_1, \ldots, t_N$ er terskelverdier,
$g(x, y)$ er segmenteringsresultatet
G er antall gråtoner i bildet

Ved global terskling [2] segmenterer man hele bildet etter samme terskelverdi(er). Det er vanlig å terskle på intensitetsverdier, men man kan gjerne bruke gradienter (se 2.2.1), tekstur eller andre konstruktive egenskaper i bildet. Det kan i mange tilfeller være vanskelig å sette gode terskelverdier, og særlig dersom bildet er preget av ujevn belysningsrefleksjon og støy. Dersom man har apriori informasjon om forholdet mellom bakgrunn og interessante objekter i bildet, kan *p-tile* segmentering [2] brukes. Denne metoden leter etter en terskel (ved å summerere i histogrammet) som splitter bildet slik at 1/p av bildearealet har verdier mindre enn terskelen. En annen metode som finner en global terskel for binær segmentering beskrives av Ridler og Calvard [7, 8]. Dette er en iterativ prosedyre som uten hjelp av apriori informasjon arbeider seg frem til en ny terskel i hver iterasjon. Metoden bygger på binær segmentering med forrige terskel og gjennomsnittsberegninger av pikselverdier i hver iterasjon. Til slutt vil konvergens oppstå når den nye terskelen blir tilstrekkelig lik den gamle. Mange andre globale tersklingsmetoder benytter seg av histogam analyse, som beskrives i avsnitt 2.1.2.

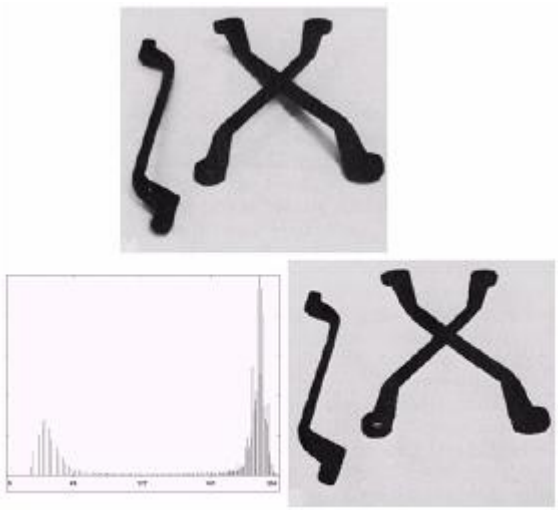
I dynamisk terskling (*dynamic/adaptive threshold*) [6] kan terskelverdiene variere over bildet. Faktorer som påviker resultatet er intensitet, posisjonen i bildet, og lokale egenskaper i pikslene rundt denne. En måte er å bruke dynamisk terskling på er å dele bildet inn i flere småbilder, og bestemme en lokal terskelverdi for hvert bilde. Alle småbildene segmenteres da med sin lokale terskel. Denne metoden gir for eksempel mye bedre resultater på bilder preget av ujevn belysningsrefleksjon i forhold til globale tersklingsmetoder.

2.1.2 Histogrambasert terskling

Et histogram er en 1-dimensjonal fremstilling av et bilde som er beregningsmessig billig å gjennomføre. Histogrammer besitter i tillegg analytiske egenskaper som gjør det attraktivt for terskelbasert segmentering. Dette er typisk forskjellige intensitetsverdiintervaller (modes), som er separert av daler, og i mer eller mindre grad korresponderer til regioner i bildet. I binær segmentering antar vi for eksempel at objekter og bakgrunn utgjør hver sin mode i det bimodale histogrammet. Kunsten med å sette terskelverdier blir da det samme som å finne dalene mellom disse intensitetsverdiintervallene.

Det er mange fremgangsmåter innnen histogrambasert terskling, og overordnet kan vi gjøre et skille mellom ikke-kontekstuelle og kontekstuelle metoder [3]. Ikke-konseptuelle metoder baseres utelukkende på diverse prosessering av bildets histogramfremstilling. Dette innebærer at romlige naboskap og andre relasjoner mellom piksler ignoreres. Segmenteringsresultater av ikke-konseptuelle metoder bør derfor sjekkes opp mot originalbildet. Konseptuelle metoder tar i tillegg til analyse av histogrammet også hensyn til relasjoner mellom pikslene i bildet. Dette kan for eksempel være lokale egenskaper som kanter (gradient, *Laplacian*) [2]. Tanken er at disse relasjonene skal kunne forbedre segmenteringsresultatene.

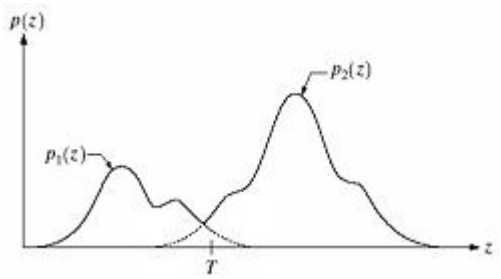
Figur 2.1 (kopiert fra [6]) viser et eksempel på en global terskling av et bilde, der terkskelen ligger mellom de to søylene i bildets histogram. Figuren viser øverst originalbildet, dets histogrammet, og tersklingsresultatet nederst til høyre.



Figur 2.1: Ikke-konseptuell global terskling

Mode metoden [2] er et eksempel på en ikke-konseptuell teknikk som finner approksimert(e) terskelverdi(er) for bi- eller multimodale histogram. Metoden baseres på å først finne de høyeste lokale maksima, for så deretter å finne terskel som et minimum mellom disse. Det kan være en fordel å arbeide med et glattet histogram for å jevne ut lokal "krangel" om minima og maksima, men man må da passe på å ikke glatte ut små modes av relevans i bildet. En annen utfordring kan være å presist lokalisere terskelverdi i bi- eller multimodale histogrammer med brede og flate daler. Felles for mange ikke-konseptuelle metoder er at de anvender statistiske sannsynlighetsfordelinger for å estimere terskelverdier. Disse kalles *optimal thresholding* [5, 8], som forsøker å approksimere histogrammet med en vektet sum

av et antall sannsynlighetsfordelinger (for eksempel normalfordelinger), for så å finne terskel(er) som korresponderer til minimal sannsynlighet mellom disse fordelingene. Målsetningen er å komme opp med et minst mulig antall feilsegmenterte piksler (*minimum error segmentation*). Vanskeligheter med disse metodene er å velge passende sannsynlighetsfordelinger, samt å bestemme parametere til disse. Strategier for å redusere disse vanskelighetene kan blant annet være å maksimalisere varians mellom objekter og bakgrunn, minimalisere variansen i histogrammet, og summere opp kvadrert feil [5]. Det er verdt å merke at parametriske metoder nok i de fleste tilfeller generelt krever høyere kjøretid enn metoder som ikke søker frem parametere. Figure 2.2 (kopiert fra [6]) illustrerer en multimodal glattet histogramapproksimasjon, der T markerer gunstig terskelverdi.



Figur 2.2: Terskelverdi i en multimodal glattet histogramapproksimasjon

En kjent strategi blant noen av de konseptuelle metoder er å bruke kantinformasjon til å utbedre forholdet mellom topper og bunner i histogrammet [4, 5]. Dette vil i sin tur gjøre det enklere å finne terskelverdier. Gjennomføringen kan skje ved først å kjøre et gradientfilter over bildet, for så deretter å vektlegge piksler med lave gradientverdier i histogramkonstruksjonen. På dette viset vil piksler som representerer kanter nedprioriteres, og dette vil merkes som dypere daler i histogrammet, siden kanter typisk representerer områder mellom pikselverdiintervaller av objekter og bakgrunn. Ved å vektlegge høye gradientverdier vil man oppnå motsatt effekt. Da vil objekter og bakgrunn nedprioriteres, og histogrammet vil få topper i posisjoner som tilsvarer kanter og konturer i bildet. Resultatet blir et unimodal histogram, der toppunktet er en approksimert terskelverdi for å skille mellom objekter og bakgrunn.

2.1.3 Flerspektral og klusterbasert terskling

En form for flerspektral terskling er å gjøre en uavhengig terskelbasert segmentering for hvert spektralbånd av det flerspektrale bildet [5]. Disse delresultatene kan så settes sammen til et endelig resultat som inneholder hele spekteret, og utgjør da det ferdigsegmenterte bildet. Prosedyren kan gjentas rekursivt helt til visse endekriterier oppnås. Dersom man for eksempel

benytter seg av histogrammer for å detektere terskelverider (som beskrevet i avsnitt 2.1.2), kan man slutte når hver komponent representeres ved unimodale histogrammer.

En annen strategi i flerspektral terskling er å lage et n-dimensjonalt histogram, der hver dimensjon tilsvarer histogrammet til ett av spektralbåndene [6]. I tilfeller av fargebilder vil man da konstruere en 3-dimensjonalt histogram, der histogrammet til det røde, grønne og blå histogrammet utgjør hver sin dimensjon. Det neste steget blir så å finne og tolke klustre i det flerdimensjonale histogramrommet, og til dette kan vi bruke passende klassifiseringsmetoder. En slik metode kan for eksempel gå ut på å tildele punkter i rommet til nærmest posisjonerte kluster (*minimum distance*), eller kluster med minst avvik i verdi (*minimum error*) [5]. Segmenteringsresultatet kan til slutt presenteres som regioner i det romlige bildet, som korresponderer til klustre i det flerdimensjonae klusterrommet.

2.1.4 Hierarkisk terskling

Et bilde kan representeres hierarkisk som et sett av forskjellige oppløsninger av seg selv [5]. Disse kan for eksempel beregnes rekursivt ved gjennomsnittsberegning av pikselverdier fra et bilde med høyere oppløsning. Segmentering i hierarkiske datastrukturer kan så foregå ved å beregne en terskel på et grovt nivå (lav oppløsning), for så å oppdatere krav til terskelen etter hvert som man graver seg ned i bilder med høyere oppløsning [5]. Man kan for eksempel la terskelen være et kriterium for signifikante kanter i bildet. Kantenes lokalitet vil forbedres etter hvert som man anvender neste bilde med høyere oppløsning, der man segmenterer om igjen de pikslene som er i posisjoner i og rundt kantene fra bildet med lavere oppløsning. En fordel med hierakisk segmentering er at støy har liten innvirkning på resultatet, siden segmenteringene på de lavere oppløsningene er basert på glattede data, der støy i stor grad er filtrert bort [5].

2.2 Kantbasert segmentering

Kantdeteksjon kan overordnet deles inn i parallelle og sekvensielle teknikker [2, 3]. I sistnevnte vil spørsmålet om en gitt piksel er en kant være avhengig av resultatene av tidligere undersøkelser av andre piksler. Parallelle teknikker har ikke denne avhengigheten. Der baseres svaret "bare" på pikselen selv og noen av dens naboer. Dette innebærer at man i parallelle metoder prinsipielt kan undersøke flere piksler i bildet samtidig, mens man i sekvensielle metoder må velge et egnet startpunkt, og jobbe med en piksel av gangen.

I kantbasert segmentering ønsker man å finne sammenhengende og lukkede konturer (*closed boundaries*). Parallelle teknikker gir ikke nødvendigvis slike

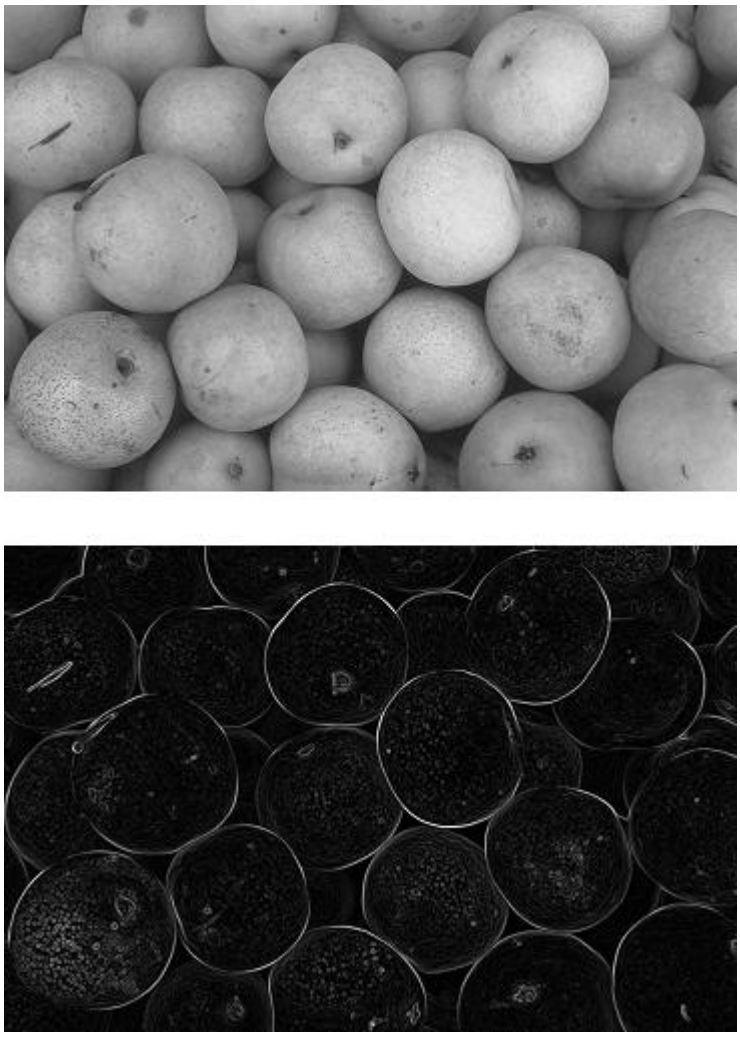
sammenhenger, og man kan derfor risikere å måtte koble sammen kantsegmentene i etterkant. Dette kan gjøres med sekvensielle teknikker som for eksempel *heuristisk søk* (se 2.2.2). Et lite utvalg av parallelle teknikker beskrives i avsnitt 2.2.1, og to sekvensielle teknikker beskrives i avsnitt 2.2.2.

2.2.1 Parallelle teknikker

En metode for kantdeteksjon er å filtrere bort lavfrekvent bildeinformasjon i frekvensdomenet ved hjelp av *Fourier transform* og høypassfilter [2, 6]. På dette viset står man etter *invers Fouriertransformasjon* igjen med skarpe intensitetsforandringer i bildet som kjennetegner kanter, men også støy. Det kan være problematisk å finne gode høypassfiltre i frekvensdomenet, der "godkjente" frekvenser korresponderer til rikige kanter i hele det romlige bildet.

Gradient operatorer utgjør en annen metode for kantdeteksjon [2, 3, 5, 6]. I digitale bilder approksimeres disse med differanser mellom intensitetsverdier, som igjen kan representeres som konvolusjonsmasker. Disse maskene multipliseres med bildet, og for eksempel ved hjelp av en global terskel kan man avgjøre om resultatene utgjør kanter i bestemte posisjoner bildet. Det finnes mange slike varianter av gradientmasker (for eksempel *Roberts*, *Prewitt*, *Sobel*, *Laplacian*). Disse skiller seg fra hverandre med hvordan de approksimerer derivasjon (med forskjellig vektligging av kantforhold i maskene og maskedimensjon). For eksempel er *Laplacian* en andre ordens derivert approksimasjon som ikke gir kantretning men kun gradientstørrelse, i motsetning til første ordens deriverte som *Roberts*, *Prewitt* og *Sobel* operatorene. Gradientmetodene gir sterke responser på hjørner, linjer og isolerte punkter. Det følger da at disse også er svært følsomme for støy, og andre ordens deriverte er enda mer sårbare enn første ordens deriverte. For eksempel vil *Laplacian* i støyete bilder med lav kontrast kunne gi høyere respons for støy enn for kanter [3]. Andre ordens deriverte har dog fordeler som første ordens deriverte ikke har. De kan si oss om en piksel befinner seg på mørk eller lys side av en kant, og de kan bestemme kantens senter med "zero-crossing" egenskapen. Sistnevnte gir opphav til en tynnere representatsjon av kanter enn første ordens deriverte, og kan naturligvis være til spesielt god nytte i bilder med tjukke kanter [6]. Ulempene i forhold til første ordens gradienter er at de ikke detekterer retning på kantene, samt at de gir en mer komplisert segmentering som følge av dobbel respons på hver kant [6]. Ved å glatte andre ordens deriverte som Laplacian, kan man gjøre kantdeteksjonen mer robust ved å redusere responsen fra støy. En slik metode er *Laplacian of Gausian* (*LoG*) [3, 5, 6], som også kan representeres som en konvolusjonsmaske. Denne glatter *Laplacian* med en normalfordeling, og avhenger derfor av parameteren sigma (standardavvik). Valg av sigma bestemmer hvor stor innflytelse piksler rundt om i bildet skal ha på piksel som undersøkes. Fordelen med *LoG* framfor klassiske metoder er at et større areal i nabolaget kan få inflytelse,

samtidig som at inflytelsen synker jo lengre unna senteret man beveger seg [5]. Ulemper er at kantfassongen kan glattes for mye, og det kan dannes lukkede konturer (*closed loops*, "spagetti effekt") [5, 6]. En felles ulempe for gradient operatorer generelt er deres avhengighet av å kjenne til størrelsen til objektene i bildet [5]. Figur 2.3 (kopiert fra et eksempel i Matlab 7.0.1, *http://www.mathworks.com*) viser et originalbilde og Sobelmaskegradienten av dette.



Figur 2.3: Et originalbilde og Sobelmaskegradienten av dette

En svært enkel metode som også bruker diskret differanse mellom intensitetsverdier, er å trekke et noe poisjonstransformert bilde fra seg selv. Denne strategien ble gjort kjent for meg av professor Richard Blake i kurset datasyn [9]. Ved å la transformasjonen tilsvare bredden på kanter av interesse vil man approksimere disse kantene i bildet, da regioninnhold i stor grad vil

nulles ut og kantene vil komme frem som intensitetsforskjeller mellom kanter og ikke-kanter. Strategien bygger selvfølgelig på at man kjenner interessante kantbredder på forhånd. Ulemper med metoden er at den i utganspunktet bare finner en av mange kantbredder av interesse, og det er relativt stor fare for falske kanter på grunn av inhomoge pikselverdier i andre posisjoner enn de som representeres av faktiske kanter.

Edge relaxation er en iterativ metode som på bakgrunn av intensiteten til piksler og deres nabopiksler vurderer om pikslene tilhører en kant eller ikke [2, 5]. Hver piksel tildeles først en kanttillit ved tolkning av deres og nabolagets intensitetsverdier. For hver iterasjon vil så disse kanttillitene styrkes eller svekkes som følge av en felles betraktning av deres nåværende verdi og verdiene i nabolaget. For eksempel vil en kant som ikke har noen andre kanter rundt seg svekkes, mens en svak kant som befinner seg mellom to sterke kanter vil styrkes. Konvergens oppnås etter tilstrekkelig antall iterasjoner. Fordeler med denne metoden er at den tar hensyn til romlig kantkontekst, og den er godt egnet til parallelle implementeringer. Ulemper er at det kan være vanskelig å bestemme funksjoner som oppdaterer kanttillitene godt, og det kan ta tid å oppnå konvergens.

Hough transform er en alternativ metode som er godt egnet til å detektere kanter med ønskede fassonger globalt i bildet [2, 5, 6]. Slike fassonger kan for eksempel være linjer, sirkler eller kurver. Metoden baserer seg på å finne de teoretiske fassonger som samsvarer best med de faktiske fassongene som er representert i bildet. For å få til dette brukes et parameterrom der de ulike teoretiske fassongene korresponderer til ulike posisjoner i rommet. Etter hvert som itererer gjennom det faktiske bildet og inkrementerer de posisjoner i parameterrommet som stemmer overens, vil parameterrommet sitte med tilstrekkelig informasjon til å bestemme fassongene i det faktiske bildet. Valg av essensielle posisjoner i parameterrommet kan gjøres enkelt ved terskling. Det kan være en fordel å kjøre et gradientfilter på inputbildet for å spare regnekraft. Ulemper med metoden er at den er sensitiv med hensyn på kvantifisering av parameterrommet, og i noen tilfeller er det ikke ønskelig at den finner fassonger som er bygd opp av ikke-sammengengende piksler [2]. For eksempel kan en linje oppnå feil vinkel fordi deteksjonen blir forstyrret av ikke-relevante punkter i en annen del av bildet. Til tross for dette er sistnevnte stort sett en styrke fordi den kan brukes til å detektere støyfulle, stiplede, overlappende eller halv-dekkende objekter [5]. *Hough transform* er også robust med hensyn på forstyrrelse av andre strukturer enn den man søker etter, som måtte befinne seg i bildet [5].

2.2.2 Sekvensielle teknikker

Heuristisk søk er en sekvensiell strategi for å bestemme kanter i et bilde [2, 5, 6]. Teknikken er et graf søk der man begrenser søkerommet ved hjelp av en heuristikk, som for eksempel kan baseres på retning, styrke, kurving, distanse eller apriori informasjon om kantene. Hver piksel i bildet kan typisk representeres som en node i grafen, og kanter i bildet vil da korrespondere til stier i grafen. De piksler som presterer best under evalueringen favoriseres, og i tilknyttning til disse finner man ved neste iterasjon de nye pikslene som leder an. På denne måten bygger man opp en sti som til slutt gir oss en kant vi søker i bildet. Det kan være problematisk å finne gode evalueringsfunksjoner, og et feilaktig retningsvalg i en tidlig fase i grafen kan gi store avvik fra det resultatet man ønsker i algoritmer som ikke støtter tilbakesporing [2]. Denne ulempen er direkte påvirket av hvilken heuristikk man velger, samt vekting av optimalitet opp mot raskere kjøretid.

Dynamisk programmering kan anvende sitt prinsipp om optimalitet til å finne kanter i et bilde [2, 5, 10]. Den optimale løsningen vil da være bygget opp av optimale løsninger på delproblemer. I et kantdeteksjonsproblem bygger man da gjerne opp kostnader av kantpikselsammengener som man ønsker å minimalisere. Den endelige løsningen er garantert optimal i forhold til de funksjoner som er brukt til å beskrive kostnad mellom piksler.

Heuristisk søk kan være mer effektivt enn *dynamisk programmering* når det gjelder å finne en sti mellom to noder i en graf dersom en god heuristikk er tilgjengelig [5]. En grunn til dette er at *dynamisk programmering* bygger opp hele løsningen fra bunn av, hvilket kan prestere dårligere enn *top-down strategier* som kan gå raskere til dybden og løsning i grafen. *Dynamisk programmering* er derimot effektiv til å beregne optimale stier mellom flere start og sluttpunkter, slik at den beste av disse to metodene avhenger generelt av evalueringsfunksjonene og kvaliteten i heuristikken [5].

2.3 Regionbasert segmentering

Regionbaserte segmenteringsteknikker [6] kan generelt sies å prestere bedre i bilder med støy enn kantbaserte segmenteringsteknikker [5]. Det er også verdt å merke at regionbaserte metoder tar hensyn til romlige lokalisjoner og alltid resulterer i lukkede konturer (*closed boundaries*), i motsetning til kantbaserte og tersklingsbaserte metoder, som ikke nødvendigvis oppfyller dette. Dersom man skal sammenligne tersklingteknikker som behøver flere terskler med regionbasterte teknikker, så løses segmenteringen generelt best med sistnevnte [6].

Ulemper med regionbasert segmentering er at resultatene avhenger av hvilken

sekvens man prosesserer pikslene, det er vanskelig å inkorporere global informasjon i segmenteringen, og implementeringene er iterative med høy kjøretid og høyt minneforbruk [2]. Ytterligere kan det nevnes at etterprosessering ofte er nødvendig på grunn av over- eller undersegmenterte resultater.

Region growing/merging beskrives i avsnitt 2.3.1, *region splitting* i avsnitt 2.3.2, *region splitting and merging* i avsnitt 2.3.3, *watershed* i avsnitt 2.3.4 og en spesifikk fargesegmenteringsmetode i 2.3.5

2.3.1 Region growing/merging

Region growing/merging [6] er en metode som baserer seg på å ekspandere regioner ved å legge til piksler eller andre regioner. Dette gjøres dersom visse kriterier er tilfredsstilt (kriterium 2 og 3 ved regioner), slik at homogenitet i regionen ivaretas. Slike kriterier kan for eksempel typisk være at gråtoner eller fargeverdier ikke avviker med mer enn en hvis terskel fra regionen, men de kan også være mer avanserte, for eksempel ved å ta hensyn til romlige posisjoner i bildet i tillegg til deres verdier (som kan være nødvendig i tekstur).

En naturlig fremgangsmåte ved metoden er å starte med en eller flere startpunkter, kalt "seed points". Disse er gjerne enkeltstående piksler, som danner et basis sett av regioner som det så vokser ut fra. En mulighet er å tilegne hver piksel i bildet ett "seed point". Dersom man ønsker et mindre antall startpunkter oppstår problemet med å bestemme disse på en tilfredsstillende måte. Dette kan være vanskelig, og særlig når apriori informasjon ikke er tilgjengelig. I slike tilfeller kan man for eksempel forsøke å bestemme disse startpunktene ved for eksempel å betrakte bildets egenskapsrom (*feature space*), og velge de piksler som ligger i sentre av klyngene (*clusters*) i egenskapsrommet [6]. Valg av passende egenskaper for å beregne egenskapsrom er generelt en utfordrende oppgave som beskrives i et senere kapittel i rapporten. Et annet alternativ kan være å ta i bruk histogramrepresentasjon av bildet. I slike tilfeller bruker man gjerne toppene i et glattet histogram til å estimere "seed points". Svakheten ved en slik fremgangsmåte i all sin ensomhet er at pikslenes lokalitet og naboskap (*connectivity*) i bildet ikke tas hensyn til. Dog vil det nok ofte være slik at regioner har noenlunde homogene gråtone eller fargeverdier, slik at metoden i mange tilfeller kan forsvares.

Resultatene av *region growing/merging* er i tillegg til selve kriteriene for å ekspandere regioner, også i betydelig grad avhengig av valg av startpunkter. Andre problemer med metoden kan være å sette stoppekriterier for når regionene skal stoppe å vokse. Dette gjelder særlig når apriori informasjon er tilgjengelig, og man baker inn informasjon som blant annet kontur, størrelse og pikselverdi i beslutningsprossen for vekst. [6]

Metoden kan selv med optimale vekstkriterier resultere i et uønsket antall regioner. Dette være seg resultater med for få regioner ("over-growing"), og resultater med for mange regioner ("under-growing") [5]. Sistnevnte opptrer gjerne som hull i andre større regioner som tilsvarer støy eller ubetydelige detaljer i bildet. Et tradisjonelt tiltak mot oversegmentering er å kjøre en eller flere *region growing post-processing* algoritmer [5], som eliminerer disse hullene. Dette skjer typisk ved å tildele dem til naboregioner med størst liket i homogenitet, eller ved mer avanserte metoder som tar i bruk kantinformasjon i tillegg til regioninformasjon. I tilfeller med undersegmentering kan man ta i bruk region splitting metoder som beskrives nærmere i neste avsnitt (2.3.2).

2.3.2 Region splitting

Region splitting [5] er en metode som baserer seg på å dele inn en inhomogen region i to eller flere nye regioner. Det kan være naturlig å starte med at hele bildet først representeres av en enkelt region. Prosedyren er å foretå splitting av regioner helt til alle regioner tilfredsstiller kravene om homogenitet. Det er verdt å merke at region splitting ikke nødvendigvis gir samme "løsning" som *region growing/merging*, selv når samme krav til homogenitet brukes [5]. Potensielle problemer tilknyttet metoden tilsvarer problemene til *region growing/merging*. I tillegg til dette kan metoden også resultere i naboregioner med felles homogene egenskaper seg imellom [6]. Dette problemet kan for eksempel lett oppstå hvis man splitter en region i flere regioner om gangen, som eksempelvis fire i tilfelle av en quadtree representasjon.

2.3.3 Splitting and merging

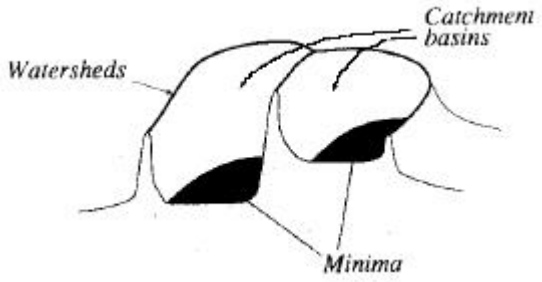
Ved å kombinere *growing/merging* og *splitting* teknikkene kan man oppnå fordeler fra begge teknikkene [5]. Man kan da lette på startkriteriene ved at regionene ikke trenger å være homogene internt eller inhomogene med sine naboer. En måte å kombinere disse to metodene på er å først kjøre en *region splitting* algoritme i sin helhet, før man så følger opp med en *region growing/merging* algoritme. På denne måten vil man overvinne svakheten med oversegmentering av en alenestående *region splitting* algoritme. I tillegg benyttes da at *region splitting* er mindre problematisk når "seed points" skal velges. Alternativt kan man også veksle mellom *splitting* og *growing/merging* underveis i segmenteringsprosessen. Problemområder med metoden vil fortsatt være å sette optimale kriterier for å splitte og slå sammen regioner.

2.3.4 Watershed

Det er vanlig å illustrere *watershed* metoden [11] som et tredimensjonalt landskap av det aktuelle bildet man ønsker å segmentere. Bildeplanet utgjør

da to av dimensjonene, mens høyden i landskapet representerer den siste. Sammenhengen mellom høydens proporsjonalitet med bildet kan bestemmes på flere måter. Blant annet kan bildets intensitetsverdier, distanseverdier og gradientverdier brukes [12]. Gradientverdier velges ofte på grunn av sin evne til å redusere mengden irrelevante data, eller alternativt, på grunn av sin evne til å trekke ut relevante data. Siden et objekt for det meste er en gruppe av piksler, som er separert fra omgivelsene med en større endring i intensitet enn innad i seg selv, vil gradienter gi større utslag langs konturen, og langt mindre utslag internt i objektet. Gradienter er derfor en vanlig metode for å estimere konturer (se kantbasert segmentering 2.2).

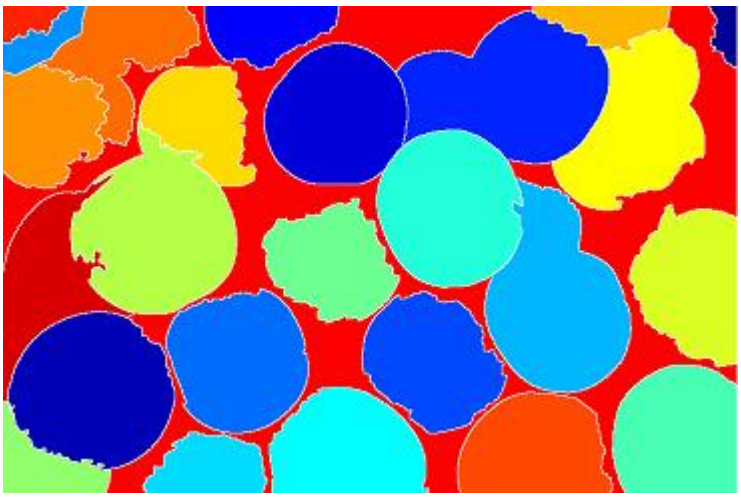
Man bruker gjerne ord som vannskillelinjer (*watershed-lines*) og vannoppsamlingbassenger (*catchment basins*) når man snakker om metoden. Vannskillelinjer representerer piksler som utgjør grenser mellom regioner i bildet, og alle piksler innenfor en region danner et vannoppsamlingsbasseng. Målet med metoden er å detektere vannskillelinjenes posisjoner i bildet. Strategien for dette er å "bore" hull i hvert av de regionale minima (den minste høyden i et vannoppsamlingsbasseng), for dernest å la vann stige oppover i landskapet. Vannstanden økes inkrementelt, og for hvert nivå dannes det nye vannoppsamlingsbassenger (regioner), og/eller gamle vannoppsamlingsbassenger utvides med nye vannivåpiksler, som tilfredstiller kravet om naboskap (connectivity). Unntak iverksettes hvis vannivået tangerer vannskillelinjer. Da "bygges" det dammer på disse posisjonene. Til slutt er hele landskapet dekket av vann. Alle piksler er da tilordnet et vannoppsamlingsbasseng, og vannskillelinjene er kartlagt av dammer i bildet. Figure 2.4 (kopiert fra [11]) illustrerer "watershedlandskapet" til et bilde.



Figur 2.4: Watershedlandskap

En rå *watershed* segmentering vil i all sin ensomhet ofte gi et betydelig oversegmentert resultat [5, 6, 11, 12]. Dette skyldes gjerne støy og andre kontraster i bildet. I litteraturen nevnes det flere fremgangsmåter for å bøte på dette. En er å anvende en *region merging* algoritme (se 2.3.1) i etterkant av watershedalgoritmen [13, 12]. En annen er å utvide watershedalgoritmen med bruk av markeringer (*markers*) [6, 13], som er et sett av koblede piskler

som setter krav til interessante objekter i bildet. Slike krav kan for eksempel være et visst antall koblede piksler med lik intensitet, som i tillegg utgjør regionale minima i bildet. På denne måten filtreres alle regionale minima som ikke tilfredstiller disse krave bort, og resultatet blir en betydelig reduksjon av mengden hull og vannoppsamlingsbassenger i landskapet. Et annet tiltak kan være å glatte bildet med en konvolusjonsmaske i forkant av beregning av gradientverdier [6]. Da vil gradientverdiene reduseres globalt, og man vil også da oppnå at små detaljer og støy utelukkes som regionale minima i landskapet. Det er verdt å nevne at en kombinasjon av disse kan gi ytterligere forbedringer [12]. En rå *watershed* metode har også andre svakheter. Vannskillelinjene kan resultere i tjukke platåer når to eller flere vannoppasmlingsbassenger konkurerer om pikselposisjoner [5], og signifikante konturer med lav kontrast og tynne strukturer detekteres dårlig [14]. Figur 2.5 (kopiert fra et eksempel i Matlab 7.0.1, *http://www.mathworks.com*) viser resultatet av eksekvert *watershed* med bruk av markeringer på bildet fra figur 2.3.



Figur 2.5: Watershedmetoden eksekvert på bildet fra figur 2.3

En av styrkene med *watershed* metoden er at den alltid gir sammenhengende grenser mellom regioner [6]. I mange tilfeller er det også en styrke at vannskillelinjene alltid korresponderer til de mest signifikante kantene [14]. Det vil si at selv på bilder uten sterke kanter vil *watershed* detektere konturer. En av de viktigste bruksområder for watershed kan derfor være å ekstrahere objekter med stor likhet i homogenitet fra bakgrunnen [6], eller separere objekter som er i kontakt med hverandre.

2.3.5 Fargesegmentering

På grunn av det faktum at fargebilder inneholder tre ganger så mye informasjon som gråtonebilder, ligger det et potensiale ved å utnytte fargeseg-

mentering fremfor gråtonesegmenteringer som *watershed*. Segmentering ved hjelp av farge fremstår derfor som mer attraktiv, og det er naturlig å tenke seg at gode fargesegmenteringsmetoder er mer til å stole på.

Richard Blake's fargesegmenteringsmetode [9] baserer seg på at hver fargepiksel er bygget opp en rød, grønn og blå komponent (r, g, b). Pikselens farge bestemmes av forholder mellom komponentenes verdier $(r : g : b)$, og pikselens intensitet kan beskrives som $\sqrt{(r^2 + g^2 + b^2)}$. Det er et viktig poeng at farge og intensitet er uavhengig. Teorien rundt denne metoden sier derimot at denne uavhengigheten ikke er tilfelle ved lave intensiteter. Forholdet $(r : g : b)$ er da forstyrret av støy. For å ta høyde for dette velger metoden å "se bort fra" slike lavintensitetspiksler i segmenteringsprosessen.

Metodens første steg er å normalisere bildet ved å skalere opp (r, g, b), slik at hver piksel har maksimal gråtoneverdi i en av sine fargekomponenter. Dette gjøres for at forskjeller mellom pikselverdier skal komme klarere frem. Neste steg er en glattingsprosedyre, som iterativt går over hver piksel i hele bidet og glatter ut piksler som antas å utgjøre innhold i regioner. Regionenes kanter forblir uglattet. Bak denne glattingsprosedyren ligger det variansberegninger av stor og liten maske sentrert i den aktuelle pikselen i iterasjonen. Maskene bestemmer hvilke nabopiksler som utgjør grunnlaget for variansberegningen. Dersom variansen i den store masken er mindre eller ikke betydelig større enn variansen i den lille masken, antar metoden at den aktuelle pikselen befinner seg innenfor en region. I så tilfelle vil metoden glatte den aktuelle piskelen med de nabopikslene som er bestemt av den lille masken. Denne glattingsprosessen gjentas så over hele bildet inntil antall pikselglattinger ebber ut. Det avsluttende steget blir så å spore regioner innenfor "vegger" av høy varians.

For å forbedre segmenteringsrsultatene foreslår Blake å følge opp metoden med *region growing*. Tanken er å redusere oversegmentering samt at piksler som ikke mappet til regioner i første omgang skal inkluderes. Som en teknikk for å hjelpe til med dette presenteres han et kromatisitetsdiagram, som kort og godt plotter hyppigheten av piksler i et plan der to av fargekomponentene (for eksempel r, g) danner hver sin akse. Ved å tolke klusteransamlinger i slike plan kan man estimere hvilke farger som sannsynligvis tilhører visse regioner, og mappinger mellom "kompatible farger" kan foretas. Selve *region growing* prosedyren foreslås som pikselvis iterasjon gjennom konturen til hver region, der hver piksel sjekkes opp mot nabopiksler utenfor regionen. Dersom en aktuell nabopiksel har en "kompatibel farge" og ikke tilhører noen annen region, så tildeles denne nabopikselen til samme region som den aktuelle pikselen. Det samme gjøres også dersom nabopikselen har en "kompatibel farge" som ligger nærmere fargen til region r_p enn region r_n, der r_p er regionen til den aktuelle pikselen og r_n er regionen til nabopikselen. Denne *region grow-*

ing-prosedyren gjentas helt inntil regionene når stabilitet.

Som en alternativ strategi for påfølgende *region growing* kan vi tenke oss å anvende regioners gjennomsnittlige fargeverdier i stedet for kromatisitetsdiagram. Tanken er å la større regioner vokse ved å "spise" opp piksler fra regioner med et areal som er under en gitt terskelverdi. "Spisingen" skjer ved å følge konturen til disse små regionene, og tildele konturpikslene til en større naboregion med størst likhet i gjennomsnittlig fargeverdi. Denne appellerer i forhold til Blake's variant hovedsakelig på grunn av dens enkelhet og robusthet. Det er ikke til å stikke under en stol at innføring av et kromatisitetsdiagram vil medføre økt kompleksitet, som kan gå på generalitetskravet løs. Her går tankene først og fremst på innstillinger for å generelt kunne avgjøre klustre og dets sentre samt "kompatible farger".

Teoriens eksterne grensesnitt ender alstå opp med disse fire nødvendige parameterne:

1. En terskel som angir et minstekrav til fargeintensitet. Piksler med intensiteter ($\sqrt{r^2 + g^2 + b^2}$) mindre enn denne terskelen vil ikke bidra i segmenteringsprosessen.

2. En terskel som angir maksimalt tillatte inhomogenitet i regioner med tanke på fargeverdier.

3. En terskel som angir øvre grense for antall iterasjoner som glatter bildet.

4. En terskel som angir nedre grense for areal med tanke på akseptable regionsegmenteringer.

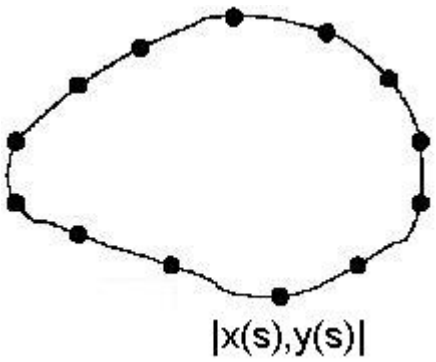
For ytterligere informasjon om denne teorien henvises leseren til kapittel 8, som gir en mer nøyaktig beskrivelse av denne teorien i form av pseudokode.

2.4 Deformerbare konturer

Deformerbare konturer er metoder som kan bake apriori eller høyere nivås kunnskap inn i segmenteringsprosessen. Disse gir sammenhengende, hele og lukkede konturerer, og har evnen til å kompensere for en del støy, hull og andre irregulariteter i bildet [15]. I tillegg er parameterverdiene til segmenteringsresultatene i seg selv egnede til egenskapsrepresentasjon. Avsnitt 2.4.1 og 2.4.2 beskriver to metoder innen denne kategorien, som henholdsvis kalles *snakes* og *level set*.

2.4.1 Snakes

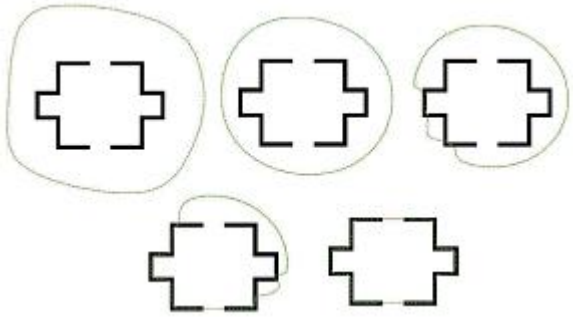
En slange (*snakes*) [15, 16, 17] er essensielt en energiminimaliserende mangekant som iterativt deformerer seg inntil den når en stabil tilstand. Denne tilstanden tilsvarer et lokalt minima, der summen av de interne og eksterne kreftene utjevner hverandre. Disse kreftene virker på slangen, som er bygget opp med noder og kanter mellom disse. Nodene representeres parametrisk ($v(s) = (x(s), y(s))$, der $s \in [0, 1]$ gir posisjonen i bildet). Figur 2.6 (kopiert fra *http://www.idi.ntnu.no/emner/tdt17/lectures/lecture2.pdf*) illustrerer et eksempel på en parameterist kontur. De interne kreftene forsøker å endre slangens lengde og krumming, og er slangens egne krefter. De eksterne kreftene er krefter fra bildet og eventuellt andre beskrankningskrefter, som kan betraktes som fjærkrefter og liknende. Disse skyver og drar i slangen. Bildekrefter står gjerne i forhold til kanter i bildet, da disse ofte samsvarer godt med de konturene som interesserer oss. Når de interne og eksterne kreftene havner i likevekt, slutter slangen å deformere seg og representerer da segmenteringsresultatet.



Figur 2.6: Parameterisert kontur

En ulempe med slanger er at disse lett havner i andre lokale minima enn de vi søker. De blir derfor avhengig av gode initialiseringer tett opptil de segmenteringsresultatene vi ønsker i bildet. I tillegg må parameterverdiene settes slik at den interne energien ikke hindrer slangen i å nå frem til og følge de konturene som vi ønsker. Andre problemer kan være å sette bildekreftene på en slik måte at disse har en stor fangvidde og evnen til å drive slangen ned i konkave konturer. Et mottiltak mot problemene nevnt i forrige setning kan være bruk av *Gradient vector flow* [17], som sprer kraftfeltet til bildekreftene på en iterativ måte. En annen svakhet er at slangens oppøsning (tetthet og antall noder) må stå i forhold til objektet av interesse i bildet. Figur 2.7 (kopiert fra *http://www.idi.ntnu.no/emner/tdt17/lectures/lecture1.pdf*) viser et eksempel på hvordan en slangedeformasjon kan foregå.

Energien til en slange kan beskrives summen av interne og eksterne



Figur 2.7: En itererende slange

krefter:

$$E = \int_0^1 E_{int}(v(s)) + E_{im}(v(s)) + E_{con}(v(s))\ ds \tag{2.3}$$

der

$E_{int}(v(s))$ er slangens interne energi
$E_{im}(v(s))$ er energien til bidekreftene
$E_{con}(v(s))$ er energien til beskrankningskrefter

De interne kreftene modelleres som:

$$E_{int} = \alpha|v'(s)|^2 + \beta|v''(s)|^2 \tag{2.4}$$

der

α bestemmer strekkkevnen til slangen
β bestemmer bøyeevnen/stivheten til slangen
$v'(s)$ er derivert av v mhp parameter s
$v''(s)$ er dobbelderivert av v mhp parameter s

Bildekreftene kan for eksempel modelleres som:

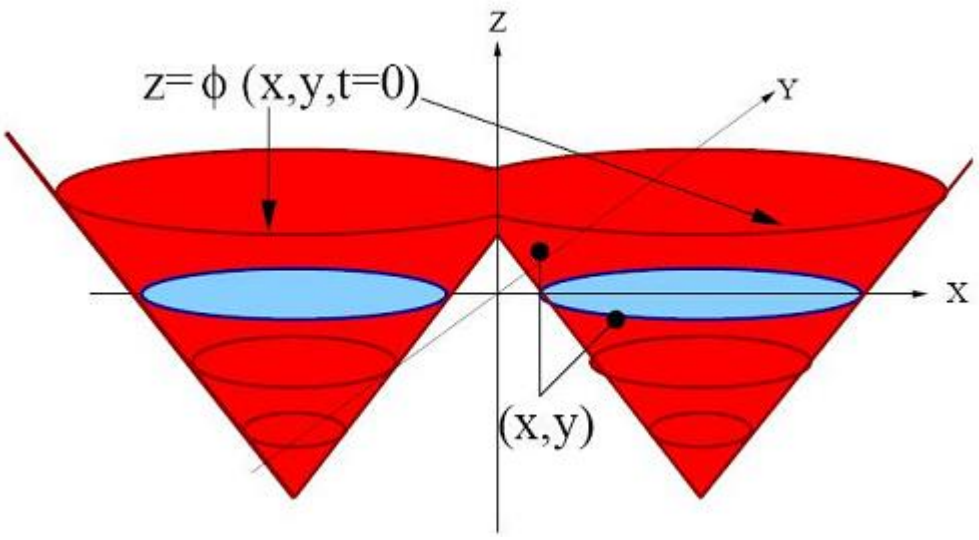
$$E_{ext}(x,y) = -|\nabla G_\sigma(x,y) * I(x,y))|^2 \tag{2.5}$$

der

∇G_σ er gradienten til en gaussisk glatting av bildet med standardavviket σ
$I(x,y)$ er intensiteten i bildet
Stjernen * betegner konvolusjon

2.4.2 Level set

Level set metoden [18] tar i bruk en tidsvariabel og en ekstra dimensjon for å segmentere bildet. For et 2-dimensjonalt bilde konstrueres en overflatefunksjon (*level set funksjon*) $z = \phi(x, y, t)$, som tar inn bildekoordinatene (x, y) og tiden t som parametere. Overflatefunksjonen representerer et distansemål/høydemål fra punktene i bildet til konturen ved tiden t. Denne kan initialiseres som positive, null og negative høyder. En måte å gøre initialiseringen på er å bruke signert euklidsk distanse mellom pikselposisjoner til den valgfrie initialkonturen i bildet, der positivt og negativt fortegn velges for piksler heholdsvis utenfor og innenfor konturen. Som følge av initialiseringen med positive og negative høyder vil overflatefunksjonen ϕ skjære med xy-planet ved tiden $t = 0$. Alle punktene (x, y) som innfrir $\phi(x, y, t = 0) = 0$ utgjør da *zero level set*, som da er initialkonturen i bildet. Figur 2.8 (kopiert fra [18]) illustrerer overflatefunksjonen til et bilde med en bildesentrert sirkel som initialkontur.



Figur 2.8: Level set funksjon

Level set metoden sørger for at høydeendringer i overflatefunksjonen $\phi(x, y, t)$ blir justert på en slik måte at disse samsvarer med konturens bevegelser i bildet. Den deformerbare konturen beveger seg med en fart som avhenger av globale, lokale og uavhengige egenskaper. Disse kan for eksempel tilsvare konturens krumming og fassong, og kanter i bildet. For å opprettholde korrespondansen mellom overflatefunksjonen og konturbevegelsen, kreves et arbeid med å oppdatere overflatefunksjonen i form av å la den stige, synke og utvide seg. For å få tak i konturen ved tiden t er det bare å trekke ut (x, y)-koordinatene som tilfredstiller $\phi(x(t), y(t), t) = 0$. Ved å tidsderivere

denne likningen, anvende kjerneregelen og substituere inn fartsfunksjonen F får man *Level set likningen*, som bestemmer hvordan overflatefunksjonen og konturen endrer seg:

$$\phi_t + F(\phi_x^2 + \phi_y^2)^{1/2} = 0 \tag{2.6}$$

der

ϕ_t er partisiell tidsderivert av ϕ
F er fartsfunksjonen
ϕ_x er partisiell derivert av ϕ med hensyn på x
ϕ_y er partisiell derivert av ϕ med hensyn på y

Level set metoden unngår elegant topologiske problemer som *snakes* må ta høyde for, som for eksempel hva som skjer i tilfeller av en overlappende slange og lignende. En annen fordel med *level set* er at den gjelder like godt for segmenteringer i høyere dimensjoner, som for eksempel 3 og 4-dimensjonale bilder osv.

2.5 Andre segmenteringsmetoder

Det finnes mange andre fremgangsmåter og kombinasjoner av teknikker ut over de segmenteringsmetodene som er nevnt hittil i kapittelet. Blant annet kan vi nevne *matematisk morfologi* [5, 6, 13], *overflate og teksturbasert segmentering* [3, 2], *mønstergjenkjenning* [12, 8], *wawelets* [6] og *klusterbasert segmentering* [2]. Ytterligere kan vi nevne diverse metoder innen kunstig intelligens som *fuzzy logikk*, *nevrale nettverk* og *genetiske algoritmer* [5, 6, 3], og statistiske metoder [5, 3] som for eksempel *Markow Random Field* og *Bayesian*.

Kapittel 3

Egenskapsbeskrivelse

Det er svært viktig å kunne beskrive egenskapene til de objektene som kommer ut av segmenteringsprosessen på en god måte. I litteraturen går disse metodene gjerne under navnet "*feature/shape description and/or representation*" [5, 6, 19, 20]. Det overordnede målet er at disse effektivt, unikt og mest mulig transformasjonsinvariant skal kunne representere viktige karakteristikker av et objekt, for eksempel i form av en vektor. På denne måten "forenkles" og "komprimeres" bildeinnholdet, og følgelig utgjør disse metodene en viktig del av automatisert bildebehandling, der man i en senere fase gjerne ønsker å sammenlikne resultatene opp mot en modell, eller andre segmentering- og egenskapsresultater.

En ideell egenskapsbeskrivelse skal prestere godt sett fra flere perspektiv, og for eksempel kan følgende kriterier nevnes [19]:

- God nøyaktighet ved gjenskapning av objekter fra egenskapsbeskrivelsen
- Kompakte egenskaper
- Generelt anvendelig, og ikke bare gjøre det godt for spesielle type objekter/fassonger
- Lav beregningsmessig kompleksitet
- Robust prestasjon

Det vil ofte være en avveining mellom nøyaktighet og effektivitet i egenskapsbeskrivninger [19]. Dette fordi effektive egenskapsbeskrivninger bør være så kompakte som mulig for å forenkle indeksering og gjenskapning, mens nøyaktige beskrivelser må ta med mer informasjon for å beskrive objektene nøyaktig.

Overordnet kan egenskapsbeskrivelse deles inn i to hovedgrupper; konturbasert og regionbasert [5, 6, 19]. Førstnevnte betrakter kun konturen til objektene, mens sistnevnte betrakter objektenes innhold i tillegg til deres kontur.

Ulemper med konturbasert beskrivelse er at innholdet i objektene ikke tas med i betraktningen, og følgelig er disse langt mer følsomme for støy enn regionbasert beskrivelse. I tillegg kan det være problematisk å ha sammenhengende kontur tilgjengelig fra segmenteringsprosessen, og i visse bruksområder kan objektenes innhold være viktigere enn deres kontur. Dog må det sies at det finnes andre bruksområder, der det bare er objektenes kontur som er viktig, og i slike utgjør naturligvis konturbasert beskrivelse den beste fremgangsmåten. [19]

Regionbasert beskrivelse overvinner mange av svakhetene til konturbasert beskrivelse. Disse er mer robust siden de bruker all objektinformasjon, de er generelt anvendelige, og produserer også generelt mer nøyaktige gjenskapelser. I tillegg kan disse hanskes med defekte konturer. Det er også av betydning at disse ikke nødvendigvis er mer komplekse enn konturbaserte beskrivelser. [19]

Dette kapittelet presenterer bare et lite utvalg av mulige egenskapsbeskrivelser og representasjoner. Kapittelet er delt opp i to avsnitt, der avsnitt 3.1 beskriver konturbaserte metoder, og avsnitt 3.2 beskriver regionbaserte metoder.

3.1 Konturbasert beskrivelse og representasjon

Dette avsnittet inneholder sju delavsnitt, der ulike former for konturbasert beskrivelse og representassjon presenteres.

3.1.1 Enkle beskrivelser

Perimeter

Perimeter (*boundary length*) representerer lengden til objektets kontur [5, 6, 21]. Ved 8-konnektivitet kan for eksempel vertikale og horisontale steg tilsvare enhetssteg, mens diagonale steg kan tilsvare enhetssteg multiplisert med $\sqrt{2}$. Perimetermålet er sterkt avhengig av oppløsningen i bildet.

Omsluttende boks

En omsluttende boks (*boundig box*, *basic retangle*) kan beskrives som et rektangel med minimalt areal som omslutter hele objektet, som finnes ved å

rotere rektangelet i små steg inntil minimalt rektangel detekteres [5, 6]. En forenklet variant kan være å se bort fra rotasjon av rektanglet, og kun beskrive de to pikselposisjonene som markerer objektets ytterpunkter i nord, vest, sør og øst.

Krumming

Krumming (*curvarure*) måler hvor mye konturens tangent forandrer retning [5, 6]. En måte å approksimere krumming i diskrete tilfeller på er å bruke differansen mellom retningen av kantsegmenter, som ligger i nærheten av hverandre.

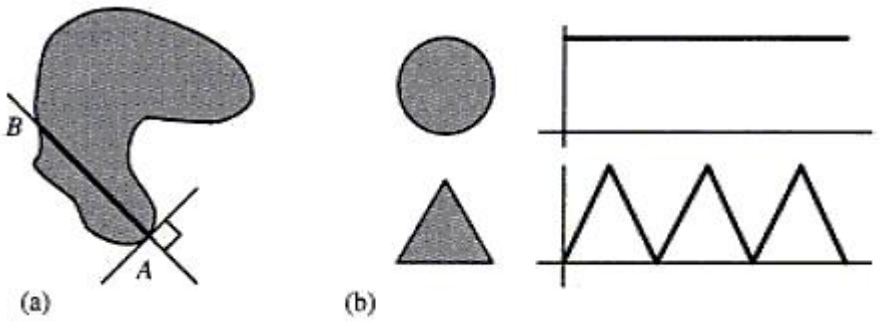
Bøyningsenergi

Bøyningsenergi (*bending energy*) kan forklares som den energien som er nødvendig for å bøye en stang til ønsket fassong, og kan beregnes som en sum av kvadrater av konturkrumming *c(k)* over konturlengden L [5, 19, 20].

$$BE = \frac{1}{L}\sum_{k=1}^{L} c^2(k) \tag{3.1}$$

Signatur

Signatur (*signature*) er en 1-dimensjonal representasjon av konturen til objektet. [5, 6, 19, 20, 21]. Dette kan for eksempel gjøres ved å plotte distansen fra senterpunktet til konturen som en funksjon av vinkelen (polar form). En annen vri er å plotte lengden fra den ene til den andre siden av konturen, når lengden står normalt på tangenten til konturen. Denne illustreres med figur 3.1 (kopiert fra [5]). Ytterligere en variant er å plotte tangenten/stigningstallet som en funksjon av konturen (*slope density function*).



Figur 3.1: Signatur

Kordefordeling

Kordefordeling (*chord distribution*) er en sum av antall tilfeller der en linje med en bestemt vinkel og lengde treffer konturen i begge sine endepunkter [5]. Dette gjøres over hele konturen, og kan matematisk beskrives på følgende måte:

$$h(\Delta x, \Delta y) = \int\int b(x,y)b(x+\Delta x, y+\Delta y)\,dx\,dy \tag{3.2}$$

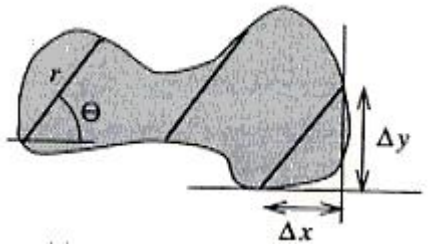
der

$$\text{linjens lengde } r = \sqrt{\Delta x^2 + \Delta y^2}$$

$$\text{linjens vinkel } \theta = \arctan\left(\frac{\Delta y}{\Delta x}\right)$$

$$b(x,y) = \begin{cases} 1 & \text{hvis punktet (x,y) ligger på konturen} \\ 0 & \text{ellers} \end{cases}$$

Dette illustreres i figur 3.2 (kopiert fra [5]).



Figur 3.2: Kordefordeling

Temperatur

Basert på en termodynamisk formalisme kan konturens temparatur defineres som: [21]

$$T = \frac{1}{\left(\lg\left(\frac{2P}{P-H}\right)\right)} \tag{3.3}$$

der

P er konturens perimeter

H er perimeteren til konturens konvekse hull [5]

Diameter

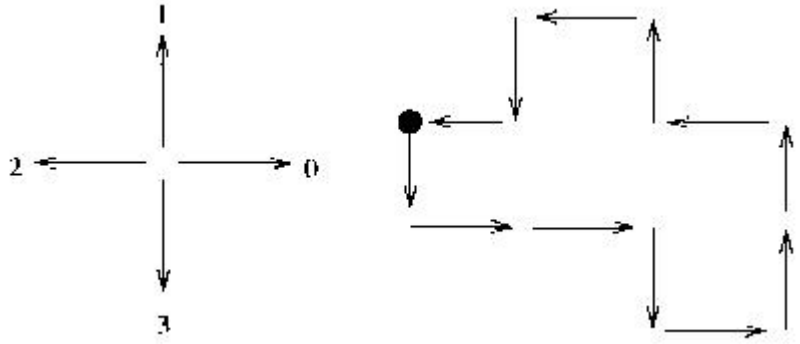
Diameter kan defineres som den største distansen mellom to punkter på konturen [21].

3.1.2 Parametrisk approksimasjon

Egenskapsbeskrivelse kan også baseres på parameterisering av nodepunkter langs konturen. Antall noder tilsvarer da grad av nøyaktighet i egenskapsbeskrivelsen. For eksempel kan deformerbare konturer som *snakes* og *level set* benytte seg av dette til å beskrive objektene direkte ved segmenteringsresultatet.

3.1.3 Kjedekode

Kjedekode (chain code) beskriver en kontur som en sekvens av koder [5, 6, 19, 20]. Kodene velges typisk på bakgrunn av enten 4 eller 8-konnektivitet. Ved 4-konnektivitet brukes gjerne koder som 0 for øst, 1 for nord, 2 for vest og 3 for øst, og ved 8-konnektivitet velges koder tilsvarende. Kodesekvensen blir så angitt av kodene når man følger konturen med eller mot urviser. Figur 3.3 illustrerer kjedekoden 3,0,0,3,0,1,1,2,1,2,3,2 med 4-konnektivitet.



Figur 3.3: Kjedekode

3.1.4 Spektraltransformasjon

Fourier descriptors [5, 6, 19, 20, 21] er en populær metode for egenskapsbeskrivelse. Fra det romlige domenet omskrives konturens parametriske $(x(k), y(k))$-koordinater til komplekse tall: $(s(k) = x(k) + jy(k)$, for $k = 0, 1, 2, ..., K - 1$. Med andre ord behandles x-aksen som den reelle aksen, og y-aksen som den imaginære aksen. Overgangen til komplekse tall medfører en reduksjon fra to dimensjoner til en dimensjon. Den diskret *Fourier transformerte* av $s(k)$ er gitt som:

$$a(u) = \frac{1}{K} \sum_{k=0}^{K-1} s(k) e^{-j2\pi uk/K}, \ for \ u = 0, 1, 2, ..., K-1 \qquad (3.4)$$

der

$a(u)$ er komplekse koeffisienter til konturen (*Fourier descriptors*)

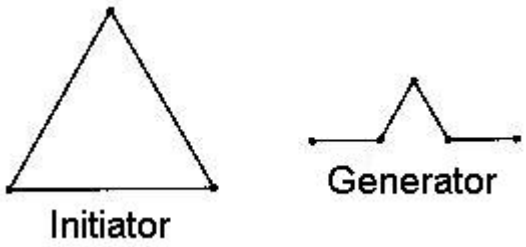
K er antall piksler som ligger på konturen

Ved å bare bruke de P første koeffisientene oppnår man en egenskapsbeskrivelse som bare inkluderer de P laveste frekvensene. Siden lave frekvenser tilsvarer global fassong i bildet, og høye frekvenser tilsvarer detaljer i konturen, vil valget av P avgjøre hvor mye detalj man ønsker i konturen.

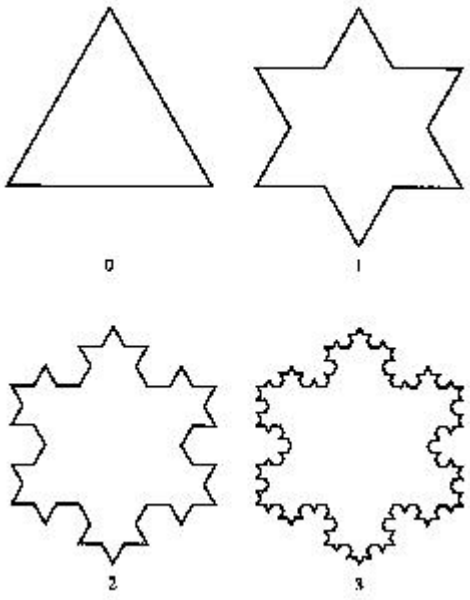
Fourier descriptors har flere fordeler. De er enkle å beregne, normalisere og implementere, og har en spesifikk fysisk mening. Metoden overvinner også problemet ovenfor følsomhet for støy og konturvariasjoner på grunn av at analysen foregår i spektraldomenet. En ulempe er at metoden ikke gir lokal informasjon om konturen.

3.1.5 Fraktaler

Fraktaler kan være nyttige egenskapsbeskrivelser i tilfeller av komplekse objekter med selvlignende (*self-similar*) strukturer over forskjellige skalaer [21]. Kjernen i disse metodene er å definere en initiator og en generator [22]. Initiatoren er en startstruktur som man i første iterasjon anvender generatoren på. I påfølgende iterasjoner anvendes generatoren på resultatet fra forrige iterasjon. Generatoren virker på en slik måte at den skifter ut deler av stukturen/objektet med skalerte kopier av seg selv. I egenskapsbeskrivelse søker man da etter enkle strukturrepresentasjoner av initiator og generator, som kan produsere stukturer som likner på det aktuelle objektet. Figur 3.4 (kopiert fra [22]) illustrer en initiator og generator, og figur 3.5 (kopiert fra [22]) viser resultatet etter de første tre iterasjonene.



Figur 3.4: Initiator og generator



Figur 3.5: Resultat etter de tre første iterasjoner

3.1.6 Syntaktisk analyse

Syntaktisk analyse [19, 20] som *relational descriptors* [6] uttrykker relasjoner mellom enkle primitiver. Dette kan for eksempel skje i form av gramatikk med evne til å beskrive strenger og produksjonsregler, som i sin tur bygger opp konturer fra en liten mengde enkle primitiver.

3.1.7 Diverse egenskapsbeskrivelser

Blant andre konturbaserte egenskapsbeskrivelser kan vi for eksempel nevne *Hough transform* [5, 6], *kurvebasert og polygonal approksimasjon* [5, 6, 19, 20, 21], *correspondence-based shape matching* [19] og *stochastic* og *scale space method* [19, 20].

3.2 Regionbasert beskrivelse og representasjon

Dette avsnittet inneholder ni delavsnitt, der ulike former for regionbasert beskrivelse og representassjon presenteres.

3.2.1 Enkle beskrivelser

Areal

Arealet til et objekt utgjør en kompakt egenskapsbeskrivelse [5, 6, 19, 21].

Massesenterpunkt

Massesenterpunktet (*centroid*) er en egenskapsbeskrivelse som kanskje enklest kan estimeres ved å ta gjennomsnittsverdiene til objektets pikselkoordinater [21]. Blant andre egenskapsbeskrivere som gjør bruk av dette punktet, kan maksimal, gjennomsnittlig og minimal distanse til massesenterpunktet fra konturen nevnes. Figur 3.6 (kopiert fra [21]) illustrerer massesenterpunktet for to objekter.

Figur 3.6: Massesenterpunkt for to objekter

Kompakthet

Kompakthet (*compactness*, *circularity*) er et mål på objektets "rundhet", og kan beskrives på følgende måte: [5, 6, 19]

$$C = \frac{P^2}{A} \tag{3.5}$$

der

P er objektkonturens perimeter
A er objektets areal

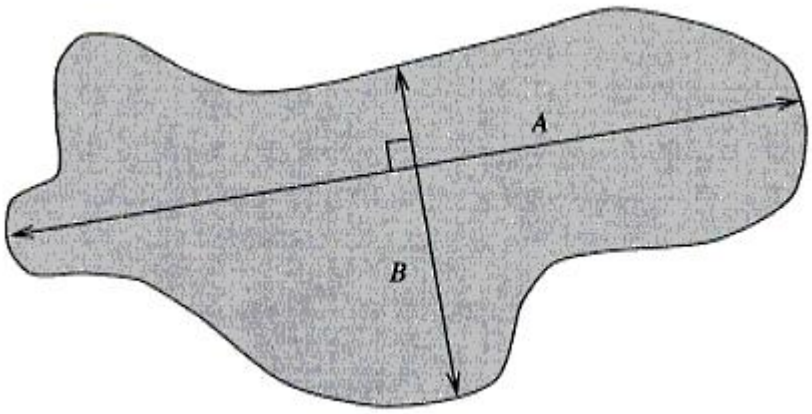
Enkle statistiske egenskaper

Gjennomsnittet, median og standardavvik av pikselverdiene til et objekt utgjør alle en kompakt egenskapsbeskrivelse [6].

Aksebasert beskrivelse

Store og lille akse (*major and minor axis*) er egenskapsbeskrivelser, der store akse representerer en akse langs den delen av objektet som er mest langstrakt,

og lille akse står normalt på store akse [21]. Disse kan brukes som komponenter i andre egenskapsbeskrivere. Et eksempel på dette er eksentrisitet (*eccentricity*), som kan formuleres som forholdet mellom store og lille akse [5, 6, 19]. Figur 3.7 (kopiert fra [5]) gir en illustrasjon på dette.



Figur 3.7: Eksentrisitet

Rektangularitet

Rektangularitet kan betraktes som et mål på hvor godt konturen kan approksimeres med et omsluttende rektangel (bounding box) [5, 21]. En definisjon på rektangulariteten R kan beskrives på følgende måte:

$$R = \max_{k}(\frac{A_o}{A_r}) \tag{3.6}$$

der

k er retningen til rektangelet

A_o er objektets areal

A_r er arealet av rektangelet som omslutter objektet

Langstrakthet

Langstrakthet (*elongadeness*) er forholdet mellom lengden og bredden til det omsluttende rektangelet (bounding box) [5, 21]. Langstrakthet kan også evalueres etter forholdet mellom objektets areal og kvadratet av objektets tjukkelse, der sistnevnte for eksempel kan bestemmes ved bruk av erosjonssteg i *matematisk morfologi* [5]:

$$L = \frac{A}{(2d)^2} \tag{3.7}$$

der

A er objektets areal

d er antall erosjonssteg

Projeksjon

Projeksjon beskriver egenskaper hos en region med hensyn på retninger, og kan eksempelvis for horisontale og vertikale retninger defineres på følgende måte [5]:

$$p_h(i) = \sum_j f(i,j) \tag{3.8}$$

$$p_v(j) = \sum_i f(i,j) \tag{3.9}$$

Figur 3.8 (kopiert fra [5]) illustrerer et eksempel på horisontale og vertikale projeksjoner av en region.

Eulertall

Eulertallet (*Euler number*) beskriver en topologisk egenskap på følgende måte: [5, 6, 21]

$$E = C - H \tag{3.10}$$

der

C er antall sammenhengende komponenter i objektet

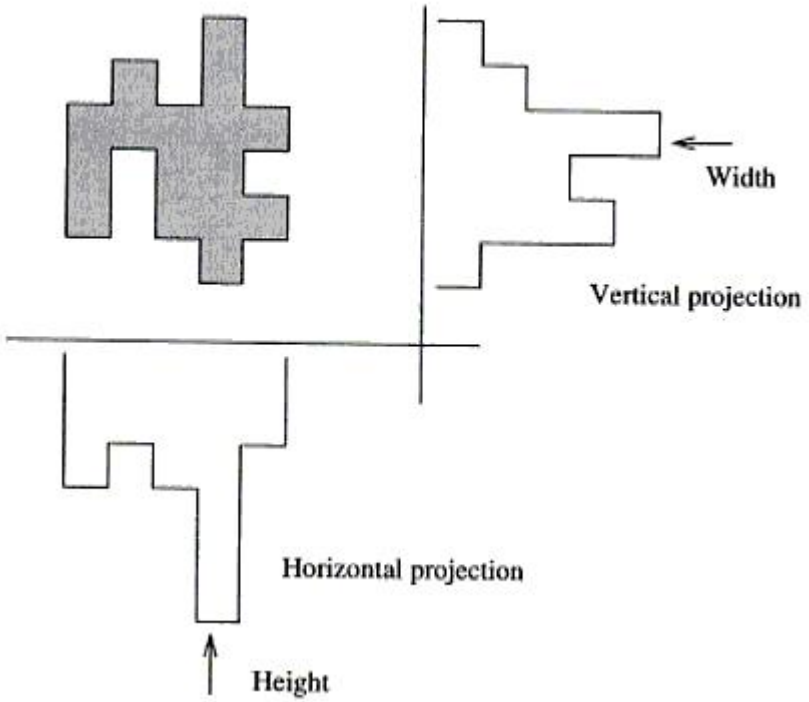
H er antall hull i objektet

Figur 3.9 (kopiert fra [6]) illustrerer et eksempel med henholdsvis 0 og -1 som Eulertall.

Gjennomsnittsdistanse til kontur

Gjennomsnittsdistansen fra regionens interne punkter til konturen (*mean distance to boundary*) kan brukes som en egenskapsbeskrivelse [21]:

$$\beta = \frac{1}{N} \sum d(r, boundary(g)) \tag{3.11}$$



Figur 3.8: Horisontal og vertikal projeksjon av en region

der

g er objektet av interesse

N er antall piksler i g

$r \in g$ er et godt punkt i g

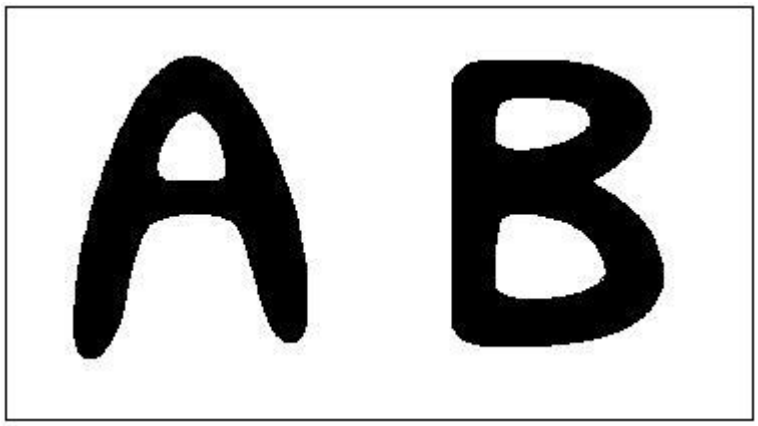
$d(r, boundary(g))$ er den minste distansen mellom r og alle de andre konturpikslene

3.2.2 Statistiske momenter

Statistiske momenter beskriver objekter ved hjelp av enkle statistiske karakteristikker (momenter) [5, 6, 19, 20, 21]. Fordeler med disse fremfor mange andre metoder er deres enkle implementasjon, invarians, og deres fysiske tolkning av lavere ordens momenter opp mot objektene [6, 20].

Den todimensjonale definisjonen av et moment med orden $p+q$ er gitt som:

$$m_{pg} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q f(x, y)\, dx\, dy \tag{3.12}$$



Figur 3.9: Eulertall, henholdsvis 0 og -1

der

$f(x, y)$ beskriver objektet av interesse

Translasjonsinvarians bygger på sentralmomentet:

$$\mu_{pg} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x - x_c)^p (y - y_c)^q f(x, y)\, dx\, dy \tag{3.13}$$

der

$f(x, y)$ beskriver objektet av interesse

$x_c = \frac{m_{10}}{m_{00}}$ som tilsvarer x-koordinat til massesenterpunktet (centroid)

$y_c = \frac{m_{01}}{m_{00}}$ som tilsvarer y-koordinat til massesenterpunktet (centroid)

Videre kan man normalisere sentralmomentene for å oppnå skaleringsinvarians:

$$\eta_{pg} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}} \tag{3.14}$$

der

$$\gamma = \frac{p + q}{2} + 1 \qquad \text{for } p + q = 2, 3, ...$$

Rotasjonsinvarians oppnås ved å velge koordinatsystem slik at

$$\mu_{11} = 0 \tag{3.15}$$

Flere invariante momenter ovenfor translasjon, rotasjon og skalering kan så bli utledet fra andre og tredje moment (normalisert):

$$\phi_1 = \eta_{20} + \eta_{02} \tag{3.16}$$

$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \tag{3.17}$$

$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \tag{3.18}$$

$$\phi_4 = (\eta_{30} - \eta_{12})^2 + (\eta_{21} - \eta_{03})^2 \tag{3.19}$$

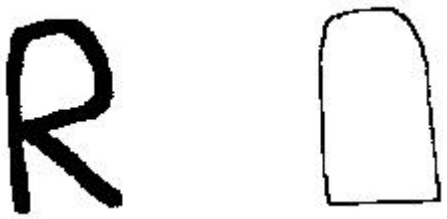
$$\vdots$$

3.2.3 Tekstur

Tekstur [6, 19, 21] kan brukes som et middel til å definere egenskaper hos objekter i et bilde. Disse analyserer pikselverdier relatert til andre pikselverdier i nabolaget. Ved bruk av forskjellige statistiske mål kan egenskaper som glatthet, grovhet, regelmessighet, kontrast, retning, likhet, kornethet, periodiskhet og lignende beregnes. *Co-occurence* matrisen er en populær teksturrepresentasjon [6, 19, 21]. Denne baseres på pikselverdier samt orientering og distanser mellom piksler. Meningsfull statistikk kan trekkes ut fra denne og representere teksturinformasjonen. *Fourier*, *Gabor* og *wavelet* transformasjoner er andre egnede teknikker for teksturbeskrivelse [6, 21].

3.2.4 Konvekse hull

Det konvekse hullet av et objekt er den minste konvekse regionen som omslutter objektet [5, 19]. Figur 3.10 (kopiert fra [5]) viser et eksempel.



Figur 3.10: Konvekse hull

3.2.5 Relasjonsbaserte beskrivelser

Blant metoder som uttrykker relasjoner mellom egenskaper kan *region adjacency graph* og [5] og *shape decomposition* [20] nevnes. *Region adjacency graph* beskriver topologiske relasjoner mellom objekter i et bilde. Dette gjøres ofte i form av en graf, der nodene og kantene henholdsvis tilsvarer objektene og de topologiske relasjonene mellom dem. *Shape decomposition* teknikker representerer en region/kontur som en kombinasjon av komponenter av andre regioner/konturer. Dette gjør det mulig å representere komplekse objekter med bare bruk av enkle primitiver.

3.2.6 Hovedkomponent analyse

Hovedkomponent analyse (*principal component analysis*) kan brukes til å beregne en forenklet beskrive av eksisterende kontur- og regioninformasjon i et bilde [6]. Metoden har evnen til å redusere mengden egenskapsinformasjon/data samtidig som de viktigste egenskapene/dataene beholdes. Denne er generelt anvendelig for n-dimensjonale vektorer, men her gis spesifikk beskrivelse for to dimensjoner, da dette er tilstrekkelig for kontur- og regionrepresentasjon av 2-dimensjonale segmenteringsresultater.

Utgangspunktet er å representere den aktuelle regionen som en populasjon av 2-dimensjonale vektorer:

$$\mathbf{v} = (x, y)^T \tag{3.20}$$

der

(x, y) er pikselkoordinatene til regionen i bildet
T betegner transponert vektor

Populasjonens gjennomsnittsvektor (centroide) beregnes som [6]:

$$\vec{m} = \frac{1}{K} \sum_{k=1}^{K} \vec{v_k} \tag{3.21}$$

der

$\vec{v_k}$ er vektor nummer k i populasjonen $\mathbf{v}$
K er antall vektorer i populasjonen $\mathbf{v}$ (antall piksler i regionen)

Kovariansmatrisen identifiserer likheter og forskjeller mellom vektorene i populasjonen, og kan approksimeres som [6]:

$$\mathbf{C_x} = \frac{1}{K}\sum_{k=1}^{K} \vec{v_k}\vec{v_k}^T - \vec{m}\vec{m}^T \tag{3.22}$$

Siden vi bruker 2-dimensjonale vektorer gjelder 2 x 2 dimensjonal kovariansmatrise, som generelt kan skrives som:

$$\mathbf{C_x} = \begin{bmatrix} cov(Dim_x, Dim_x) & cov(Dim_x, Dim_y) \\ cov(Dim_y, Dim_x) & cov(Dim_y, Dim_y) \end{bmatrix} \tag{3.23}$$

der

Dim_x er x-dimensjonen av regionen
Dim_y er y-dimensjonen av regionen
$cov(Dim_x, Dim_y) = cov(Dim_y, Dim_x)$
$cov(Dim_x, Dim_x) = var(Dim_x)$
$cov(Dim_y, Dim_y) = var(Dim_y)$
cov betegner kovariansen mellom to dimensjoner
var betegner variansen av en dimensjon

Med tanke på egenskapsbeskrivelse søker vi eigenvektorer og eigenverdier. Disse beregnes ut av kovariansmatrisen, og eksisterer bare dersom følgende likning oppfylles [23]:

$$\mathbf{C_x}\vec{e} = \lambda\vec{e} \tag{3.24}$$

der

$\vec{e}$ er eigenvektor
λ er eigenverdi

Eigenvektorene og eigenverdiene kan finnes ved en to stegs prosedyre [23]. Først løses den karakteristiske likningen med hensyn på eigenverdiene lambda:

$$|\mathbf{C_x} - \lambda I| = \vec{0} \tag{3.25}$$

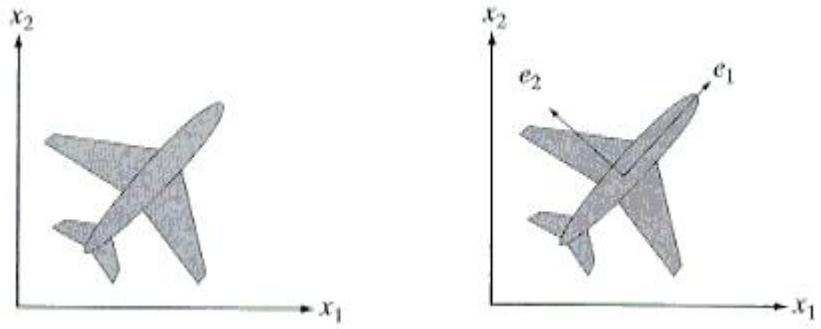
der

I er identitetsmatrisen

Deretter beregnes to aktuelle eigenvektorene assosiert med hver sin eigenvektor ved følgende likning:

$$(\mathbf{C_x} - \lambda I)\ \vec{e} = \vec{0} \tag{3.26}$$

Første hovedkomponent (*first principal*) er definert som den av de to eigenvektorene med høyest eigenverdi. Denne hovedkomponenten er et mål på den største variansen mellom dimensjonene i regionen. Den andre eigenvektoren kalles andre hovedkomponent (*second principal*), og gir et mål på den nest største variasjonen mellom dimensjonene i regionen. Denne er uavhengig av første hovedkomponent, og vil grafisk stå normalt på den første hovedkomponeneten. Sammen vil dermed disse to hovedkomponentene utgjøre et nytt koordinatsystem for regionen. Denne beskrivelsen blir derfor rotasjonsinvariant. Figur 3.11 (kopiert fra [6]) gir et eksempel.

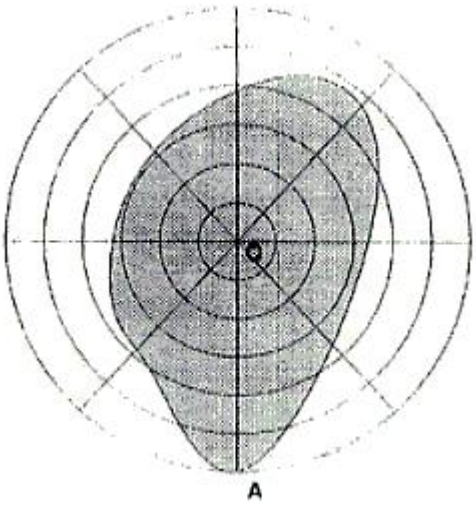


Figur 3.11: Hovedkomponent analyse

3.2.7 Matriserepresentasjoner

Rutenett metoden (*grid based method*) baserer seg på å legge et rutenett over det aktuelle objektet, og så tildele rutene binære verdier ved å betrakte innholdet i hver rute [19]. For eksempel kan man tildele verdien 1 til ruter som dekkes av objektet, og verdien 0 til ruter som ikke dekkes helt av objektet. Rutestørrelsene bør naturligvis være betydelig større enn enkeltpikseler, og hvor mye er avhengig av hvor grov beskrivelse man ønsker.

En annen liknende egenskapsbeskrivelse oppnår rotasjonsinvarians i motsetning til rutenett metoden. Denne er mer avansert ved at den anvender sirkulær rastersampling. Metoden går under navnet *shape matrix* [19, 20]. I stedet for rutenett brukes en polar raster av konsentriske sirkler og radielle linjer, der massesenterpunktet til objektet utgjør senter for sirklene og skjæringer for linjene. Binærverdiene trekkes (samples) i skjæringspunktene mellom sirklene og de radielle linjene. Matrisen konstrueres ved at sirklene korrespinderer til matrisekolonner, og radielle linjer korresponderer til matriserader. Figur 3.12 (kopiert fra [20]) illustrerer et eksempel.



Figur 3.12: Matriserepresentasjon

3.2.8 Morfologiske metoder

Morfologiske metoder er nyttige i egenskapsbeskrivelser på grunn av deres direkte relavans til objektkonturer [20]. Det finnes mange eksempler på morfologiske egenskapsbeskrivelser, deriblant tykkelse og skjelett/midtakse.

Tykkelse (*thickness*) kan defineres som det antallet ganger en erosjonsoperator må anvendes for å komplett erodere objektet når det strukturerende elementet er det samme ved hver utførelse [21].

Skjelettet (*skeleton*) av en region kan defineres som en midtakse (*medial axis*) i regionen [5, 6]. Midtaksen har en intuitiv definisjon kalt "prairie fire concept". Forutsatt at man tenner fyr på hele konturen av en et objekt samtidig, og at brannfrontene avanserer inn mot regionen i samme hastighet, så vil midtaksen bestå av alle punkter der to av en eller flere fronter møtes samtidig. Morfologiske metoder som tynning er en mulig fremgangsmåte for å finne et objekts skjellet, der man iterativt fjerner kanter av objektet.

3.2.9 Symmetri

Det finnes flere måter å måle et objekts symmetri (*symmetry*) på. En av disse kalles tosidet (bilateral) symmetri, som baserer seg på å reflektere konturen om en akse som går gjennom massesenteret. Neste steg er å fylle eventuelle hull som er forårsaket av refleksjon, som for eksempel kan bli gjort ved hjelp av closing-operatoren i *matematisk morfologi*. Selve egenskapsbeskrivelsen S gis som: [21]

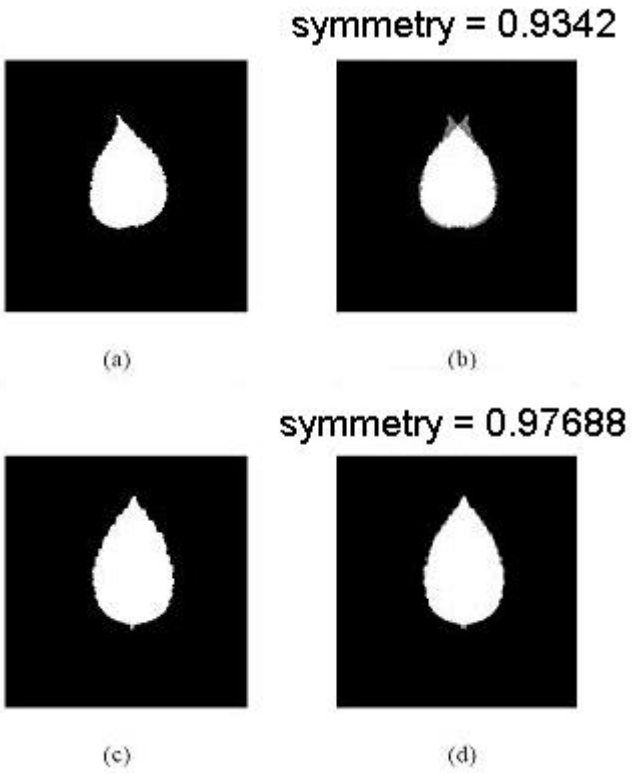
$$S = \frac{N_2}{N} \tag{3.27}$$

der

N_2 er antall symmetriske piksler

N er antallet av både asymmetriske og symmetriske piksler

Figur 3.13 (kopiert fra [21]) illustrerer to eksempler på tosidet symmetri.



Figur 3.13: Symmetri

Kapittel 4

Egenskapsuttrekking

I litteraturen omtales gjerne metoder for å finne frem til en delmengde av en større mengde egenskaper som *feature selection and extraction methods* [24, 25, 26]. Disse er gunstige for å begrense, forenkle og kvalitetssikre mengden av egenskaper som mates inn til den endelige maskinlæringsalgoritmen som klassifiserer dataene. Seleksjon refererer til algoritmer som kun gjør et utvalg av mengden egenskaper, mens ekstrahering refererer til algoritmer som i tillegg kan lage nye egenskaper basert på transformasjoner eller kombinasjoner av det originale settet. Ekstrahering vil derfor kunne gi en bedre separabilitet enn seleksjon, men taper til gjengjeld egenskapenes originale fysiske tolkning, i tillegg til at den vil være beregningsmessig dyrere i delmengdegenereringen [26]. Resten av dette kapittelet omhandler seleksjon.

Målet med seleksjonen er å velge ut den beste delmengden av det originale settet. Denne delmengden kan blant annet defineres som den minste gruppen av egenskaper som er nødvendig og tilstrekkelig for å tjene målet [27, 28]. For at målet skal tjenes må egenskapene oppnå høy nok nøyaktighet i resultatene, som vurderes av en evalueringsfunksjon, eller et eller annet klassifiseringssystem.

Det er beregningsmessig vanskelig å effektivt finne en optimal delmengde av egenskaper, og mange slike problemer er vist NP-harde [29, 30]. Innbakt i dette ligger det at store egenskapsmengder impliserer en høy dimensjon i egenskapsrommet, som i seg selv er vanskelige å hanskes med, og kan føre til beregningsmessige utfordringer i klassifiseringsfasen. Sistnevnte omtales gjerne som *the curse of dimensionality* [26]. En uttømmende søk strategi av hele egenskapsmengden innebærer for eksempel å sjekke 2^N tilstander, der hver av de N egenskapene enten tas med eller ikke.

Et dårlig valg av egenskaper kan gi opphav til flere problemer, som for eksempel ufullstendig informasjon, støyete eller irrelevante egenskaper, og naturlig

nok andre tap av å ikke ha den beste blandingen/delmengden av egenskaper [28]. I slike tilfeller kan kjøretiden bli unødvendig treg på grunn av den høye dimensjonen i egenskapsrommet, og resultatene kan bli unøyaktige på grunn av at innflytelsen til irrelevante data.

Gode egenskapsseleksjoner gir flere fordeler [28, 30, 31]. Disse reduserer dimensjonene i egenskapsrommet, og forbedrer dets separabilitet ved å fjerne redundante, irrelevante eller støyete data. Følgelig reduserers både de beregningsmessige kostnadene, og mengden av data som trengs til trening i tilfelle av maskinlæring. Den økte kvalitet på dataene medfører i seg selv bedre klassifisering.

Som en konsekvens av *the curse of dimensionality* oppstår *the peaking phenomenon* [26]. Fenomenet sier at den beste klassifiseringsnøyaktigheten oppnås et sted mellom bruk av ingen og alle egenskapene som er tilgjenelig [32], og støtter dermed opp under egenskapsuttrekning som et mulig gode. Fenomenet viser seg i prasis ved at at en økende mengde egenskaper ut over et visst nivå reduserer klassifiseringsprestasjonen.

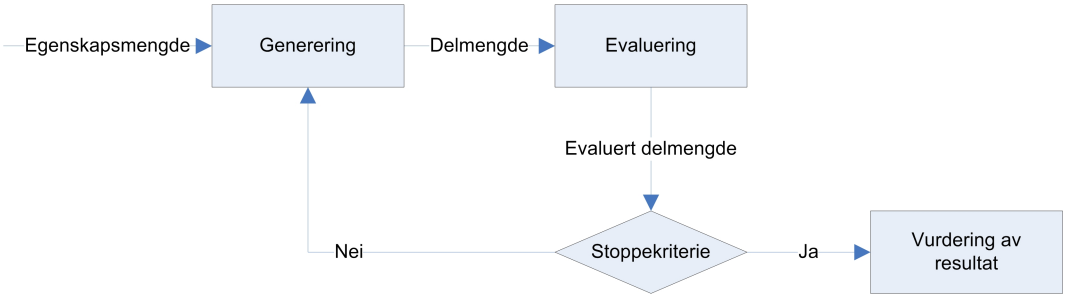
Følgende fire stegs prosedyre [26, 27, 29, 30] er en av mange mulige egenskapsseleksjoner som blir beskrevet i litteraturen:

- Generering av delmengde
- Evaluering av delmengde
- Stoppekriterier
- Vurdering av resultat.

Denne prosedyren kan illustreres med figur 4.1, som er hentet og rekonstruert fra [27, 29, 30]. Kort fortalt genereres og evalueres en ny delmengde inntil visse stoppekriterier tilfredstilles. Resultatvurderingen er en siste fase, der man avgjør om den aktuelle delmengden gir akseptable resultater. De neste avsnittene (4.1, 4.2, 4.3 og 4.4) gir en mer omfattende beskrivelse av stegene i prosedyren.

4.1 Generering av delmengde

Generering av delmengde kan betraktes som et graf søk problem, der grafen symboliserer hele egenskapsrommet. Hver node tilsvarer en egenskap, og stier i grafen representerer delmengder av egenskaper. Søkestrategien inkluderer valg av startpunkt og søkeretning. Startpunktet bestemmer hvilken eller hvilke egenskaper søket starter med, som kan være ingen, noen eller alle egenskapene. Søkeretningen avgjør om antall egenskaper i genereringen skal



Figur 4.1: Seleksjon

vokse, avta, eller være fleksibel i form av begge deler. Organiseringen av søket kan overordnet beskrives som komplett, sekvensielt eller tilfeldig søk. Disse beskrives i henholdsvis avsnitt 4.1.1, 4.1.2 og 4.1.3.

4.1.1 Komplett søk

Et komplett søk garanteter optimal løsning i henhold til den aktuelle evalueringsfunksjonen som blir brukt. Uttømmende søk som *bredde først søk* hører naturligvis hjemme her, men det eksisterer også algoritmer som kan redusere søkerommet uten å gi opp kravet om optimalitet. *Branch and bound* er en slik algoritme [26, 29, 33]. Denne forutsetter monotome evalueringsfunksjoner, hvilket vanligvis ikke er tilfelle [33]. *Branch and bound* unngår uttømmende søk ved å kutte grener i søkerommet (*prunning*). I praksis viser det seg derimot at komplett søk metoder ikke er egnet for store egenskapsrom, da eksekveringstiden er for høy selv i en "off-line"-fase.

4.1.2 Sekvensielt søk

Sekvensielle søkemetoder gir opp kravet om å undersøke alle muligheter til fordel for raskere eksekveringstid. Slike metoder risikerer derfor å miste optimal løsning. Disse følger gjerne grådighetsprinsippet, som velger egenskaper som ser ut til å være best i øyeblikket. Ulemper med slike fremgangsmåter er at den beste mengden av egenskaper ikke nødvendigvis inneholder de egenskapene som etter grådighetsprinsippet ble vurdert til å være best. Med andre ord tas det ikke høyde for avhengigheter mellom egenskapene. I tillegg risikerer disse metodene å fanges i lokale minima. Dog kan disse metodene være nyttige til eksempelvis å begrense veldig store egenskapsmengder i forkant av prosedyrer som er i stand til å betrakte avhengigheter mellom egenskaper.

Sequential Forward Selection (SFS) og *Sequential Backward Selection (SBS)* er metoder som henholdsvis adderer og subraherer en og en egenskap om gangen, på en slik måte at evalueringsfunksjonen maksimaliseres for hvert valg [25, 26, 27]. Disse er beregningsmessig attraktive siden de fjerner en

kandidat fra mulige gjenværende egenskaper i hver iterasjon.

Blant mer sofistikerte metoder kan *Plus l-take away r* [26] og *Sequential Floating Search* [25, 26] nevnes. Førstnevnte legger først til l stk egenskaper, før den så fjerner r stk egenskaper. Sistnevnte er essensielt en generalisering av førstnevnte, der parameterene l og r bestemmes og oppdateres automatisk. Disse metodene medfører tyngre beregningstid enn *SFS* og *SBS*, men gir til gjengjeld bedre resultater på grunn av større fleksibilitet i egenskapsseleksjonen.

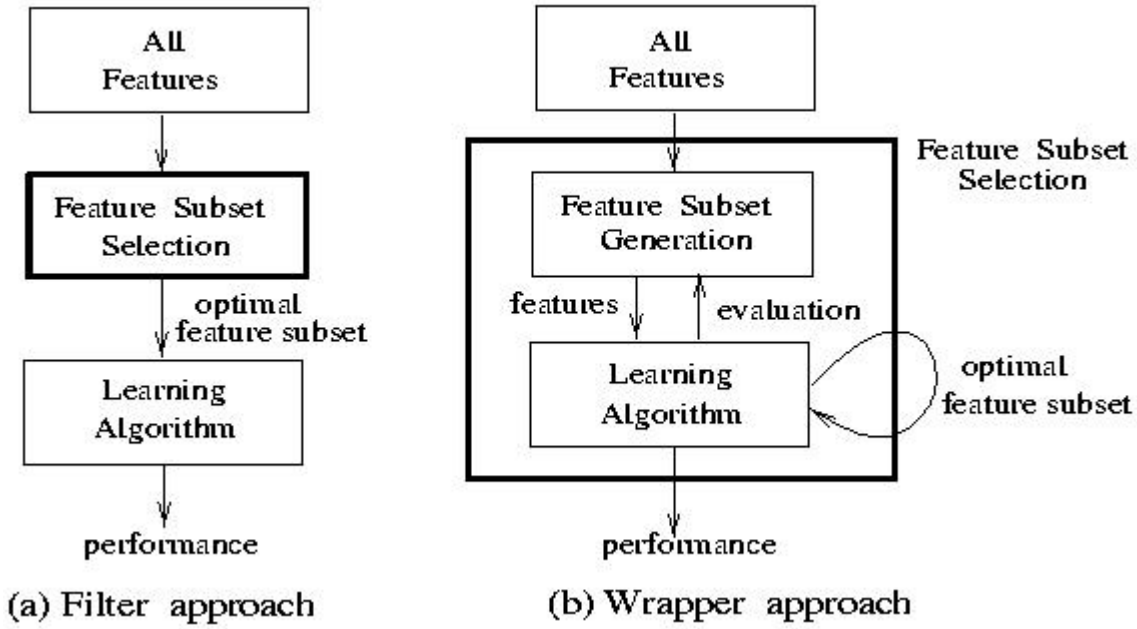
Ut over de få algoritmene som kort er beskrevet her, gis det langt mer informasjon i [25, 26, 27, 30]. Disse artiklene inkluderer både sekvensielle, komplette og tilfeldige strategier for egenskapsseleksjon.

4.1.3 Tilfeldig søk

Egenskapsseleksjon kan også baseres på ikke-deterministiske metoder [26, 27, 29, 30]. Disse tar i bruk tilfeldigheter, og oppnår dermed fordelen med at disse kan unnslippe lokale minimum underveis i søket. Tilfeldighet kan for eksempel integreres i sekvensielle metoder, der man kan starte søket med et tilfeldig utvalg av egenskaper, og/eller kombinere tilfeldighet sammen med grådighetsprinsippet når nye egenskaper velges. Metoder som *simulated annealing*, *genetiske algoritmer* og *random-start hill-climbing* er eksempler på slike [25, 26, 27, 28, 29]. Et annet alternativ er *Las Vegas algoritmer* [26, 29] som genererer de nye delmengdene helt tilfeldig uten å ta hensyn til tidligere valgte egenskaper og kombinasjoner av disse. Disse metodene forlanger parameterverdier, og det er viktig at disse settes riktig for at gode resultater skal oppnnås. I tillegg benyttes gjerne en terskel for maksimalt antall iterasjoner som et stoppekriterium, da flere av disse metodene kontinuerlig kan fortsette og forbedre seg uten å konvergere med en løsning innenfor rimelig tid.

4.2 Evaluering av delmengde

Denne fasen evaluerer de genererte kandidatene, og holder til en enhver tid rede på den beste ved å sammenlikne med beste i forrige runde, og eventuelt oppdatere denne. I litteraturen nevnes spesielt tre modeller for hvordan en slik evaluering kan foregå. Disse kalles *filter* (avsnitt 4.2.1), *wrapper* (avsnitt 4.2.2) og *hybrid* (avsnitt 4.2.3) [26, 27, 29]. Forskjellen i de to førstnevnte ligger i deres bruk av den endelige maskinlæringsalgoritmen. Wrappermetoden anvender denne endelige maskinlæringsalgoritmen til å evaluere potensielle delmengder, mens filtermetoden er helt uavhengig av denne, og genererer delmengder basert på andre evalueringskriterier. Hybrid metoden er en kombinasjon som forsøker å dra nytte av begge de to forannevnte modeller. Filter og wrappermetodene illustreres i figur 4.2, som er kopiert fra [34].



Figur 4.2: Seleksjonsmodeller

4.2.1 Filter

Filtermetoden evaluerer gjerne egenskaper etter mål som konseptuelt kan deles inn i distanse, informasjon, avhengighet og konsistens [26, 27, 29, 35]. Modellen bygger på antakelsen om at egenskaper som gjør det godt på disse målene også har en gunstig effekt på klassifiseringen i den endelige maskinlæringsalgoritmen. Distansemål vektlegger egenskaper som skiller klassene i klassifiseringsfasen mest mulig ved å måle differansen mellom forskjellige klassers sannsynligheter gitt disse egenskapene. Euklidsk distanse er et eksempel. Informasjonsmål velger ut de egenskaper som gir mest informasjon, som for eksempel differansen mellom apriori og aposteriori usikkerhet når aktuelle egenskaper er brukt. Avhengighetsmål måler muligheten til å forutse verdier av en variabel ut fra verdier av en annen. Ved vekt på klassifisering betraktes styrken av assosiasjonen mellom egenskaper og klasser, og de egenskapene med sterkeste assosiasjoner velges. Ved vekt på klustering måles likheten av egenskaper etter deres assosiasjon med hverandre. Disse verdiene gir da et mål på redundans i disse egenskapene, og følgelig velges de med minst assosiasjon. Konsistensmål baseres på klasseinformasjon og sterk vektlegging av minimale delmengder når delmengder skal velges. Disse forsøker å finne et minimalt antall egenskaper som separerer klassene like konsistent som hele den originale mengden av egenskaper kan gjøre. Dette innebærer at de relativt få egenskapene som velges må bestå av svært lav inkonsistens.

Måling av inkonsistens kan tilnærmes på flere måter. En måte er å betrakte egenskaper som gir tilstrekkelig like data men som samtidig klassifiseres til ulike klasser. En annen måte er å telle opp hvor mange ganger et et mønster forkommer i hele datamengden, og å trekke fra antall ganger det forekommer i den egenskapen som produserer det samme mønsteret flest ganger. En tredje mulighet er å summere opp alle inkonsistenser som kan produseres av en egenskapsdelmengde, og dividere dette antallet på det totale antallet av mønstre. Følgende pseudokode viser prosedyren for en generell filteralgoritme. Denne er rekonstruert fra [29].

```
1:  procedure FILTERALGORITME
    input :  D(F_0, F_1, ..., F_{n-1})      ▷ treningsdata med n egenskaper
             S_0                             ▷ delmengde som starter søket
             δ                               ▷ stoppekriterium
    output : S_beste                         ▷ optimal delmengde
2:      instansier S_beste = S_0;
3:      γ_beste = eval(S_0, D, M);     ▷ evaluerer S_0 med et uavhengig mål M
4:      while (δ ikke nådd) do
5:          S = genererer(D);          ▷ genererer en delmengde for evaluering
6:          γ = eval(S, D, M);              ▷ evaluerer delmengden S med M
7:          if (γ er bedre enn γ_beste) then
8:              γ_beste = γ;
9:              S_beste = S;
10:         end if
11:     end while
12:     return S_beste;
13: end procedure
```

Algoritme 1: En generalisert filteralgoritme

4.2.2 Wrapper

Wrappermetoden måler prestasjonen til de ulike egenskapsdelmengdene direkte med den endelige maskinlæringsalgoritmen. Dette innebærer nødvendigvis at nøyaktigheten etter endelig maskinlæring blir forholdsvis god, siden det er maskinlæringsalgoritmen selv som bestemmer hvilke egenskapsdelmengder som velges i seleksjonsprosessen. Maskinlæringsalgoritmen kan typisk baseres på klassifisering og klustrering. I tilfelle av førstnevnte brukes vanligvis nøyaktighetsprediksjon, og i tilfelle av sistnevnte kan klusterkvaliteten evalueres med mange heuristiske kriterier, som for eksempel kompakthet, separasjonsmuligheter og maksimal sannsynlighet [29]. Generelt kan resultatet av wrappermetoden vurderes etter generalitet, tidskompleksitet og nøyaktighet. Generalitet måler hvor godt den utvalgte delmengden passer for ulike klassifikatorer. Tidskompleksiteten måler tiden det tar for å velge ut delmengden,

og nøyaktigheten måler hvor presis prediksjonen er ved bruk av den valgte delmengden. Følgende pseudokode viser prosedyren for en generell wrapper-algoritme. Denne er rekonstruert fra [29], og er lik pseudokoden for filteralgoritmen med unntak av at evalueringsfunksjonen bruker en maskinlæringsalgoritme isteden for et uavhengig mål.

```
1:  procedure WRAPPERALGORITME
    input : D(F_0, F_1, ..., F_{n-1})      ▷ treningsdata med n egenskaper
            S_0                             ▷ delmengde som starter søket
            δ                               ▷ stoppekriterium
    output : S_beste                        ▷ optimal delmengde
2:      instansier S_beste = S_0;
3:      γ_beste = eval(S_0, D, A);      ▷ evaluerer S_0 med maskinlæringsalg. A
4:      while (δ ikke nådd) do
5:          S = generer(D);             ▷ genererer en delmengde for evaluering
6:          γ = eval(S, D, A);              ▷ evaluerer delmengden S med A
7:          if (γ er bedre enn γ_beste) then
8:              γ_beste = γ;
9:              S_beste = S;
10:         end if
11:     end while
12:     return S_beste;
13: end procedure
```

Algoritme 2: En generalisert wrapperalgoritme

Wrappermetoden går generelt for å være beregningsmessig mindre effektiv enn filtermetoden, men gir til gjengjeld høyere nøyaktighet i maskinlæringsalgoritmens klassifisering. Lavere nøyaktighet i filtermetoden skyldes at den ikke har kontakt med maskinlæringsalgoritmen, som en optimal løsning av seleksjonsproblemet ikke nødvendigvis er uavhengig av. Store seleksjonsproblemer skyldes tilfeller av store antall egenskaper og data (bilder). I slike tilfeller foretrekkes filtermetoden, da wrappermetoden har upraktisk eksekveringstid. For at wrappermetoden skal være egnet blir det uansett viktig å bruke effektive maskinlæringsalgoritmer, siden disse evaluerer hver delmengde av egenskaper i undersøkelsen.

4.2.3 Hybrid

Hybridmetoden gjør bruk av både filter og wrappermodellen [26, 29]. En typisk hybrid algoritme bruker gjerne filtermetoden til å bestemme den beste delmengden i hver kardinalitet av delmengder, for så deretter å ta i bruk maskinlæringsalgoritmen til å velge den beste delmengden blant disse beste i hver kardinalitet. Prestasjonen er gjerne en mellomting av prestasjonen i filter og wrappermetodene, både med tanke på kjøretid og nøyaktighet.

Det er vel og merke at hybridmodellen inkluderer et naturlig stoppekriterium. Følgende pseudokode viser prosedyren for en generell hybrid algortime. Denne er rekonstruert fra [29], og anvender både et uavhengig mål og en maskinlæringsalgoritme i evalueringen av potensielle delmengder.

```
1:  procedure HYBRID ALGORTIME
    input : D(F_0, F_1, ..., F_{n-1})        ▷ treningsdata med n egenskaper
            S_0                               ▷ delmengde som starter søket
    output : S_beste                          ▷ optimal delmengde
2:      instansier S_beste = S_0;
3:      C_0 = kard(S_0);                      ▷ kalkuler kardinalitet til S_0
4:      γ_beste = eval(S_0, D, M);            ▷ evaluerer S_0 med et uavhengig mål M
5:      Θ_beste = eval(S_0, D, A);            ▷ evaluerer S_0 med maskinlæringsalg. A
6:      for c = c_0 + 1 to n do
7:          for i = 0 to n − c do
8:              S = S_beste ∪ {F_j} ; ▷ generer en delmengde med kardinalitet c
    for evaluering
9:              γ = eval(S, D, M);            ▷ evaluer delmengden S med M
10:             if (γ er bedre enn γ_beste) then
11:                 γ_beste = γ;
12:                 S'_beste = S;
13:             end if
14:             Θ = eval(S'_beste, D, A);     ▷ evaluer S'_beste med A
15:             if (Θ er bedre enn Θ_beste) then
16:                 S_beste = S'_beste;
17:                 Θ_beste = Θ;
18:             else
19:                 break og return S_beste;
20:             end if
21:         end for
22:     end for
23:     return S_beste;
24: end procedure
```

Algoritme 3: En generalisert hybrid algoritme

4.3 Stoppekriterier

Stoppekriterier bestemmer når seleksjonsprosessen skal avbrytes. Disse forsøker å forhindre lang eksekveringstid utover hva som streng talt er nødvendig, og kan defineres på mange måter. Man kan naturligvis også la være å bruke stoppekriterier, og må da følgelig vente på at søket gjør seg ferdig. Et stoppekriterium kan for eksempel være en terskel, som bestemmer et minimalt antall

egenskaper, eller et maksimalt antall iterasjoner. Et annet kriterium kan være å sjekke når addisjon eller subtraksjon av egenskaper ikke skaper en bedre delmengde. Et annet naturlig stoppekriterium kan være å stoppe når en tilstrekkelig god delmengde er valgt, som for eksempel kan sjekkes ved at feil i klassifiseringen er mindre enn en viss terskel.

4.4 Vurdering av resultat

Resultatvurderingen avgjør om den aktuelle delmengden gir akseptable resultater. Siden apriori informasjon om dataene ofte er fraværende i applikasjoner fra den virkelige verden, må denne vurderingen i slike tilfeller baseres på indirekte metoder for å måle prestasjonen av egenskapsutvalget. Et eksempel på en slik metode er å måle klassifiseringsfeil før og etter seleksjonen, for å sammenlikne klassifikasjonen til utvalget opp mot klassifikasjonen til hele egenskapsmengden. Det må sies at denne vurderingen blir spesielt viktig som kompensasjon for tilfeller med enkle evalueringer og stoppekrierier, som for eksempel minimum antall egenskaper eller maksimal antall iterasjoner.

Kapittel 5

Automatisk visuell inspeksjon

Automatisk visuell inspeksjon (AVI) er attraktivt for industrien av flere grunner. I slike industrisammenhenger skjer gjerne produksjonen i høy hastighet, og det kan være krav om at samtlige produktindivider må kvalitetskontrolleres. Dette sammen med snevre toleranser kan gjøre manuel inspeksjon utilstrekkelig [36]. I tillegg kan det nevnes at menneskelige vurderinger er subjektive, hvilket gir lav reproduksjon av resultater som opparbeides [37]. Mennesker liker ikke kjedelig og ensformet arbeid, og må bruke mye ressurser på å dokumentere resultatet av inspeksjonen. AVI går for å være løsningen på mange av disse problemene.

En måte å definere AVI på kan være følgende trestegs prosedyre [38]:

1. Definer en minimal mengde egenskaper, som er i stand til å beskrive objektet i bildene tilstrekkelig godt.

2. Detekter og ta i bruk teknikker innen bildebehandling. Disse må være egnet til å beregne egenskapene i punkt 1 på en slik måte at nøyaktigheten og kjøretiden er akseptabel i en reel situasjon.

3. Implementer en beslutningsprosedyre i form av inteligente teknikker for maskinlæring.

Litteraturen beskriver en hel bråte med spesialtilpassede AVI systemer. Disse er bygget opp av skreddersydde algoritmer som presterer godt i sine tenkte omgivelser. Blant bruksområder med spesialtilpassede løsninger kan vi for eksempel nevne tekstilindustrien, pakke- og etikettindustri, trykte kretsløp (PCB), matindustrien, radiografi, bilindustrien, og fabrikkering generelt med mer [36, 37, 39, 40, 41, 42, 43, 44, 45, 46].

Siden spesialiserte systemer har for mange beskrankninger til å kunne brukes generelt, oppstår behovet for fleksible systemer. Hovedmålet med dette arbeidet er nettopp å undersøke og reflektere rundt forslag der AVI tar høyde for

generelle objekter. Følgelig gis det i dette kapittelet en forenklet beskrivelse av noe av det som finnes i litteraturen på dette området. Det må sies at jeg ikke har lykkes i å finne mye relevant litteratur om fleksible systemer. Dette kan tyde på at tilstrekkelig robuste og effektive metoder for slike systemer enda ikke er allment akseptert, og at utviklingen av slike fortsatt ligger på et eksperimenteringsstadium. I tillegg gjelder i stor grad at artikkelkildene holder seg på et relativt overordnet nivå i form av konsept og modulbeskrivelser.

Kapitlet deles inn i fire avsnitt med hver sin strategi for mer eller mindre fleksible systemer. Avsnitt 5.1 beskriver en metode for perseptuell egenskpasbasert objektgjenkjenning, avsnitt 5.2 gir en fremgangsmåte som leder til regelbasert AVI, avsnitt 5.3 beskriver et prototypekonsept i forskningsprosjektet SIOB, og avsnitt 5.4 beskriver en metode under navnet "General analysis graph".

5.1 Perseptuell egenskapsbasert objektgjenkjenning for automatisk inspeksjon

Brown, Forte, Malyan og Barnwell beskriver i [47] objektgjennkjenningsdelen av deres *Surface Mounted Electronic Assemblies* prototype for industriell AVI. Prototypen forsyner objektgjennkjenningsdelen med perseptuelt signifikante geometriske representasjoner av objektkonturer i flere oppløsninger. Dette gjøres i form av *Canny edge* deteksjon [5] og en ikke-lineær abstraksjonsprosedyre. Objektgjennkjenningsdelen sammenlikner så denne informasjonen opp mot lagrede modeller. Med dette utganspunktet bør det være mulig å bestemme om objektene på bildet godkjennes eller underkjennes, eller eventuelt graderinger av disse.

For å gjøre prototypen tilpasningsdyktig ovenfor flere bruksområder anvendes et objektorientert rammeverk både i prosseseringen og modelleringen av bildeinformasjonen. Ved å representere objektene på en slik objektorientert måte oppnås økt fleksibilitet og objektkarakteristikk i systemet. Komplekse objekter og klasser kan da modelleres, og man kan assosiere aktuell informasjon med et objekt eller deler av objektet. Det må sies at ulike industrielle bruksområder vil kreve ulike spesifikke deteksjon og sammenlikningsfunksjoner. Dog vil kjernen i systemet, samt eventuelle deteksjon og sammenlikningsfunksjoner som de ulike bruksområdene har felles, forbli uforandret.

Objektgjenkjenningsmodulen strukturerer informasjon i tre sammenkjedede deler; perseptuelt egenskapsnettverk (avsnitt 5.1.1), innhold (avsnitt 5.1.2) og arv (avsnitt 5.1.3). Perseptuelt egenskapsnettverk er en hierarkisk egenskapsbeskrivelse som konstrueres ved rekursiv gruppering av egenskaper fra

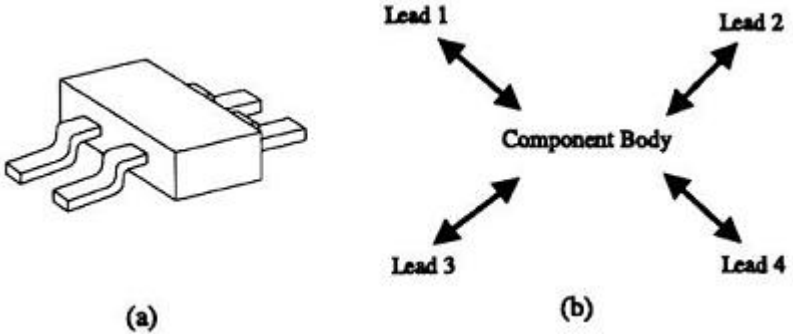
lavere til høyere nivå. Innhold er en trestruktur som beskriver koblinger mellom delkomponenter av objekter, slik at representasjon av komplekse så vel som enkle objekter blir mulig. Arv er en trestruktur som beskriver likheter hos objekter, og gir muligheten til å modellere flere variasjoner av samme type objekt. Selve egenskapssammenlikningen beskrives i avsnitt 5.1.4, og avsnitt 5.1.5 gir en vurdering av modulen.

5.1.1 Perseptuelt egenskapsnettverk

Det perseptuelle egenskapsnettverket (*Perceptual Feature Network*) [47] representerer signifikante elementer i et bilde med perseptuelle egenskaper som byggesteiner. Slike perseptuelle egenskaper kan for eksmepel være parallellitet, tilkoblingsbarhet, symmetri, polygon, overflate, hjørner og lignende. Nettverket gir en hierarkisk egenskapsbeskrivelse, der egenskaper på lavt nivå kan kombineres for å beskrive egenskaper på høyere nivå. Disse perseptuelle egenskapene utgjør primitivene i sammenlikningsprosessen.

5.1.2 Innhold

"Innhold" (*Containment Tree*) [47] er en trestruktur som beskriver hvordan uavhengige objekter er relatert til hverandre. Denne strukturen kobler sammen primitiver fra det perseptuelle egenskapsnettverket, og beskriver forholdet mellom disse ved hjelp av spesielle egenskapsbeskrivelser. Eksempler på slike relasjoner kan være beskrivelser om hvordan overflater eller kanter er koblet sammen, eller om en primitiv omslutter en annen. Figure 5.1 (kopiert fra [47]) illustrerer i *a)* en komponent, og i *b*) komponentens innholdstre.

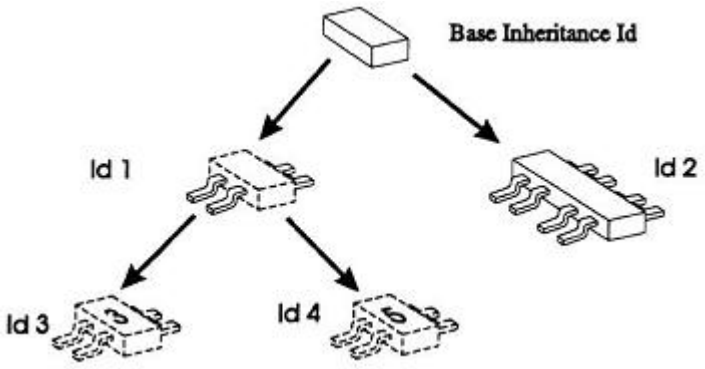


Figur 5.1: Innhold

5.1.3 Arv

"Arv" (*Inheritance Tree*) [47] er en mekanisme som gjør det mulig å representere ulike variasjoner av objekter som tilhører samme klasse. På denne

måten unngås et problemet med bare støtte for et begrenset antall strukturelle variasjoner. I tillegg vil man spare plass i tilfeller av mange modeller med høy grad av like egenskaper, og gjennkjenningsfasen kan bli mer effektiv siden primitivene er godt kategorisert. Med andre ord abstraheres felles strukturelle og dimensjonelle egenskaper opp til et høyere nivå. Figure 5.2 (kopiert fra [47]) illustrerer arv.



Figur 5.2: Arv

5.1.4 Modell og sammenlikning

To teknikker for automatisk generering av objektmodeller nevnes i [47]. Den ene er å bruke trådmodeller av objektene som skal gjenkjennes. Det perseptuelle egenskapsnettverket vil da bli konstruert fra relasjonene mellom alle segmentene i trådmodellen. Den andre er å konvertere objektinformasjon fra CAD modeller, for eksempel i form av en relasjonsgraf.

Modellene representeres som en database bestående av fem tabeller [47]:

1. Perseptuelle egenskaper: Denne tabellen representeres alle perseptuelle egenskaper sammen med referanser til delegenskaper, attributter og andre perseptuelle egenskaper, som deler de samme underliggende segmentene.

2. Delegenskaper: Denne tabellen holder på detaljerte karakteristikker om en bestemt perseptuell egenskap. Delegenskaper som har variabel verdi eller en rekke av verdier refereres videre til attributtabellen.

3. Attributter: Denne tabellen inneholder informasjon for både perseptuelle egenskaper og delegenskaper.

4. Kanter: Denne tabellen inneholder detaljert beskrivelse om alle relasjoner mellom komponentene i objektene.

5. Arv: Denne tabellen inneholder trestrukturen som tar for seg beskrivelsen av arv.

Sammenlikningsprosedyren er en generell mekanisme som sjekker egenskaper i bildet opp mot objekter i modelldatabasen. Den første fasen består av å sjekke for likheter i relasjonene mellom komponentene i et objekt (den mest abstraherte formen). Hvis relasjonene er gyldige blir mer detaljert informasjon fra hver delkomponent verifisert. Den objektorienterte strukturen tillater progressiv forfining i klassifiseringsfasen, og søket er derfor veldig effektivt.

Systemet benytter seg av en kontrollmekanisme som holder en liste over alle perseptuelle egenskaper, samt gyldige sammenlikninger mellom disse. Systemet blir derfor i stand til å velge den mest passende sammenlikningsfunksjonen for bestemte tilfeller. Ved innføring av nye egenskaper oppdateres denne informasjonen. Man må da oppdatere sammenlikningsfunksjonen mellom den nye primitiven og den mest beslektede klassen denne arver fra.

5.1.5 Vurdering

Brown, Forte, Malyan og Barnwell skriver i [47] at prosedyren for å konstruere det perseptuelle egenskapsnettverket er tidsmessig for tung til å kunne brukes i en praktisk industriell inspeksjonsapplikasjon (artikkelen ble utgitt i 1993). I samme artikkel skriver de at dette generelt synes å være tilfelle også for lignende objektorienterte AVI systemer.

På den positive siden reduserer objektorienteringen søkerommet og følgelig antallet "hit and miss" i sammenlikningsprosessen, slik at unødvendige sjekker forhindres. Det gis muligheter for å returnere klasseinformasjon også når objekter ikke stemmer helt overens med modellen, hvilket kan være nyttig hvis man for eksempel ønsker å spesifisere hvor feilen i det defekte objektet ligger.

Det underliggende rammeverket sørger for muligheten til å tilpasse seg forskjelige inspeksjonsproblemer relativt enkelt, da det er mulig å lage nye høyere nivås primitiver ut fra gamle primitiver. Nye applikasjonsspesifikke systemer trenger "bare" å forsynes med tillegsinformasjon om spesifikke primitiver med tanke på deteksjon og sammenlikningsfunksjoner, dersom disse ikke finnes i systemet fra før av. På denne måten er rammeverket i stand til å modellere og sjekke en stor mengde av objekter.

5.2 C4.5 og regelbasert AVI

Bariani, Cucchiara, Mello og Piccardi beskriver i [38] en automatisk kunnskapsdreven prosess for AVI. Beskrivelsen fokuserer på hvordan maskinlæring kan

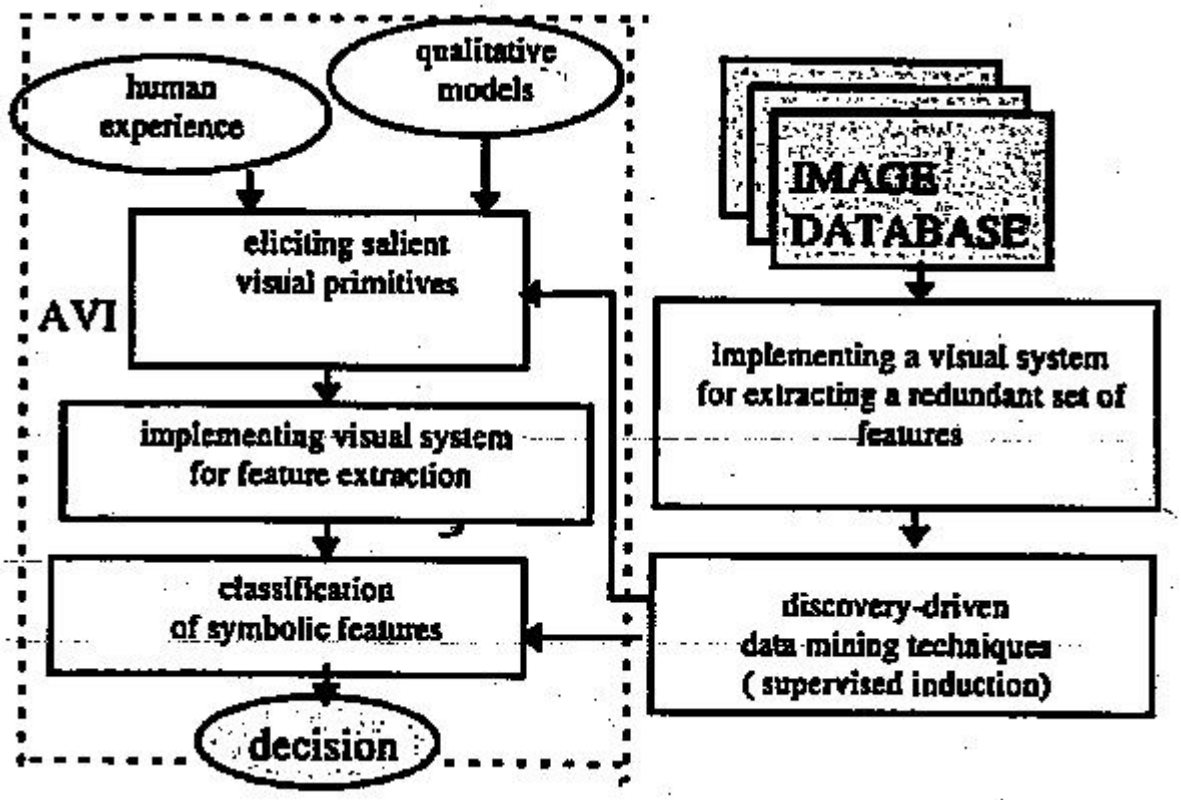
utnyttes i datagruvedrift (*data mining*). Mer spesifikt beskrives hvordan man kan velge en minimal mengde visuelle primitiver, for å kunne gjøre en pålitelig og robust klassifisering av de inspiserte objektene. Systemet baseres på regler som sluttes på bakgrunn av tidligere prosesseringer.

Figur 5.3 (kopiert fra [38]) viser en overordnet struktur av systemet. Her modelleres AVI og datagruvedrift på henholdsvis venstre og høyre side. Første modul på AVI-siden definerer pålitelige, stabile og minimal mengde av egenskaper, som er i stand til å beskrive den aktuelle modellen. Disse er under inflytelse av tidligere erfaringer og kvalitative modeller. Med sistnevnte menes informasjon som er vanskelig å måle spesifikt, i motsetning til kvantitative mål som eksemelvis areal og perimeter. Dette er typisk den informasjonen som menneskelige eksperter lærer. De kvalitative modellene åpner dørene for en stor mengde applikasjoner [38]. Neste modul detekterer og implementer teknikker som kan beregne de visuelle primitivene fra forrige modul med god nok nøyaktighet, samt innenfor rimelig tidskompleksitet. Klassifiseringsmodulen tolker så egenskapsberegningene, og disse resultatene ligger til grunn for den endelige beslutningen. Høyre side illustrerer datagruvedriften. Først ekstraheres en redundant mengde egenskaper fra bildedatabsen ved bruk av en mange spesifikke datasynteknikker. Disse blir så treningssettet til datagruvedrifmodulen, som kan inkludere beslutningstrær, nevrale nettverk, symbolske læringsalgorimter og en mengde andre teknikker [38]. Den overvåkede læringen har som hensikt å indusere regler til bruk i klassifiseringsmodulen. Spesielt velger denne passende parameterverdier (f.eks. terskelverdier) med hensyn på å kunne å avsløre defekte objekter. For å innkludere usikkerhet i modellen opereres det med grader av "fuzzyness" i reglene [38]. Den overvåkende læringen sørger også for å luke bort egenskaper som ikke er obligatorisk for klassifiseringen. Dette utgjør a-posteriori informasjon til modulen som velger ut passende egenskaper på AVI-siden.

Rammeverket beskrevet ovenfor er blant annet spesifikt testet med maskinlæringsalgoritmen *C4.5* som datagruvedriftmodul [38]. Denne lærer hvordan korrekt klassifisering skal gjennomføres ved å tolke treningssettet, bestående av attributtverdier til objektene i bildedatabasen. *C4.5* modelleres som et beslutningstre med løvnoder korresponderende til klassifiseringsresultater, som i dette tilfellet tilsvarer defekte og ikke-defekte objekter. Etter hvert som treningssettet tolkes vil beslutningstreet bygges. Noder som ikke er løvnoder i beslutningstreet spesifiserer tester, som vurderer treningssettets attributtverdier, og avgjører på bakgrunn av dette hvilken grein eller deltre som velges for videre prosessering.

C4.5 har evnen til å utlede produksjonsregler ut fra beslutningstreet. Disse reglene er enklere og mer kompakte, og fører følgelig til raskere klassifiseringer kjøretidsmessig. Samtidig indikerer disse en ekslisitt måte å klassi-

fisere objektene på, hvilket innebærer at disse lett kan tolkes og kontrolleres av menneskelige eksperter. Algoritmen har med andre ord evnen til å generalisere klassifiseringsprosessen.



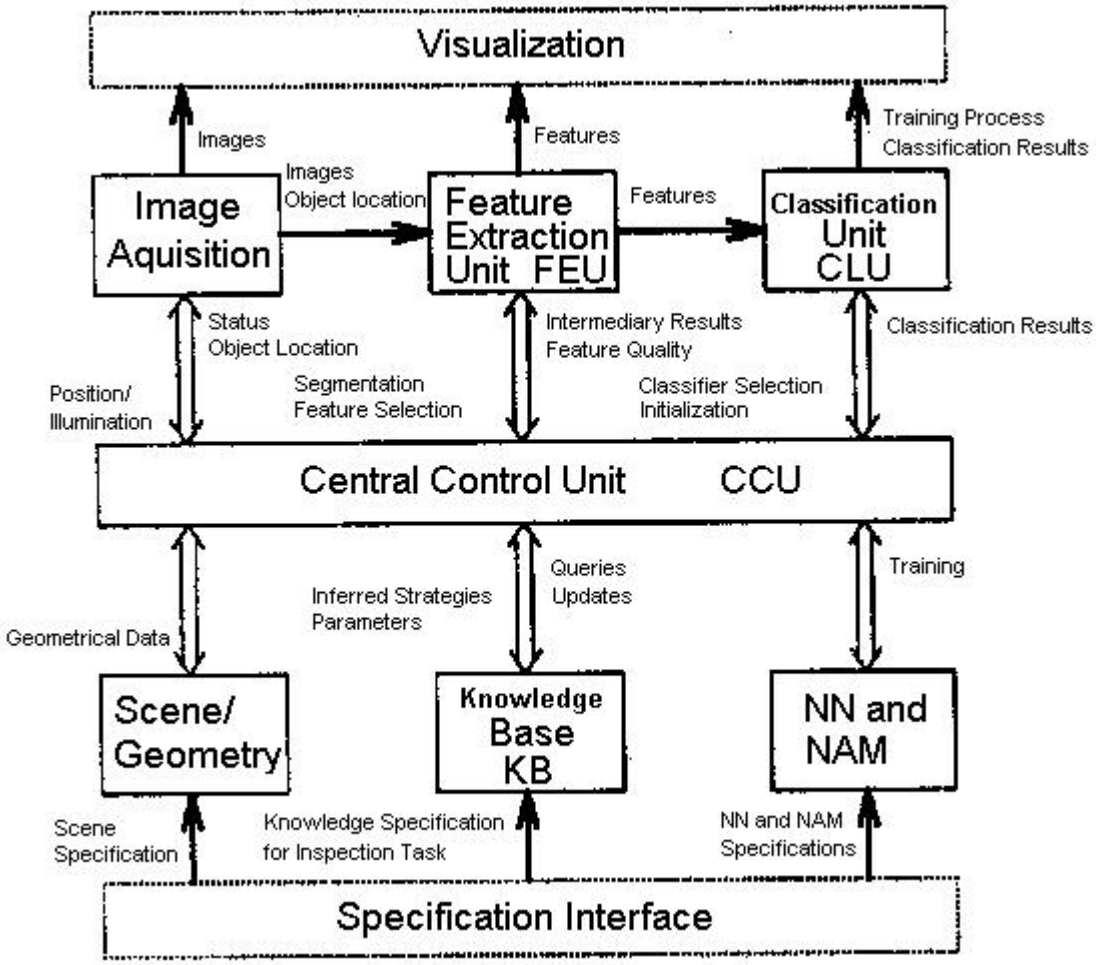
Figur 5.3: Datagruvedrift i auotmatisk visuell inspeksjon

5.3 Prototype fra forskningsprosjektet SIOB

SIOB er et forskningsprosjekt med målsetning å utvikle et hybrid og dynamisk system med selvorganiserende egenskaper og fleksibel topologi, som inkluderer bildebehandling og egenskapsuttrekking, kunnskapsprosessering, nevrale nettverk og NAM moduler [48].

Figur 5.4 (rekonstruert fra [48]) illustrerer en prototype av SIOB i form av oppdeling i enheter og kommunikasjon mellom disse. Prototypen konfigureres med resultater fra en selvovervåkende prosess samt apriori kunnskap. Den sentrale kontrollenheten (CCU) kontrollerer prosessene i de andre enhetene og overvåker resultatene. Systemet sies i [48] å være i stand til å detektere feil og komme opp med alternative prosesseringer. En essensiell grunn til dette er at kommunikasjonen mellom den sentrale kontrolenheten

og de andre enhetene kan gå begge veier. Systemet argumenteres derfor for å ha evenen til å organisere seg selv, som for eksempel innebærer å finne frem til riktige parametere i enhetene.



Figur 5.4: Inspeksjonssytem konsept

Kunnskapsbasen (KB) blir instansiert med informasjon som er samlet fra eksperimenteringer eller eksplisitt kunnskap. Den eksplisitte kunnskapen kan for eksempel omhandle produksjonsprosessen, typer feil og deres posisjoner i bildet, delelister, bilderegioner og lignende. I tillegg innkluderer kunnskapsbasen egenskapsalgoritmer og klassifiseringsprosedyrer. Den ekslisitte kunnskapen kan bestemmes interaktivt for hvert objekt som skal inspiseres, og objekter i bildet lokaliseres etter en kantmodell av denne informasjonen. Dette skjer i forkant av segmenteringsprosessen. Selve segmenteringsprosessen gjøres etter to forskjellige innfallsvinkler, som komplementerer hverandre. Den ene baserer seg på den strukturelle informasjonen som finnes

i systemets kunnskapsbase, som kan brukes til å segmentere objektet i en mengde inspiserte regioner. Den andre metoden legger først et homogent rutenett over objektet som skal inspiseres, og trekker så ut egenskaper i hver rute. Sistnevnte metode brukes utelukkende til å indikere om objektet har en feil eller ikke.

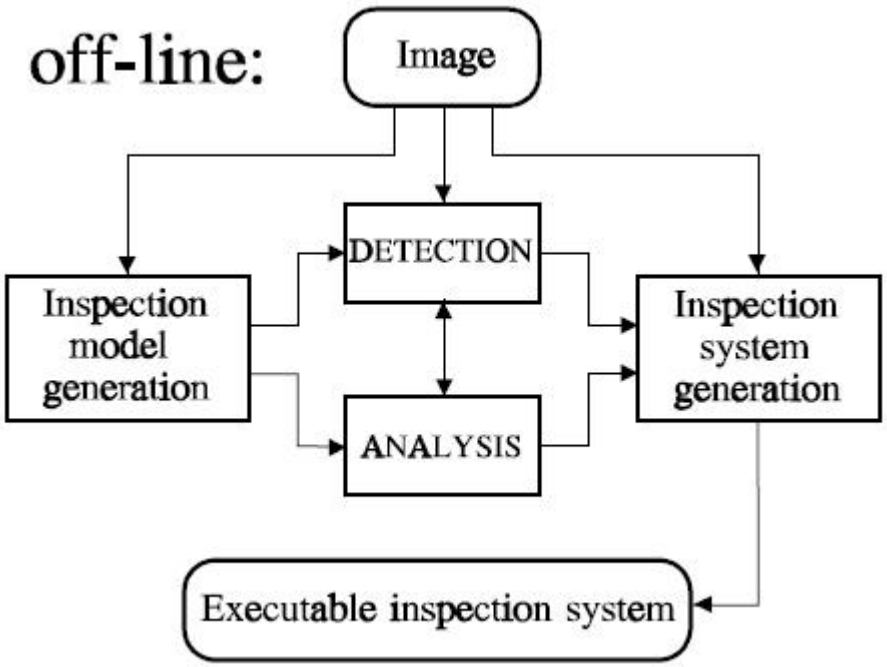
Valget av egenskapsuttrekningsmetoder og klassifiseringsmetoder blir gjort av den sentrale kontrollenheten, og egenskapene beregnes på bakgrunn av de segmenterte regionene. Egenskapsuttrekningsenheten (FEU) består en kjerne som utgjør grensesnittet mot et vilkårlig antall uavhengige egenskapsuttrekkingsmoduler, og kan derfor bestå av en mengde teknikker innen egenskapsuttrekning. Det samme gjelder for klassifiseringsenheten (CLU), der nevrale nettverk og NAM paradigmer er tilgjengelige [48].

5.4 Generell grafanalyse

Sablatnig beskriver et systematisk og fleksibelt konsept for AVI i [49, 50, 51]. Konseptet er modellbasert, og bygger på seperasjonen mellom en applikasjonsuavhengig egenskapsdeteksjon og en applikasjonsavhengig modellanalyse. Dette åpner dørene for adopsjonsmuligheter i flere applikasjonsområder.

Systemet har to hovedfaser; "off-line" og "on-line". Førstnevnte kan sies å konfigure systemet for det bestemte bruksområdet, mens sistnevnte kjører systemet i en reell praktisk situasjon på bakgrunn av resultatene fra "off-line"-fasen. Figur 5.5 (kopiert fra [49]) viser "off-line" delen av systemet. Fire moduler sørger for prosesseringen fra bildetdata ("Image") til et ferdig konfigurert inspeksjonssystem ("Executable inspection system"). Først ut er "Inspection model generation"-modulen (avsnitt 5.4.1), som konstruerer inspeksjonsmodellen ved å lete opp egnede primitiver for applikasjonsområdet, samt relasjoner mellom disse. Dernest kommer "Detection'-modulen' (avsnitt 5.4.2), som finner frem til selve egenskapsdeteksjonsalgoritmene, som er i stand til å detektere primitivene i inspeksjonsmodellen. I nær dialog med de to foregående modulene er "Analysis"-modulen (avsnitt 5.4.3), som justerer parameterverdier og rekkefølgen til deteksjonsalgoritmene, og lager en ny graf der denne informasjonen er bakt inn. Sist ut er "Inspection system generation"-modulen (avsnitt 5.4.4). Her vurderes det om resultatene man er kommet frem til er i stand til å detektere egenskapene i ikke-defekte bilder, og eventuelle beregninger av nøyaktighet og tidsbruk på dette. Samme modul vil ved positivt resultat på disse testene gjøre en siste sjekk, der den anvender både defekte og ikke-defekte bilder. Dersom kravene innfris er systemet klart for å gå "on-line". Hvis ikke må justeringer foretas. Siden brukeren har betydelig og verdifull informasjon om det spesifikke inspeksjonsproblemet, må det sies at konseptet har gjort rom for brukerinteraksjon i "Inspection

model genertation", "Analysis" og "Inspection system generation"-modulene.

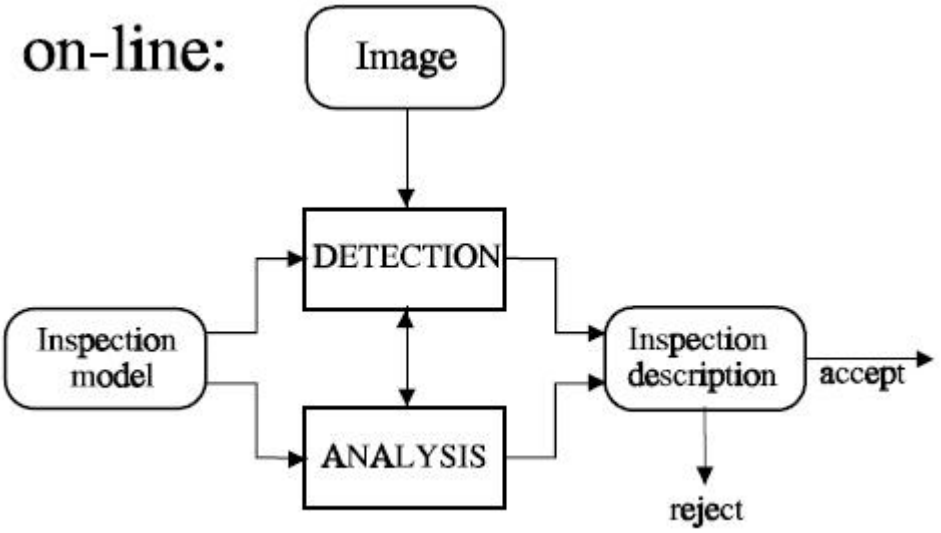


Figur 5.5: Off-line

Figur 5.6 (kopiert fra [49]) viser "on-line" delen av systemet. Bildedata ("Image") fanges med samme betingelser som under "off-line"-fasen. De algoritmene som er funnet aktuelle i "Detection"-modulen, brukes etter den rekkefølgen og de parametere som henholdsvis er bestemt av analysemodulen og inspeksjonsmodellen. I analysemodulen sjekkes det om egenskapene tilfredsstiller inspeksjonsmodulens toleranseverdier. Inspeksjonsbeskrivelsesmodulen vurderer så resultatene, og avgjør om objektet aksepteres eller forkastes.

5.4.1 "Inspection model generation"-modul

Målet med "Inspection model generation"-modulen er å komme frem til en måte å beskrive komplekse objekter på ved hjelp av en relativt liten mengde enkle primitiver og relasjoner mellom disse. For det generelle kosneptet er det viktig at disse primitivene ikke har begrensninger som går på bekostning av generalitet. Selve modellen er en graf struktur, der noder representerer objekter, delobjekter og primitiver, og kanter representerer relasjoner mellom dem. I tillegg benyttes attributter på kanter og noder. Attributtene representerer vekter og toleranseverdier, og gis som et suplement til inspeksjonsmodellen i form av apriori informasjon og brukerinteraksjon. Som apriori informasjon nevner Sablatnig CAD-modeller som eksempel, da disse



Figur 5.6: On-line

har veldefinerte matematiske beskrivelser av objektet, og er egnet til å verifisere deteksjonsresultater og til å gi toleranseverdier [49].

Sablatnig beskriver et rammeverk for nøyaktig og robust ekstrahering av parametriske primitiver i [49]. Dette deles inn i fire hovedkomponenter; *Exploration*, *Selection*, *Fit* og *Final selection*, og kan veldig kort beskrives på følgende måte. Utforskningskomponenten (*exploration*) er en dynamisk datadreven masketeknikk som foreslår en mengde modeller. Seleksjonen (*selection*) utføres med "tabu search"-metoden, som velger den modellen som beskriver dataene med minimal lenge på beskrivelsen. Tilpasningskomponenten (*fit*) øker nøyaktigheten i den valgte modellen, og kan endre klassifiseringen av dataelementer som lokalitet og retning. Den endelige seleksjonen (*final selection*) gir så en liste av aktuelle parametriske primitiver.

5.4.2 "Detection"-modul

"Detection"-modulen inneholder et stort bibliotek av deteksjonsalgoritmer. Det bør være flere algoritmer tilgjengelig for alle primitiver i inspeksjonsmodellen, og modulens målsetning er å velge ut passende algoritmer til disse. For at dette skal være mulig er det nødvendig å representere algoritmene med grensesnitt, som beskriver algoritmens funksjonalitet sammen med akseptable "input" og "output". Med slike representasjoner tilgjengelig kan man velge ut en algoritme eller kombinasjoner av flere algoritmer fra biblioteket, samt evaluere bruken av disse. Evalueringen må ta høyde for om valgte algoritmer oppnår en lovlig løsning for det gitte problemet.

5.4.3 "Analysis"-modul

"Analysis"-modulen justerer parametere, rekkefølge og søkerom til de valgte algoritmene fra deteksjonsmodulen. Modulen opererer med applikasjonsspesifikk informasjon, og har som en viktig oppgave å sette vektene i inspeksjonsmodellen, da det hittil er ukjent hvilke primitiver som er troverdige og viktige for gjennkjenning. Vektene tilpasses den gitte inspeksjonsoppgaven ved å teste ut og evaluere deteksjonsalgoritmenes rekkefølge og parameterverdier. En ny graf (analysegrafen) bygges på bakgrunn av disse resultatene og inspeksjonsmodellen for øvrig. Denne grafen representerer hele rommet av mulige løsninger for det spesifikke problemet, og representeres som et hierarki av deteksjonsalgortimer (noder), og semantiske relasjoner (kanter) mellom nodene. På denne måten gis det en spesifikk beskrivelse av deteksjonsalgoritmenes rekkefølge, samt parameterverdiene til disse.

5.4.4 "Inspection system generation"-modul

Siden analysegrafen i "Analysis"-modulen genereres som en delmengde av den totale mengde treningsbilder, gjøres en ny sjekk om inspeksjonssystemet virker tilstrekkelig godt med andre testbilder i en "on-line"-simulering. Det gjøres rom for at brukeren interaktivt skal kunne avgjøre om systemet innfrir de industrielle kravene. Eventuelle interaksjoner medfører at analysegrafen instantieres på nytt, og da med mulighet for en redusert mengde egenskaper. Nye tester gjøres på både defekte og ikke-defekte treningsbilder for å avgjøre feilklassifiseringsraten på begge sider. Det endelige inspekssjonssystemet avgjør om inspeksjonssystemet er egnet, eller må tilpasses ytterligere. Hvis testene bekrefter at systemet oppnår de vesentligste kravene for inspeksjonsoppgaven, gjøres inspeksjonssystmet klart til å gå "on-line".

Kapittel 6

Løsningsforslag

Dette kapittelet gir et teoretisk forslag på oppbygningen av et fleksibelt AVI system. Det har vært en viktig målsetning at systemet skal være mest mulig automatisk, slik at flest mulig parametere gjelder for generelle bilder. Forslaget antar at tilstrekkelig store treningssett er tilgjengelig for det aktuelle inspeksjonsproblemet, og at disse er kategorisert i godkjente og underkjente bilder. For å begrense oppgaven har løsningsforslaget valgt å se helt bort fra nødvendigheten av forhåndsprosessering av bildene. Forslaget skiller seg klart fra de fremgangsmåtene som er beskrevet i kapittel 5. Jeg mener at dette løsningsforslaget baserer seg på en enklere strategi. Denne er blant annet uavhengig av valg av primitiver og flere deteksjon- og sammenlikningsmetoder, og bruker ikke ressurser på å bygge opp en objektorientert modell av objektene som avbildes. Til tross for dette viser løsningen stor fleksibilitet for ulike bruksområder i form av å være raskt konfigurerbar for disse. Bare et relativt beskjedent antall parameterverdier må settes i henhold til enkel apriori informasjon om det aktuelle bruksområdet (antall objekter, størrelse, fargeforhold og lignende). Løsningen har en god modularitet, og kan lett utvides til å støtte flere egenskaper. På en enkel måte støtter systemet også en spesialtilpasset løsning for spesifikt bruksområde dersom brukeren ønsker dette, som kan gjøres ved å "avgeneralisere" parameterverdier. Et minus kan riktignok være at løsningen krever implementasjon av svært mange egenskapskandidater, og at 2 parameterverdier må settes for hver av disse. Det er derimot kun nødvendig å gjøre dette en gang, men denne operasjonen krever mange treningssett med stor variasjon. En annen ulempe med løsningen er at den er sterkt avhengig av prestasjonen til segmenteringsalgoritmen, da segmenering og egenskapsvurdering er separert.

Kjernen i hele løsningsforslaget ligger i å lete opp egnede egenskaper basert på følgende to hovedkrav:

1. Små forskjeller på beregninger av egenskapsbeskrivelser mellom godkjente bilder seg imellom.

2. Store forskjeller på beregninger av egenskapsbeskrivelser mellom godkjente og underkjente bilder.

I likhet med "Generell grafanalyse"-metoden beskrevet i kapittel 5 deles systemet inn i to faser; konfigureringsfase ("off-line") og eksekveringsfase ("on-line"). Kort sagt stiller konfigureringsfasen inn systemet for det aktuelle inspeksjonsproblemet, og eksekveringsfasen kjører resultatet av konfigureringsfasen i en reell situasjon. Disse to fasene beskrives i henholdsvis avsnitt 6.1 og 6.2.

6.1 Konfigureringsfase

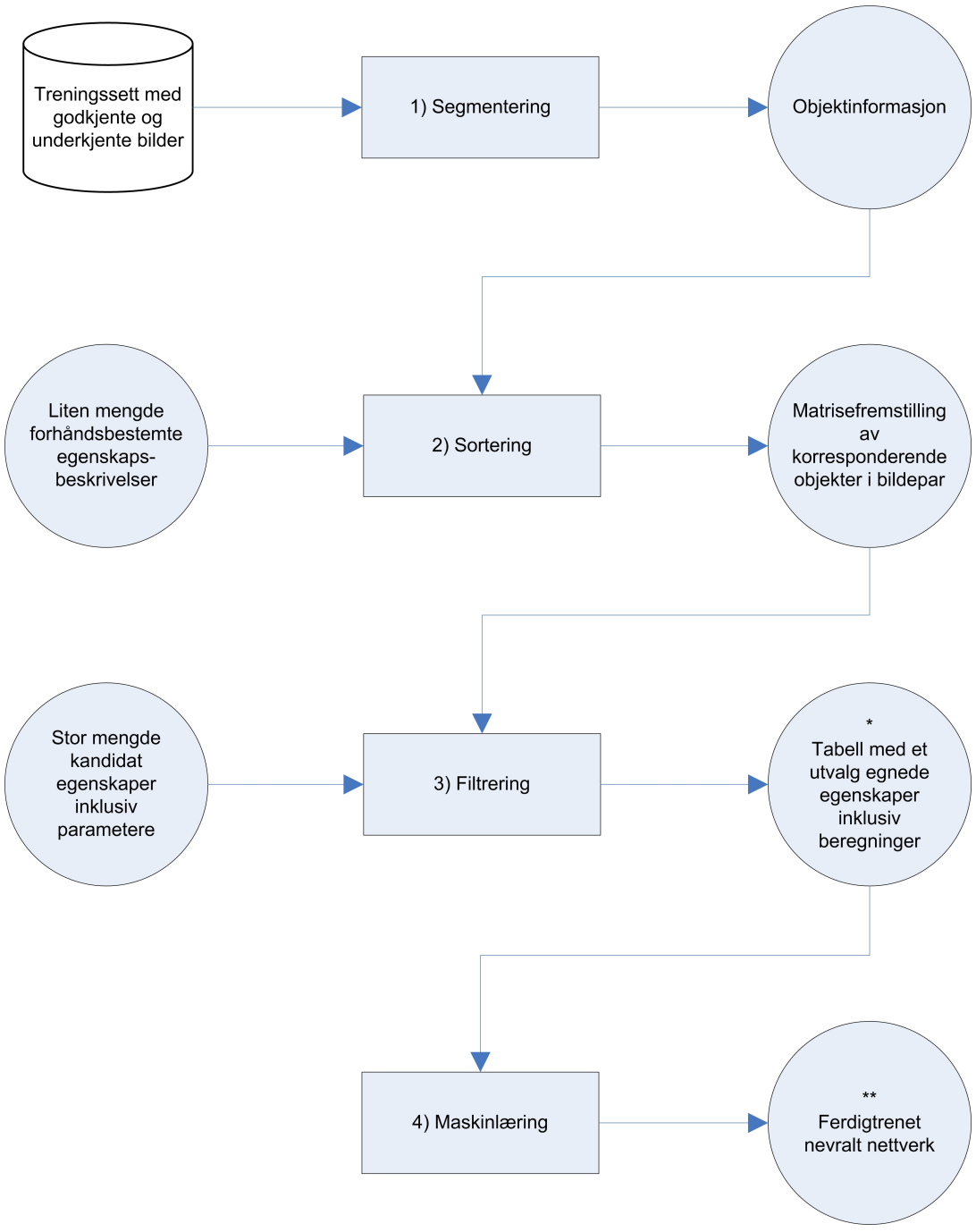
Hensikten med konfigureringsfasen er å forsyne eksekveringsfasen med nødvendig informasjon som trengs for å klassifisere objekter i et spesifikt inspeksjonsområde. For å være i stand til dette mates konfigureringsfasen med bilder fra treningssettet, sammen med informasjon om disse inneholder godkjente eller underkjente objekter. På bakgrunn av disse dataene, og en stor mengde egenskapsbeskrivelsesmetoder, er målet at konfigureringsfasen skal produsere følgende:

- Et utvalg egnede egenskaper som er i stand til å beskrive de aktuelle objektene på en tilsrekkelig godt måte.
- Et ferdigtrenet nevralt nettverk som er i stand til å klassifisere de aktuelle objektene som godkjent eller underkjent.

Figur 6.1 gir en overordnet illustrasjon av konfigureringsfasen. Her illustreres prosesser (rektangler) og ressurser (sirkler). De ressurser som brukes av eksekveringsfasen er merket med (*) og (**). Den videre beskrivelsen av konfigureringsfasen deles inn i fem underavsnitt, der de fire første korresponderer til prosessene i figuren og modulene i systemet. Disse gir en dypere beskrivelse av hver sin modul, som inkluderer den aktuelle prosessen samt de ressursene som denne prosessen bruker og skaper. Det siste avsnittet oppsummerer viktige trekk angående konfigurasjon og parameterverdier.

6.1.1 Segmentering

Systemet starter med å segmentere alle bildene i bildedatabasen (treningssettet). På grunn av store krav til fleksibilitet i systemet, må det anvendes generelle segmenteringsteknikker, og i utgangspunktet var *watershed* (se avsnitt 2.3.4) med påfølgende *region merging* (se avsnitt 2.3.1) planlagt. Denne framgangsmåten ble tidlig i arbeidet testet ut ved hjelp av det åpne biblioteket *Insight Segmentation and Registration Toolkit* (*ITK*, se vedlegg). Dette biblioteket inneholder blant annet segmentering basert på *watershed*.



Figur 6.1: Konfigureringsfase

Det viste seg at ITK implementasjonen ikke kunne brukes til tross for akseptable segmenteringsresultater. Årsaken til dette var lang kjøretid på litt i underkant av 2 minutt per 352 x 258 bilde. Strategien ble derfor å finne en tilsvarende robust men kjappere segmenteringsmetode. Løsningen falt på *Richard Blake's fargesegmenteringsmetode* [9] (se avsnitt 2.3.5). Denne ble valgt fordi metoden fremstår som enkel, robust og generell i forhold til de mange andre segmenteringsmetoder som er beskrevet i kapittel 2. Det er klart at segmenteringsmetoden langt fra vil gi gode segmenteringsresultater for alle bilder. Håpet er allikevel at den skal kunne forsyne systemet med tilstrekkelig gode segmenteringsresultater.

Samtlige segmenteringsresultater registreres sammen med en referanse til sitt bilde i tabell (markert som *Objektinformasjon* i figur 6.1). Tabellen opprettholder oversikten mellom godkjente og underkjente bilder. Tabell 6.1 eksemplifiserer dette på en generell måte, der treningssettet består av n godkjente bilder og m underkjente bilder. Her har segmenteringen funnet a antall objekter i det første godkjente bildet, b antall objekter i det siste godkjente bildet osv.

G1	...	Gn	U1	...	Um
O1		O1	O1		O1
⋮		⋮	⋮		⋮
Oa		Ob	Oc		Od

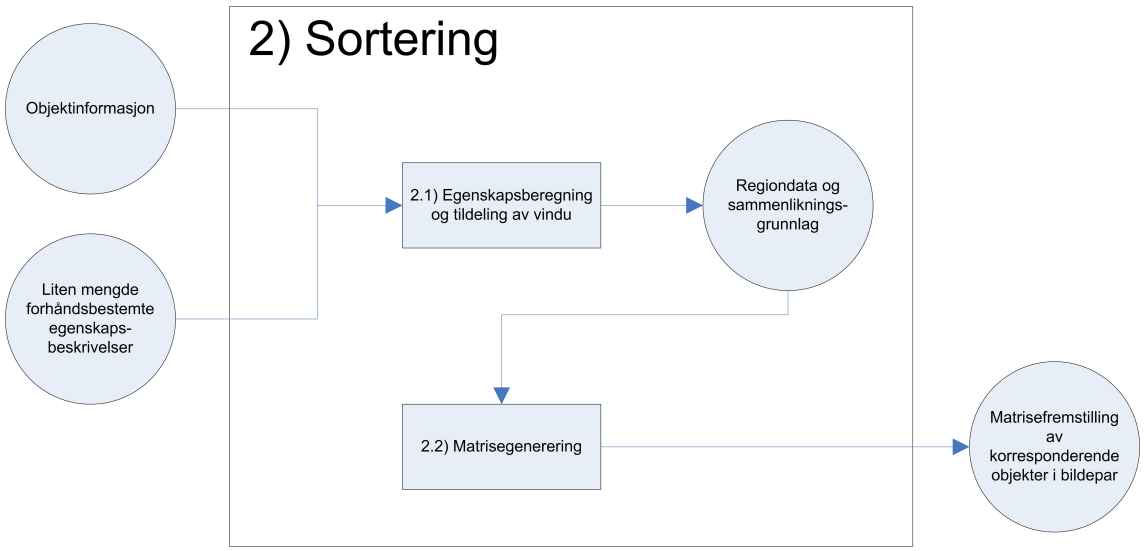
Tabell 6.1: Objektinformasjon

6.1.2 Sortering

Sorteringsmodulens oppgave er å finne korresponderende objekter i forskjellige bilder, samt registrere denne informasjonen som matriserepresentasjoner av bildepar. Hensikten er å spare neste modul (*filtrering*) for mye ekstraarbeid ved at denne "bare" kan konsentrere seg om de objektene i aktuelle bildepar som er "mappet" i sorteringsmodulen, når denne skal vurdere egenskaper og likheter i mer detaljert grad.

Som input tar modulen inn segmenteringsresultatene (som vist i tabell 6.1) og en liten mengde forhåndsbestemte egenskapsbeskrivelser. Disse forhåndsbestemte egenskapsbeskrivelsene utgjør grunnlaget for sorteringen, og er tenkt å gjelde for generelle anvendelsesområder. Jeg mener at følgende lille mengde av egenskapsbeskivelser kan egne seg (se kapittel 3):

- Areal
- Massesenterpunkt



Figur 6.2: Sortering

- Omsluttende rektangel

Massesenterpunktet posisjonerer objekter i bildet, og areal beskriver objekters størrelser. Det omsluttende rektangelet dekker både objekters posisjon og størrelse. Disse er tenkt å være gunstige som sorteringsfundament, da de segmenterte objektene ikke kan ha samme eller overlappende posisjon i bildet. Arealmålet bidrar også til økt robusthet i tilfeller av translasjon og rotasjonsforskjeller mellom bilder. Alle tre egenskaper er i tillegg beregningsmessig billige å gjennomføre.

Figur 6.2 illustrerer trinnene i sorteringsprosessen. Beregninger gjøres for det lille egenskapsutvalget. Resultatet (*regiondata*) registreres for hver av de tre egenskapene i sin egen tabell, som illustrert i tabell 6.2, der *F(objekt)* betegner den aktuelle egenskapsberegningen av objektet.

G1	...	Gn	U1	...	Um
F(O1)		F(O1)	F(O1)		F(O1)
⋮		⋮	⋮		⋮
F(Oa)		F(Ob)	F(Oc)		F(Od)

Tabell 6.2: Egenskapsberegning

I figur 6.2 ser vi også at tildeling av vinduer finner sted. Dette gjøres for samtlige segmenterte regioner, slik at hver region assosieres med et vindu. Hensikten er å avlaste kompleksiteten til maskinlæringsmodulen ved å sørge for en viss orden i inputvektorens innhold. Det være seg for eksempel at en

region i bildets øverste venstre del tilsvarer første rad i inputvektoren, og en region i bildets nederste høyre del tilsvarer siste rad i inputvektoren. Dette er helt essensielt for å oppnå korrespondanse mellom systemets treningsfase og eksekveringsfase, som henholsvis trener og kjører det nevrale nettverket. For at inputvektoren i hver av disse to fasene skal følge samme ordning, kan vi dele det aktuelle bildet inn i vinduer, og la hvert vindu tilsvare visse områder i inputvektoren. Hvert vindu dekker da hvert sitt pikselområde, og kun en region som har massesenterpunkt innenfor dette kan "besitte" det aktuelle vinduet. Det følger da naturlig at det kan oppstå en konflikt om samme vindu mellom to eller flere regioner. Det blir derfor vikig å dele bildet inn i et passende antall vinduer, slik at hver region av betydning får "besitte" sitt vindu. På bakgrunn av dette foreslår vi en parameter som bestemmer hvor mange vinduer bildet skal deles inn i. For liten parameterverdi resulterer i tap av viktige regioner, og for stor verdi kan tillate for mange regioner og skape unødvendig kompleksitet for systemet. Figur 6.3 illustrerer et bilde inndelt i vinduer og korresponderende inputvektor. Her symboloserer $V_1, ..., V_9$ hvert sitt vindu, F_1, F_2, F_3 tre egenskapskandidater, og $r_1, ..., r_9$ de regioner (kan også være tomme) som "besitter" henholdsvis vindu $V_1, ..., V_9$. Det er verdt å merke at parameterverdien for vinduer i dette eksempelet er 3 (bildets bredde og høyde deles inn i 3, slik at antall vinduer blir 3^2). Figuren er også en forenkling, da alle egenskapskandidater her gir skalare resultater av $F_i(r_j)$. I praksis kan beregninger av egenskapskandidatene F_1, F_2, F_3 anvendt på regioner gjerne gi resultater på vektorform. Et viktig poeng som derimot illustreres godt i figuren er at størrelsen til inputvektoren øker sterkt med antall vinduer (samt med antall egnede egenskapskandidater og deres krav til vektordimensjon). Generelt kan inputvektorens størrelse altså skrives som:

$$Dim(V) = W \sum_{i=1}^{N} Dim(F_i) \tag{6.1}$$

der

V er inputvektor

$W = p^2$ er antall vinduer (lik kvadratet av parameterverdien p)

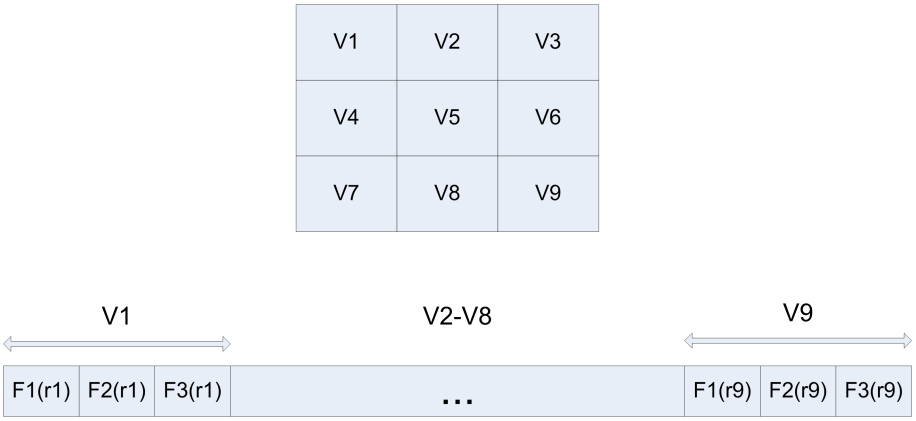
F_i er egnet egenskapskandidat nr i

N er antall egnede egenskapskandidater

Dim er vektordimensjon

Når det gjelder selve sammenlikningen av to objekter fra ulike bilder, så gjøres dette som en vektet sum av differansen mellom egenskapsberegningene av disse. Matematisk kan vi beskrive dette på følgende måte:

$$V(O_i, O_j) = \sum_{k=1}^{E} v_k [F_k(O_i) - F_k(O_j)]^2 \tag{6.2}$$



Figur 6.3: Vinduinndeling og innputvektor

der

$V(O_i, O_j)$ er den samlede vekten til sammenhengen mellom objekt i og objekt j
E er antall egenskaper i bildet (dvs 3 for utvalget vårt)
v_k er vekten til egenskap nr k
$F_k(objekt)$ er beregningen av egenskap nr k til objektet

Etter å ha beregnet alle mulige objektmappinger mellom to bilder, lagres resultatene i matriseform som vist i tabell 6.3. *B1.O1* betegner her objekt 1 til bilde 1 osv.

B1/B2	B2.O1	...	B2.Om
B1.O1	V(B1.O1,B2.O1)	...	V(B1.O1,B2.Om)
⋮	⋮		⋮
B1.On	V(B1.On,B2.O1)	...	V(B1.On,B2.Om)

Tabell 6.3: Sammenlikningsgrunnlag

Neste steg (*matrisegenerering*) er å finne de objekter i bildeparet som mapper hverandre best i henhold til sammenlikningsgrunnlagmatrisene. Som en standard velges bilde 1 som bildet med det minste antallet objekter n, og bilde 2 som bildet med det største antallet objekter m. For bilder med like mange objekter (n x n matriser) skal alle objektene i bilde 1 mappes med hver sitt objekt fra bilde 2. I tilfeller av ulikt antall objekter i bildeparene (n x m matriser), gjøres samme optimale mapping av de n objektene fra bilde 1 opp mot n objekter fra bilde 2. De m-n resterende objektene i bilde 2 mappes i etterkant til hvert sitt objekt i bilde 1 med minst verdi i sammenlikningsgrunnlagmatrisen. Dette innebærer at 1:1 korrespondansen i n x m matrisen

brytes. Dette er nødvendig for at konfigureringsfasen i filtreringsfasen skal være i stand til å sette viktige terskelverdier. Avsnitt 6.1.3 beskriver dette mer.

En rett frem rekursiv strategi for optimal løsning på "mappeproblemet" kan beskrives på følgende måte:

$$f(R,K) = \begin{cases} \min_b \{C_{ab}\} & \text{for } |R| = 1 \\ \min_{a \in R, b \in K} \{f(R-a, K-b) + C_{ab}\} & \text{ellers} \end{cases} \tag{6.3}$$

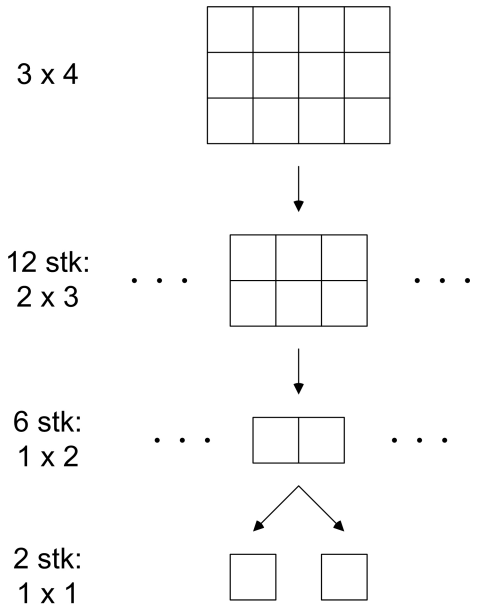
der

R er en mengde som består av rader i matrisen
K er en mengde som består av kolonner i matrisen
$f(R,K)$ er den minimale summen av R elementer for matrisen med dimensjon $|R|$x$|K|$
C_{ab} er kostnaden/vekten til element med posisjon (a,b) i matrisen

Figur 6.4 illustrerer de rekursive trinnene for en matrise med dimensjoner n=3 og m=4. Siden n=3 må det gjøres et utvalg av 3 elementer fra matrisen. Hver av disse kan ikke ha felles rad eller kolonne med noen av de andre to elementene, siden alle objekter i bilde 1 skal mappes med hvert sitt objekt i bilde 2. På toppnivå (dybde 0) i rekursjonen er det derfor 3x4=12 muligheter til å velge det første elementet. På dybde 1 er matrisedimensjonen redusert til (n-1)x(m-1)=2x3 siden et element fra dybde 0 allerede er valgt. Den raden og kolonnen som dette elementet representerer er da fjernet i dybde 1. For hver av de 12 matrisene i dybde 1 kan det andre elementet derfor velges på 2x3=6 ulike måter. I dybde 2 er matrisedimensjonen redusert til 1x2, siden to elementer allerede er valgt med respektive rader og kolonner fjernet. Det siste elementet kan derfor velges på 1x2=2 ulike måter for hver av de 12*6=72 matrisene i dybde 2. Vi har altså et samlet antall på 12*6*2=144 måter å velge løsninger på i en 3x4 matrise.

Vi observerer at *dynamisk programmering* ikke er anvenbar, siden den rekursive oppdelingen i delproblemer ikke er av overlappende natur. En optimal løsning på dette problemet må undersøke/beregne hele rekursjonstreet, og for en n x m matrise med $n \leq m$ kan beregningskompleksiteten beskrives på følgende måte:

$$\begin{aligned} & nm * (n-1)(m-1) * (n-2)(m-2) * \cdots * 2(m-n+2) * 1(m-n+1) \\ & = (n * (n-1) * (n-2) * \cdots * 1) * (m * (m-1) * (m-2) * \cdots * (m-n+1)) \\ & = n! * \frac{m!}{(m-n)!} \end{aligned}$$



Figur 6.4: Søkerom for 3x4 matrise

Vi merker oss at løsningen krever stor beregningskapasitet selv for relativt små matrisedimensjoner. Tabell 9.4 viser noen kjøretider med en Java-implementasjon (se elektronisk vedlegg) av denne optimale mappingmetoden (denne viser også kjøretider for en grådighetsmetode som beskrives nedenfor). Datamaskinen som ble brukt var en 3 GHz Pentium 4 prosessor.

Siden vi skal mappe alle bildene i treningssettet opp mot hverandre, ser vi av effektivitetsmessige hensyn at en "optimal mapping"-løsning bare kan anvendes for bildepar med relativt få antall segmenterte objekter (for eksempel med $n \leq 5$ og $m \leq 5$). For å oppnå redusert kjøretid av større dimensjoner blir vi nødt til å gi opp kravet om en garantert optimal løsning. En naturlig approksimasjon er å bruke en grådighetsalgoritme, som vist i Algoritme 4. Denne velger en samlet løsning basert på de delløsninger, som til enhver tid ser ut til å være mest gunstig i øyeblikket. Linje 2 instansierer M som en matrise av 0'ere med samme dimensjon som sammenlikningsmatrisen (inputmatrisen) A. Linje 3 instansierer S som en tom liste. Linje 4-6 sorterer kolonnene i A's rader etter stigende verdi av kolonneinholdet. Hver sortert rad representeres som en lenket liste av parene (kolonneverdi/kolonne), og legges til i S. Merk at kolonne betegner kolonnens posisjon i A (altså før sortering fant sted). Det første elementet i radens lenkede liste er et enkeltstående element (ikke par av kolonneverdi/kolonne), som korresponderer til radens posisjon i A. Linje 7-19 itererer gjennom radene i S og trekker ut den minste frontverdien den finner. Metodene *frontVerdi* og

frontKolonne returnerer henholdsvis kolonneverdien og kolonneposisjon til første par i S sin rad j. Den aktuelle raden r, som inneholder den minste verdien i S, fjernes fra S i linje 16. Linje 17 fjerner alle referanser i S, som refererer til kolonnen som ble funnet å inneholde det minste frontelementet. Dette gjøres ved å dra ut paret fra den lenkede listen og koble sammen de to resulterende delene, slik at lenket liste strukturen opprettholdes. Linje 18 markerer posisjonen som mapper objektene i matrisen M. Metoden *radPosisjon* returnerer første element til aktuell rad, som betegner radens posisjon i matrise A. Linje 4-6 kjører på $O(n\,m\,lg(m))$, da sorteringen av hver rad i M gjøres på $O(m\,lg(m))$ ved for eksempel *quicksort* [10]. Linje 7-15 kjører på $O(n^2)$ når linje 17 er implementert som pekerhåndtering med tidskompleksitet lik $O(1)$. Siden $n \leq m$ kan den samlede tidskompleksiteten skrives som $O(m^2\,lg(m))$. Tabell 6.4, 6.5, 6.6 og 6.7 illustrerer grådighetsalgoritmen med et eksempel.

```
1:  procedure MATRISEMAPPING(A)
2:      instansier matrise M
3:      instansier liste S
4:      for all rader r ∈ A do
5:          S.add(sorterRad(A[r]))
6:      end for
7:      while |S| > 0 do
8:          instansier r = null, k = null, min = ∞
9:          for j = 1..|S| do
10:             if frontVerdi(S_j) < min then
11:                 min = frontVerdi(S_j)
12:                 k = frontKolonne(S_j)
13:                 r = j
14:             end if
15:         end for
16:         S = S − S_r
17:         S = oppdaterPekere(S, k)
18:         M[radPosisjon(S_r)][k] = 1
19:     end while
20:     return M
21: end procedure
```

Algoritme 4: Grådighetsalgoritme for matrisemapping

Tabell 6.6 viser eksekveringen av linje 17-19 i grådighetsalgoritmens pseudokode, og vi ser av resultatene i tabell 6.7 at objekt 1 i bilde 1 mappes med objekt 4 i objekt 2 osv. Sorteringsfasens resultat (*matrisefremstilling av korresponderende objekter*) utgjøres av tilsvarende tabeller for alle bildepar i treningssettet.

2	9	6	2	2
7	4	0	3	9
1	0	6	3	5
1	6	3	7	5
9	4	6	8	3

Tabell 6.4: Inputmatrise til grådighetsmapping

1	2/4	2/5	2/1	6/3	9/2
2	0/3	3/4	4/2	7/1	9/5
3	0/2	1/1	3/4	5/5	6/3
4	1/1	3/3	5/5	6/2	7/4
5	3/5	4/2	6/3	8/4	9/1

Tabell 6.5: Sortert matrise

Den optimale løsningen (eksponentiell tid) kan sies å være en "skyte spurv med kanon"-løsning, da grådighetsløsningen (polynomisk tid) i svært mange tilfeller også gir optimal løsning. Dette løsningsforslaget velger derfor å bruke grådighetsstrategien. I tillegg til pseudokoden som er gitt i algoritme 4 kommer restleddet for dimensjoner der $m > n$, der man må mappe hvert av de $m - n$ objekter i bilde 2 til beste objekt i bilde 1. Dette gjøres på en rett frem måte med kjørertiden $O((m - n)\, n)$.

S=1,2,3,4,5	min=0	M(2,3)=1
S=1,3,4,5	min=0	M(3,2)=1
S=1,4,5	min=1	M(4,1)=1
S=1,5	min=2	M(1,4)=1
S=5	min=3	M(5,5)=1

Tabell 6.6: Eksekvering

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

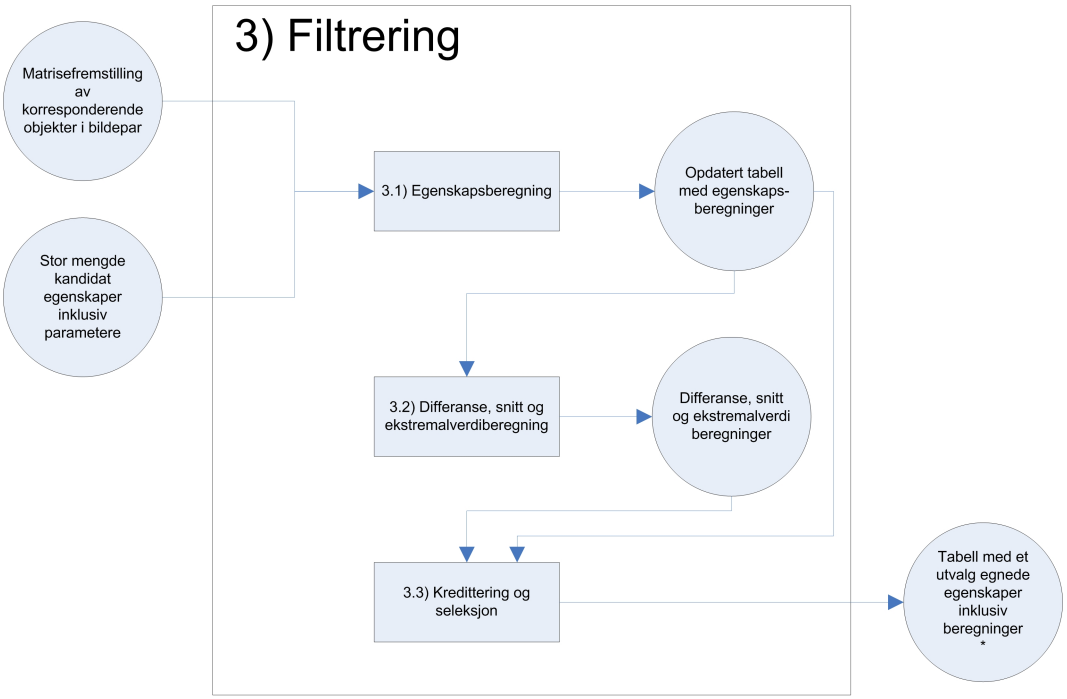
Tabell 6.7: Resultat

Dimensjon	Optimal mapping	Grådighetsmapping
5 x 5	16 ms	15 ms
6 x 7	1 485 ms	16 ms
6 x 8	3 985 ms	15 ms
6 x 10	17 453 ms	15 ms
7 x 7	22 406 ms	16 ms

Tabell 6.8: Kjøretider for optimal og grådighetsmapping

6.1.3 Filtrering

Filtreringsmodulen leter frem et egnet utvalg av egenskapsbeskrivelser for det aktuelle inpseksjonsproblemet. Dette gjøres på bakgrunn av objektmappingen fra sorteringsmodulen og en stor mengde kandidater av egenskapsbeskrivelser, samt faste terkselverdiparametere for disse. Det er viktig at egenskapskandidatene gir egenskapsbeskrivelser som kan dekke alle mulige former for objekter på en god måte. Dette kan medføre krav om hundrevis av egenskapskandidater, for eksempel statistiske momenter, gjennomsnittsverdi, hovedkomponentanalyse, omkrets og mange andre (se egenskapsbeskrivelse kapittel 3). Figur 6.5 illustrerer filtreringsmodulen og dens delprosesser.



Figur 6.5: Filtrering

Filtreringsmodulen tar tak i de objektene som er mappet i bildeparene fra treningssettet, og beregner alle egenskapskandidatbeskrivelsene for disse. Resultatene registreres i tabell på samme måte som for det lille egenskapsutvalget, som illustrert i tabell 6.2. Videre følger differanseberegninger av kandidategenskapsbeskrivelsene i henhold til objektmappingen, med påfølgende snitt og ekstremalverdiberegninger av disse. Algoritme 5 beregner snittdifferansen og maksimaldifferansen for hver kandidategenskap ved å betrakte godkjente bilder seg imellom (homogene bildepar). I pseudokoden symboliserer H homogene bildepar, og A betegner alle mulige H fra treningssettet. Algoritme 6 beregner snittdifferansen for hver kandidategenskap ved å be-

trakte godkjente og underkjente bilder (inhomogene bildepar). I pseudokoden symboliserer I inhomogene bildepar, og B betegner alle mulige I fra treningssettet. Vi har altså følgende definisjoner:

- $\bar{H}(F)$ er gjennomsnittlig differanse mellom godkjente bilders objekter seg imellom for egenskap F.
- $H_{max}(F)$ er maksimal differanse mellom godkjente bilders objekter for egenskap F.
- $\bar{I}(F)$ er gjennomsnittlig differanse mellom godkjente og underkjente bilders objekter for egenskap F.

```
1:  procedure KalkulerHomogen
2:      for all egenskapsbeskrivelser F do
3:          H_max(F) = −∞
4:          H̄(F) = 0
5:          i = 0
6:          for all bildepar (B1, B2) ∈ A do
7:              for all objektmappinger av (B1, B2) do
8:                  i = i + 1
9:                  diff = F(B1.obj) − F(B2.obj)
10:                 H̄(F) = H̄(F) + diff
11:                 if (diff > H_max(F)) then
12:                     H_max(F) = diff
13:                 end if
14:             end for
15:         end for
16:         H̄(F) = H̄(F) / i
17:     end for
18: end procedure
```

Algoritme 5: Gjennomsnittlig og maksimal differanseberegning mellom godkjente bilder seg imellom for hver egenskapskandidat

Vi merker oss at $\bar{H}(F)$ og $H_{max}(F)$ knyttes opp til hovedkrav 1 (små forskjeller i egenskapsbeskrivelser mellom godkjente bilders objekter seg imellom), og $\bar{I}(F)$ opp til hovedkrav 2 (store forskjeller i egenskapsbeskrivelser mellom godkjente og underkjente bilders objekter). Teoretisk kan vi tenke oss at det ville være gunstig med en parameter til, som kan måle minimalt avvik mellom godkjente og underkjente bilders objekter for egenskap F. Tanken er at denne $I_{min}(F)$-parameteren kan brukes til å avgjøre om egenskapskandidaten F ikke skiller tilstrekkelig på godkjente og underkjente bilder. Ved praktisk testing viste det seg derimot at $I_{min}(F)$ resulterte i verdien 0 for alle testsett. Det eksisterte med andre ord alltid to regioner R_G og R_U, som

```
1:  procedure KalkulerInhomogen
2:      for all egenskapsbeskrivelser F do
3:          Ī(F) = 0
4:          i = 0
5:          for all bildepar (B1, B2) ∈ B do
6:              for all objektmappinger av (B1, B2) do
7:                  i = i + 1
8:                  diff = F(B1.obj) − F(B2.obj)
9:                  Ī(F) = Ī(F) + diff
10:             end for
11:         end for
12:         Ī(F) = Ī(F) / i
13:     end for
14: end procedure
```

Algoritme 6: Gjennomsnittlig differanseberegning mellom godkjente og underkjente bilder for hver egenskapskandidat

"nullet" ut $I_{min}(F)$-verdien for hver egenskapskandidat F. Her betegner R_G er en region fra bildeparets godkjente bilde, og R_U er en region fra bildeparets underkjente bilde. Med bakgrunn i disse resultatene ble beslutningen å droppe $I_{min}(F)$ fra løsningsforslaget.

De ulike egenskapskandidatene fra det store utvalget kreditteres på bakgrunn av resultatene av $H_{max}(F)$, $\bar{H}(F)$, $\bar{I}(F)$, og apriori parametere tilknyttet hver egenskapskandidat ($T_1(F)$ og $T_2(F)$). Disse apriori parameterne (vektorer) er tenkt anvendt på generelle objekter og må derfor være faste for hver sin egenskapsbeskrivelse. Det bør la seg gjøre å komme frem til rimelige verdier av disse disse ved å analysere egenskapberegninger for et tilstrekkelig stort treningssett med stor objektvariasjon.

Definisjon av første hovedkravs kriterium for egenskapskandidater (T_1) er som følger:

$$\frac{\bar{H}(F)}{H_{max}(F)} \geq T_1(F) \tag{6.4}$$

Siden $\bar{H}(F) \geq 0$, $H_{max}(F) \geq 0$ og $\bar{H}(F) \leq H_{max}(F)$ gjelder det at $0 \leq \frac{\bar{H}(F)}{H_{max}(F)} \leq 1$. Uttrykket måler hovedkrav 1, og terskelen $T_1(F)$ bestemmer kravet for akseptans. Lav $T_1(F)$ aksepterer store forskjeller, mens høy $T_1(F)$ kun aksepterer små forskjeller. For inspeksjonssystemet er det derfor viktig at denne terskelen settes passelig høy for alle egenskapskandidater.

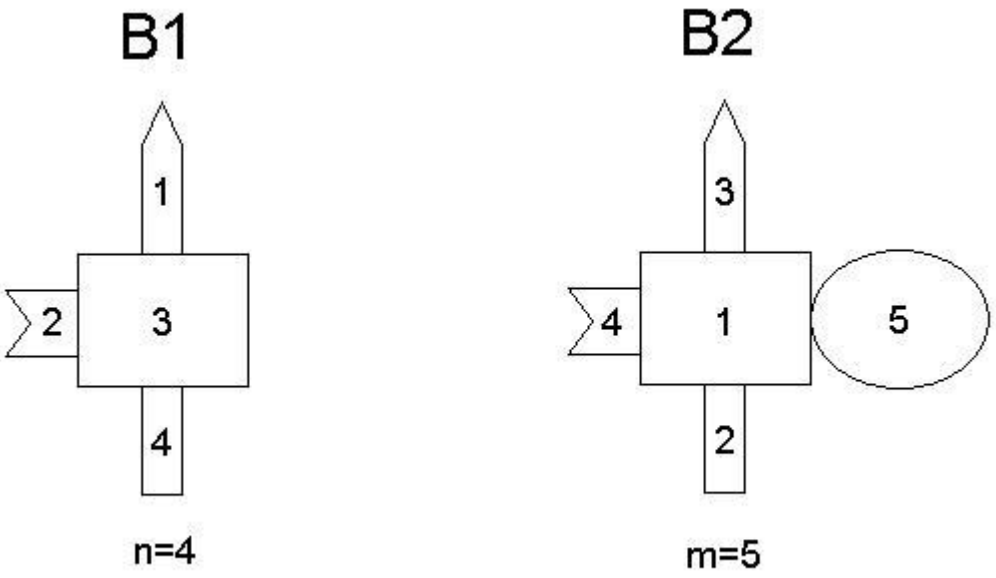
Definisjon av andre hovedkravs kriterium for egenskapskandidater (T_2) er som følger:

$$\frac{|\bar{I}(F) - \bar{H}(F)|}{\bar{I}(F)} \geq T_2(F) \tag{6.5}$$

Siden $\bar{I}(F) \geq 0$ og $\bar{H}(F) \geq 0$ gjelder det at $|\bar{I}(F) - \bar{H}(F)| \geq 0$. Uttrykkets teller måler hovedkrav 2, og nevneren tas bare med for å skalere ned resultatet. $T_2(F)$ bestemmer kravet for akseptans. Lav $T_2(F)$ aksepterer liten forskjell, i motsetning til stor $T_2(F)$ som ikke gjør dette. For inspeksjonssystemet er det derfor viktig at denne terskelen settes passelig høy for alle egenskapskandidater.

Det er ikke til å komme bort fra at segmenteringen kan resultere i et ulikt antall objekter i bildeparets to bilder (m og n, der $m \geq n$). I slike situasjoner kaller vi de $m - n$ ekstra mappingene for "annenrangs", da det ene objektet allerede er mappet til et annet objekt med lavere kostnad ut fra den lille egenskapsmengden (areal, massesenterpunkt og omsluttende rektangel). Annenrangs mappinger er viktige for at systemet skal være i stand til å skille ulikheter i bildeparene. Det er fysisk sett klart at ett objekt i bilde 1 bare kan tilsvare ett riktig objekt i bilde 2. Siden annenrangs mappinger fører til at et objekt i bilde 1 mappes med to eller flere andre objekter i bilde 2, kan maksimalt en av disse være riktig. På grunn av at systemet legger like mye vekt på alle objektmappinger i sin beregning av avvik og snittdifferanser, vil nødvendigvis "feilaktige" mappinger bidra til endrede verdier for $\bar{H}(F)$ og $\bar{I}(F)$. Det er i slike situasjoner naturlig å tenke seg at disse i all hovedsak økes, da det for de fleste egenskapsbeskrivelser vil være større forskjell mellom to generelt mindre like objekter enn for to generelt mer like objekter. Ved å følge samme tankegang ser vi dette også kan øke $H_{max}(F)$. Figur 6.6 viser et simulert eksempel på segmenteringsresultatene av et bildepar bestående av et godkjent og et underkjent bilde (henholdsvis $B1$ og $B2$). Vi ser at det godkjente bildet består av $n = 4$ objekter, og det underkjente bildet består av $m = 5$ objekter. Det eneste som skiller den defekte avbildningen fra den ikke defekte, er et ekstra segmenteringsobjekt markert som nummer 5 i $B2$. I reelle bilder kan vi for eksempel tenke oss at dette tilsvarer et defekt utspring fra den normale fassongen til en eller annen gjenstand. Tabell 6.9 viser objektmappingen av bildeparet fra figur 6.6. Her er segmenteringsobjekt nummer 5 i $B2$ markert med (*), for å symbolisere at dette danner en annenrangs mapping. Ut fra den lille egenskapsmengden er det klart at objekt nummer 3 i $B1$ er det objektet som ligner mest. Vi har altså at objekt nummer 3 i $B1$ mapper til to objekter.

Ved å gi poeng til de egenskapsbeskrivelser som innfrir kravene, kategoriserer løsningsforslaget disse egenskapskandidatene som egnet. Algoritme 7 presenterer pseudokode med dette formålet. Siden krediteringsalgoritmen bare krediterer de egenskapskandidatene som etter dette løsningsforslagets strate-



Figur 6.6: Segmentering av bildepar

gi er betraktet som egnede, er første del av egenskapsseleksjonen i stor grad gjort. Størrelsen til mengden av krediterte egenskapeskandidater kan derimot variere betydelig. Hvis det for et inspeksjonsproblem viser seg at denne mengden blir stor, og flere av egenskapskandidatene som denne inkluderer behøver vektorbeskrivelser med høy dimensjon, blir det nødvendig å redusere mengden data med hensyn på kompleksitet i maskinlæringen. En enkel strategi for å motvirke dette problemet er å velge de egenskapskandidater som krever minst vektordimensjon i sine beskrivelser blant de egnede egenskapene. For eksempel vil egenskapskandidater som bare anvender skalarverdier eller små vektordimensjoner i sine beskrivelser være svært gunstige. En annen strategi er rett og slett og gjøre et tilfeldig utvalg blant de egnede egenskapskandidatene, eventuelt i kombinasjon med førstnevnte strategi. Som utvidelse av dette løsningsforslaget kan man også tenke seg at krediteringsalgoritmen på en eller annen måte kan kreditere de ulike kandidatene med poengsummer korresponderende til hvor godt disse gjør det i forhold til parameterverdiene, med påfølgende utvalg av de n beste av disse. En annen strategi er å bruke seleksjonsmetoder som beskrevet i egenskapsuttrekking kapittel 4, med krediteringskandidatene som mulige egenskaper.

n/m	1	2	3	4	*5
1	0	0	1	0	0
2	0	0	0	1	0
3	1	0	0	0	1
4	0	1	0	0	0

Tabell 6.9: Mapping av bildepar

1: **procedure** KREDITER
2: **for all** *egenskapsbeskrivelser* F **do**
3: $poeng(F) = 0$
4: **if** (

$$\frac{\bar{H}(F)}{H_{max}(F)} \geq T_1(F)$$
$$\frac{|\bar{I}(F) - \bar{H}(F)|}{\bar{I}(F)} \geq T_2(F)$$

) **then**
5: $poeng(F)++$
6: **end if**
7: **end for**
8: **end procedure**

Algoritme 7: Kreditering av egenskaper

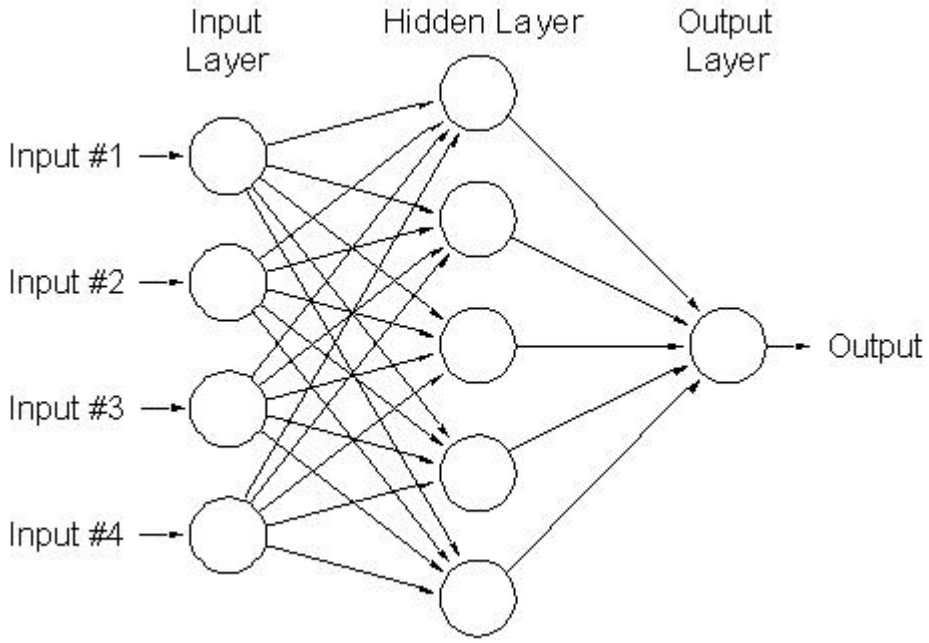
6.1.4 Maskinlæring

Nevrale nettverk [52] er velkjente maskinlæringsmetoder som blant annet er godt egnet til klassifisering. Litteraturen beskriver svært mange ulike topologier og variasjoner av slike, inklusiv deres styrker og svakheter. Disse nettene har evnen til å lære fra eksempler på en slik måte at eksplisitte klassifiseringsregler blir unødvendig. Nettene representerer dataene på sin egen måte, og kan ved riktig trening generalisere essensen i treningsdataene. Sistnevnte gjør det mulig å klassifisere uforutsette og varierende data. Det er også et viktig poeng er at ferdigtrenede nett kan eksekveres raskt i parallelle implementasjoner.

Dette løsningsforslaget velger et "feed forward" type nettverk, som ikke har noen sykler. Figur 6.7 (kopiert fra [52]) illustrerer hvordan et lite slikt nettverk kan se ut. Dette nettverket består av fire inputnoder, ett skjult lag med fem noder, og en outputnode. For å avdekke en gunstig topologi for nettet i systemet vårt kreves eksperimentering, men antallet outputnoder kan fastsettes til 1. Ved å bare ha en outputnode betegner denne klassifisering i form av outputnodens verdier ($[0.5, 1]$: godkjent, $[0, 0.5 >$: underkjent). I utgangspunktet kan løsningsforslaget også basere seg kun på ett skjult lag. Når det gjelder antall inputnoder preges disse av antall egnede egenskapskandidater samt deres vektordimensjoner, som beskrevet i filtreringsmodulens seleksjonsprosess. Samlet sett vil alle disse dataene utgjøre en inputvektor til nettverket. Løsningsforslaget innfører derfor en øvre grense for hvor stor denne inputvektoren kan være, for eksempel 200. Eksperimentering er viktig for å avgjøre størrelsen på denne terskelen. Det er verdt å merke at dette bare er en øvre grense, siden mulighetene for få egenskapskandidater med små vektordimensjoner er tilstede.

Figur 6.8 illustrerer en neuron (node) i det nevrale nettverket. Hvert utgangssignal har i tillegg til en vekt w, også en verdi $O_v \in [0, 1]$, der O_v representerer utganssignatet til noden (v). Disse verdiene vises ikke i figuren. Inngangssignalet til node i er gitt som $\sum_v W_{iv} O_v$. Utgangssignalet til node i beregnes på bakgrunn av en aktiveringsfunksjon, som avgjør styrken på utgangssignalet. Som aktiveringsfunksjon velger løsningsforslaget *sigmoid-funksjonen* [52]. Denne er gitt som $\frac{1}{1+e^{-inngangssignal}}$. Dersom denne overstiger en hvis terskelverdi noden, velges denne verdien som utgangssignal for noden. Hvis ikke gjelder utgangssignalet 0.

Læring kan gjøres ved *backpropagation* [52] metoden. Denne justerer vekter slik at differansen mellom det beregnede og riktige svaret reduseres. Ved å gjenta prosedyren tilstrekkelig mange ganger vil nettverket som regel konvergere til en tilstand der differansen og vektene ikke endrer seg betydelig. Nettverket har da "lært" den aktuelle treningsvektoren. Ved initialisering er

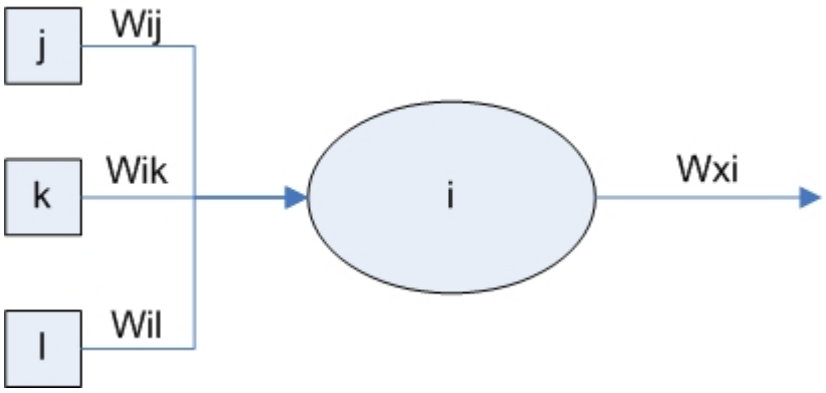


Figur 6.7: Feed forward nevralt nettverk

det vanlig å tildele vektene små og tilfeldige verdier.

6.1.5 Konfigurasjon og parameterverdier

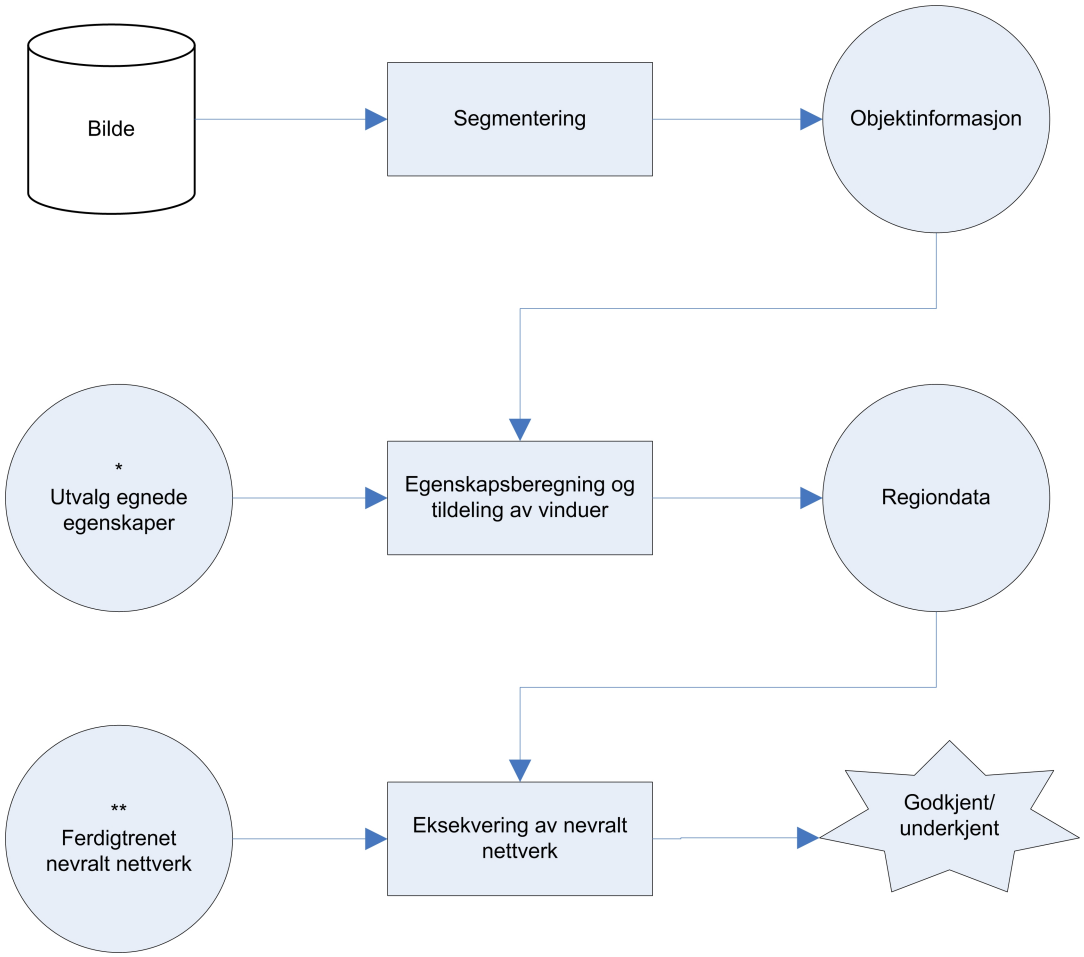
Det er opplagt at systemet krever parameterverdier både for vekter til den lille egenskapsmengden, *Richard Blake's fargesegmenteringsmetode*, vinduinndeling og krav til egenskapskandidater med mer. Optimale innstillinger av disse parameterne varierer fra bruksområde til bruksområde. På grunn av inspeksjonssystemets målsetning om generell og fleksibel anvendelse, har det vært viktig å holde antall parametere på et relativt beskjedent nivå. Tanken er at det skal gå raskt og greit å stille inn de få parameterverdiene som er nødvendig for det aktuelle bruksområdet. Dette inkluderer både segmenteringsparameterne, vinduinndeling og vekter til den lille egenskapsmengden. Når det gjelder krav til egenskapskandidater, så velger vi å generalisere disse for å holde antall variabler på et lavt nivå. En måte å estimere disse på er å la systemet kjøre gjennom flere ulike bildesett, og observere beregningene av de to kriteriumsuttrykkene for hovedkrav i avsnitt 6.1.3. Ved hjelp av gjennomsnittsberegninger og prosentavvik på disse resultatene er tanken å komme frem til akseptable parameterverdier. Det brukes med andre ord en mengde ulike bildesett til å approksimere et generelt bruksområde, for så å velge parameterverdier ut fra dette. Rapporten vil komme tilbake til konfigurasjon og parameterverdier i senere kapitler.



Figur 6.8: Neuron

6.2 Eksekveringsfase

Etter at konfigureringsfasen ("off-line") har stilt inn systemet for det aktuelle inspeksjonsproblemet, overtar eksekveringsfasen ("on-line"). Denne anvender det egnede utvalget av kandidategenskaper og det ferdigtrenede nevrale nettverket fra konfigureringsfasen. Figure 6.9 illustrerer prosesser og ressurser i eksekveringsfasen. Segmenteringen foregår på nøyaktig samme måte som konfigureringsfasen. Eksekveringsfasen har derimot ingen sorterings eller filtreringsprosess, men beregner egenskapsbeskrivelser direkte av segmenteringsresultatene, og tildeler vinduer til regioner på tilsvarende måte som konfigurasjonsfasen. Regiondataene blir så i sin tur eksekvert i det nevrale nettverket, som kort og godt forteller om bildet underkjennes eller godkjennes (dvs om bildet inneholder et defekt objekt eller ikke).



Figur 6.9: Eksekveringsfase

Kapittel 7

Konstruksjon

Dette kapitlet tar sikte på å vise oppbygning og logisk flyt i inpseksjonssystemet. Dette gjøres ved hjelp av klasse- og sekvensdiagrammer, som henholdsvis presenteres i avsnitt 7.1 og 7.2.

7.1 Klasser

Samtlige klassediagrammer under dette avsnittet er modellert ved hjelp av vektøyet *Umbrello UML modeller* (se vedlegg). Avsnitt 7.1.1 gir en overordnet presentasjon av systemets klasseoppbygning. Videre følger avsnitt 7.1.2 opp med mer detaljert informasjon om de enkelte klassene.

7.1.1 Overordnet

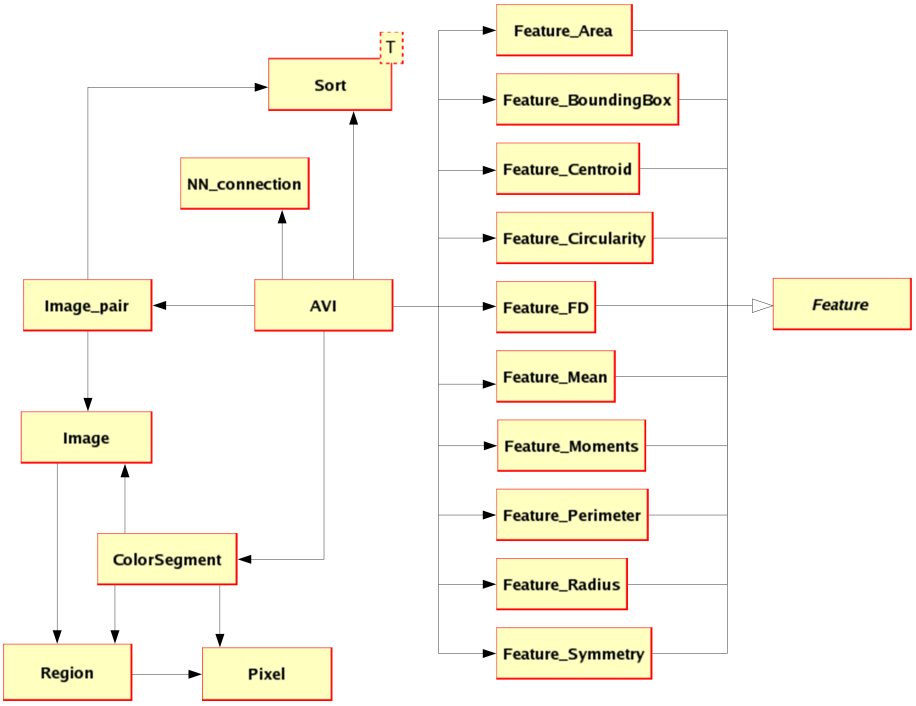
Figur 7.1 viser et overordnet klassediagram av systemet. En svart pil som går fra klasse A og peker på klasse B betyr at A kjenner klasse B. Figuren illustrer også at allle ti egenskapskandidater arver funksjonalitet fra *Feature*-klassen.

7.1.2 Detaljert

Dette avsnittet presenterer klassene med attributter og metoder. For informasjon om de enkelte attributtene og metodene henvises leseren til kodens dokumentasjon.

AVI

Figur 7.2 illustrerer hovedklassen i det automatiske visuelle inspeksjonssystemet.



Figur 7.1: Overordnet klassediagram

ColorSegment

Figur 7.3 viser oppbygningen av klassen som utfører fargesegmentering på bilder.

Feature

Figur 7.4 viser oppgbygningen av den abstrakte klassen *Feature*, som inneholder basis funksjonalitet for alle egenskapskandidater, samt egenskapskandidater som arver fra denne klassen. Egenskapskandidatene implementerer hver en egen *calculate()*-metode. Samlet kan disse oppsummeres til følgende punktliste:

1. Regioners areal.
2. Regionkonturers omsluttende rektangel.
3. Regioners massesenterpunkt.
4. Regioners rundhet/sirkularitet.
5. Regionkonturers *Fourier Descriptor*.
6. Regioners gjennomsnittlige fargeverdier.

7. Regionens *statistiske momenter*.
8. Regioners omkrets.
9. Regioners "radius" fra massesenteret til kontur.
10. Regionkonturers symmetri om x og y-aksen.

Image

Figur 7.5 viser oppgbygningen av klassen som holder rede på bildeinformasjon.

Image_pair

Figur 7.6 viser oppbygningen av klassen som holder informasjonen om bildepar.

NN_connection

Figur 7.7 viser oppbygningen av klassen som tar for seg koblinger mot det nevrale nettverket.

Pixel

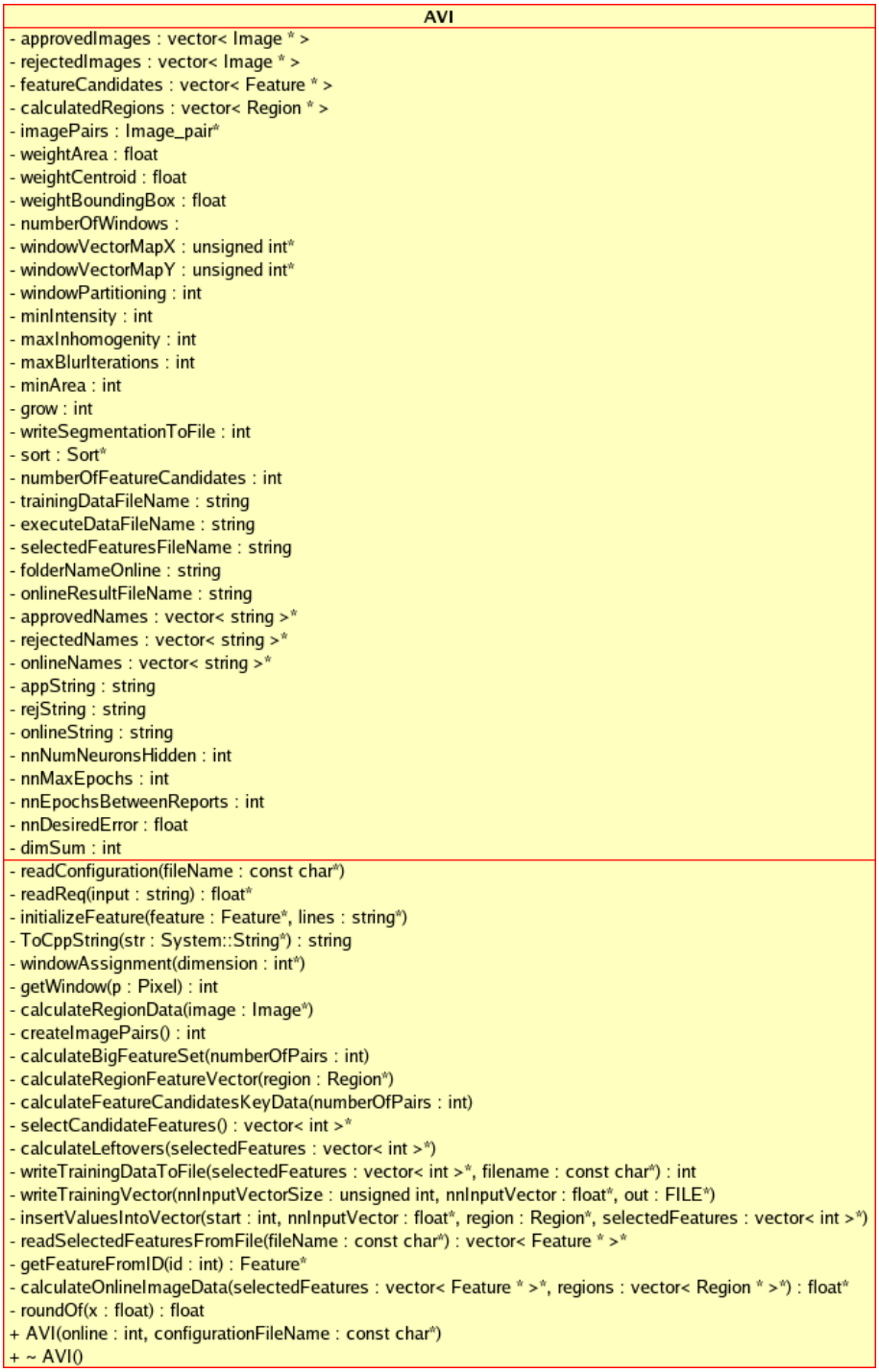
Figur 7.8 viser oppbygningen av klassen som representerer piksler i et bilde.

Region

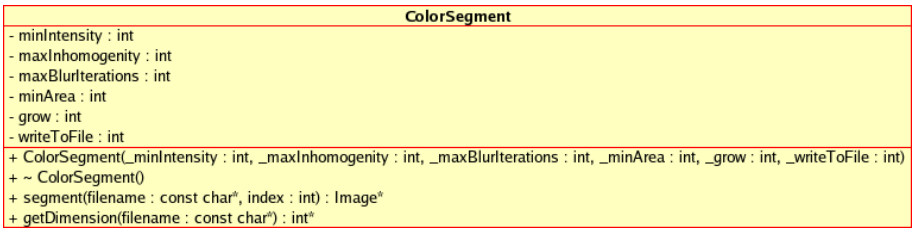
Figur 7.8 viser oppbygningen av klassen som representerer regioner.

Sort

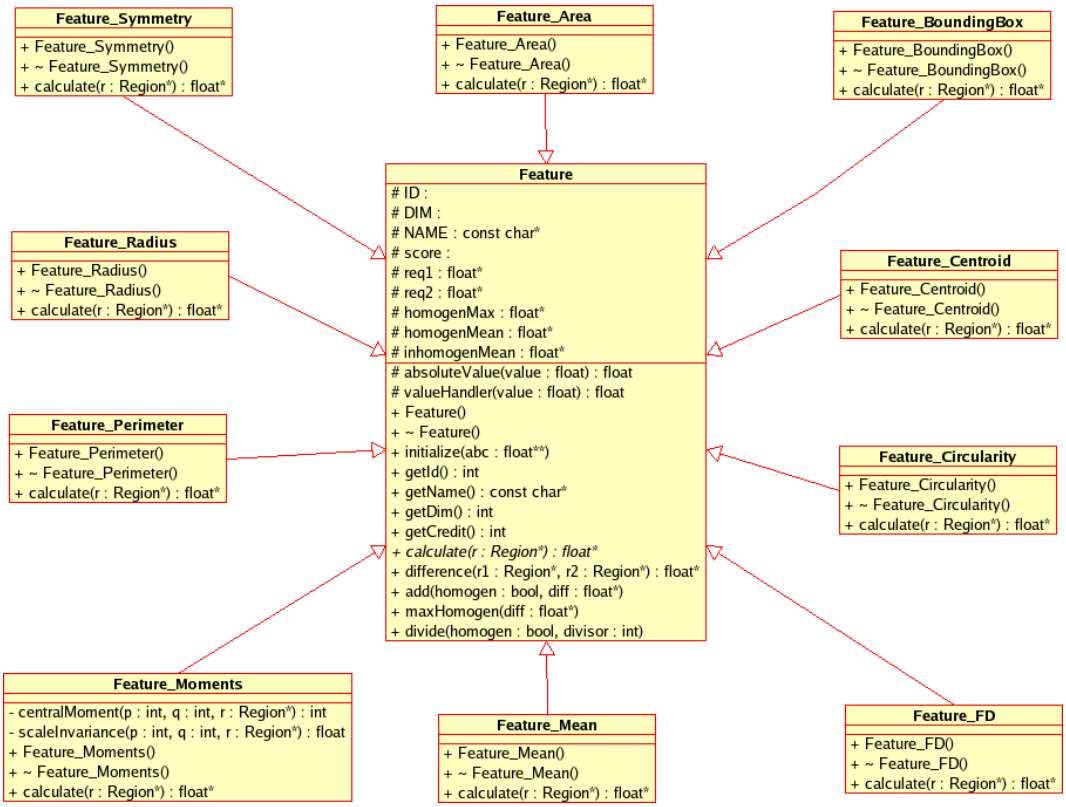
Figur 7.8 viser oppbygningen av klassen som beregner beste korrespondans/-mapping mellom regioner.



Figur 7.2: AVI-klassen

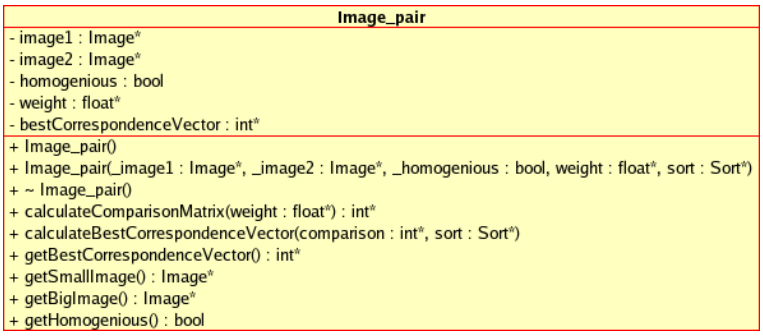


Figur 7.3: ColorSegment-klassen

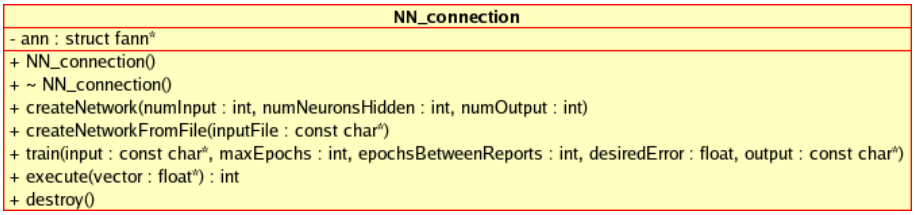


Figur 7.4: Feature-klassen og dens arvtagere

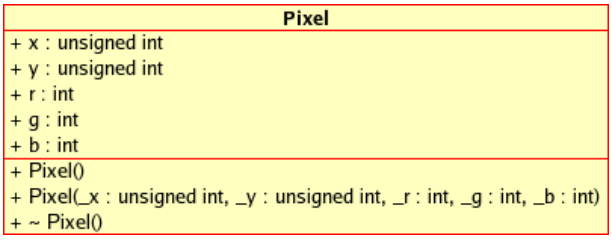
Figur 7.5: Image-klassen



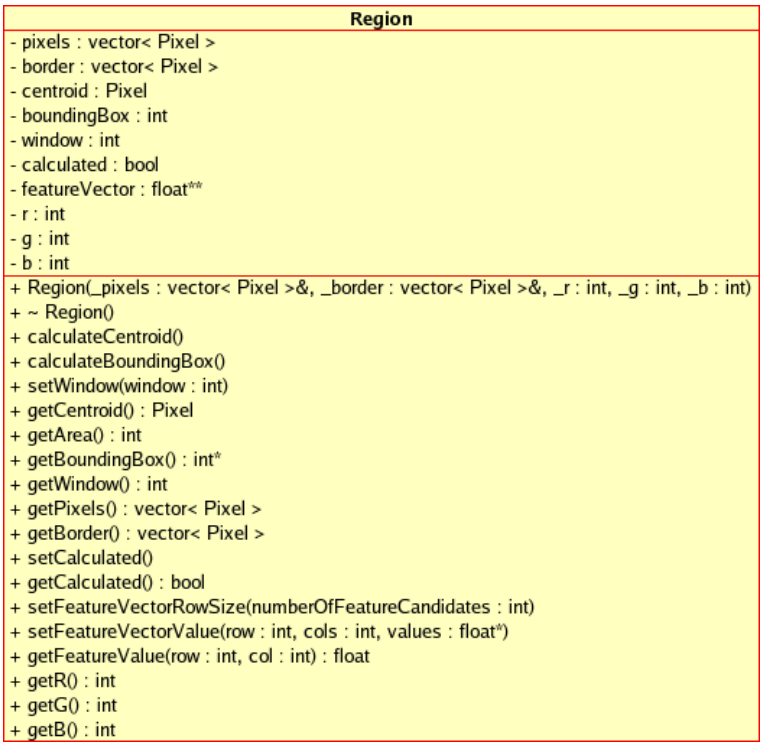
Figur 7.6: Image_Pair-klassen



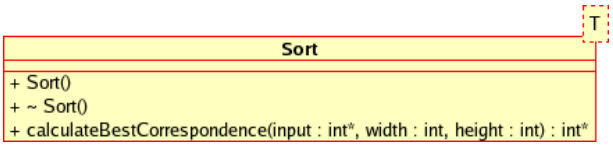
Figur 7.7: NNconnection-klassen



Figur 7.8: Pixel-klassen



Figur 7.9: Region-klassen



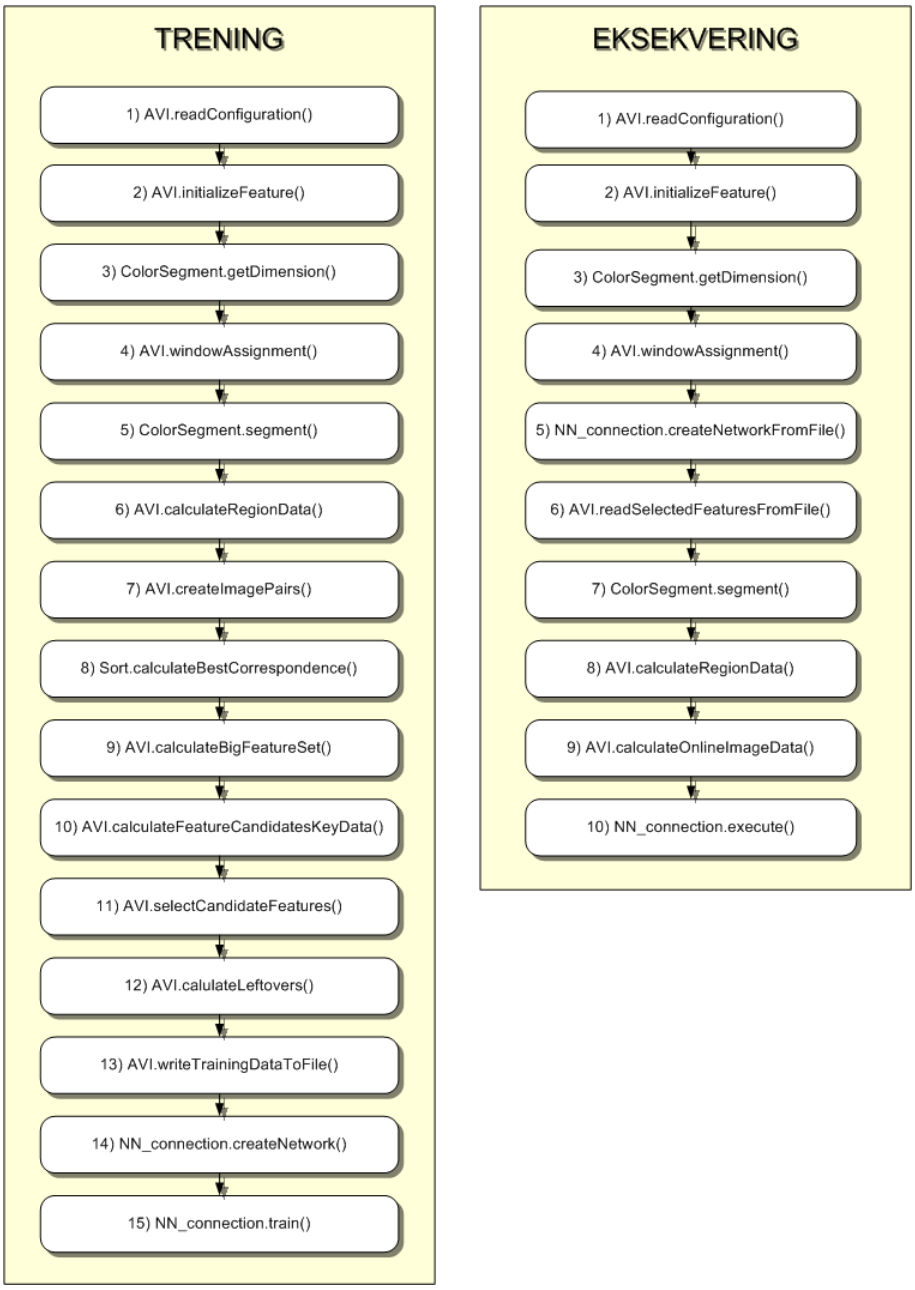
Figur 7.10: Sort-klassen

7.2 Logisk flyt

Dette avsnittet beskriver inspeksjonssystemets logiske flyt og oppførsel. Det er viktig å merke seg at diagrammene kun illustrerer de viktigste sekvensene i inpseksjonssystemet. Avsnitt 7.2.1 presenterer en "rød tråd" i inspeksjonssystemet. Her er det en målsetning å formidle på et mer overordnet nivå hvordan først og fremst hovedklassen *AVI* løser inspeksjonsproblemet. Videre følger sekvensdiagrammer i avsnitt 7.2.2. Disse setter fokus på når og hvordan de ulike klassene kommuniserer med hverandre.

7.2.1 Metodeflyt

Figur 7.11 viser inspeksjonssystemets hovedmetoder og deres rekkefølge under kjøring. Figuren er laget i *Microsoft Visio* (se vedlegg). Venstre og høyre side viser henholdsvis oppførsel i trenings- og eksekveringsfasen. De neste to avsnittene følger opp figuren med supplerende informasjon.



Figur 7.11: Inspeksjonssystemets røde tråd

Treningsfase

Systemet starter ved å lese inn samtlige konfigurasjonsinnstillinger i *AVI.readConfiguration()*. Videre initialiseres hver egenskapskandidat med *AVI.initializeFeature()*, slik at systemet blir bevisst på at disse eksisterer. *ColorSegment.getDimension()* finner aktuell dimensjon (bredde og høyde) for gjeldende bildesett, som er helt essensiell for den videre bildeprosesseringen. *AVI.windowAssignment()* deler bildet inn i vinduer i henhold til parameterverdi anitt i konfigurasjonen. *ColorSegment.segment()* gjør fargesegmentering på bilder, og *AVI.calculateRegionData()* beregner den lille egenskapsmengden for de segmenterte regionene. Videre struktureres bildene i objekter av homogene (godkjent,godkjent) og inhomogene (godkjent,undekjent) bildepar i *AVI.createImagePairs()*. *Sort.calculateBestCorresondence()* beregner beste mapping mellom regioner i bildeparets to bilder. *AVI.calculateBigFeatureSet()* tar så over med å beregne alle egenskapskandidater på de regioner som er mappet i bildeparene. Neste steg beregner nøkkeldata for hver egenskapskandidat i *AVI.calculateFeatureCandidatesKeyData()*. Dette gjøres på bakgrunn av alle bilder, og innebærer å beregne maskimal og gjennomsnittlig forskjell mellom godkjente bilders regioner seg imellom, og gjennomsnittlig forskjell mellom godkjente og underkjente bilders regioner. *AVI.selectCandidatesFeature()* følger så opp ved å betrakte resultatene, og velge ut de egnede egenskapskandidatene. For de regioner som ikke er mappet i bildeparene har systemet heller ikke beregnet noen egenskapskandidater. Dette er problematisk da alle segmenterte regioner må være med å bidra til inputvektoren for det nevrale nettverket under trening såvel som ved "online" eksekvering. *AVI.calculateLeftovers()* beregner derfor utvalgte egenskapskandidater for disse regionene. Fordelen med denne oppbygningen er å spare systemet for beregingen av uegnede enskapskandidater på regioner som ikke mappes. *AVI.writeTrainingDataToFile()* skriver treningsdata til fil. Disse dataene består av inputvektorene sammen med informasjon som angir om bildene er godjent eller underkjent. *NN_connection.createNetwork()* kaller funksjonalitet i FANN (se vedlegg), som lager et nevralt nettverk i henhold til parametere gitt i konfigurasjonsfilen. *NN_connection.train()* trener til slutt opp det nevrale nettverket med aktuelle treningsdata. Metoden sørger også for å kalle funksjonalitet i FANN som lagrer det nevrale nettverket og dets vekter i en fil, som brukes i eksekveringsfasen.

Eksekveringsfase

Punktene 1-4 er helt identisk med treningsfasen. Neste steg er å opprette det ferdigtrenede nevrale nettverket fra fil. Dette gjøres i metoden *NN_connection.createNetworkFromFile()*. Videre leser og oppretter *AVI.readSelectedFeaturesFromFile*-metoden de egenskapskandidatene som ble funnet egnede i treningsfasen. *ColorSegment.segment()* segmenterer bildene,

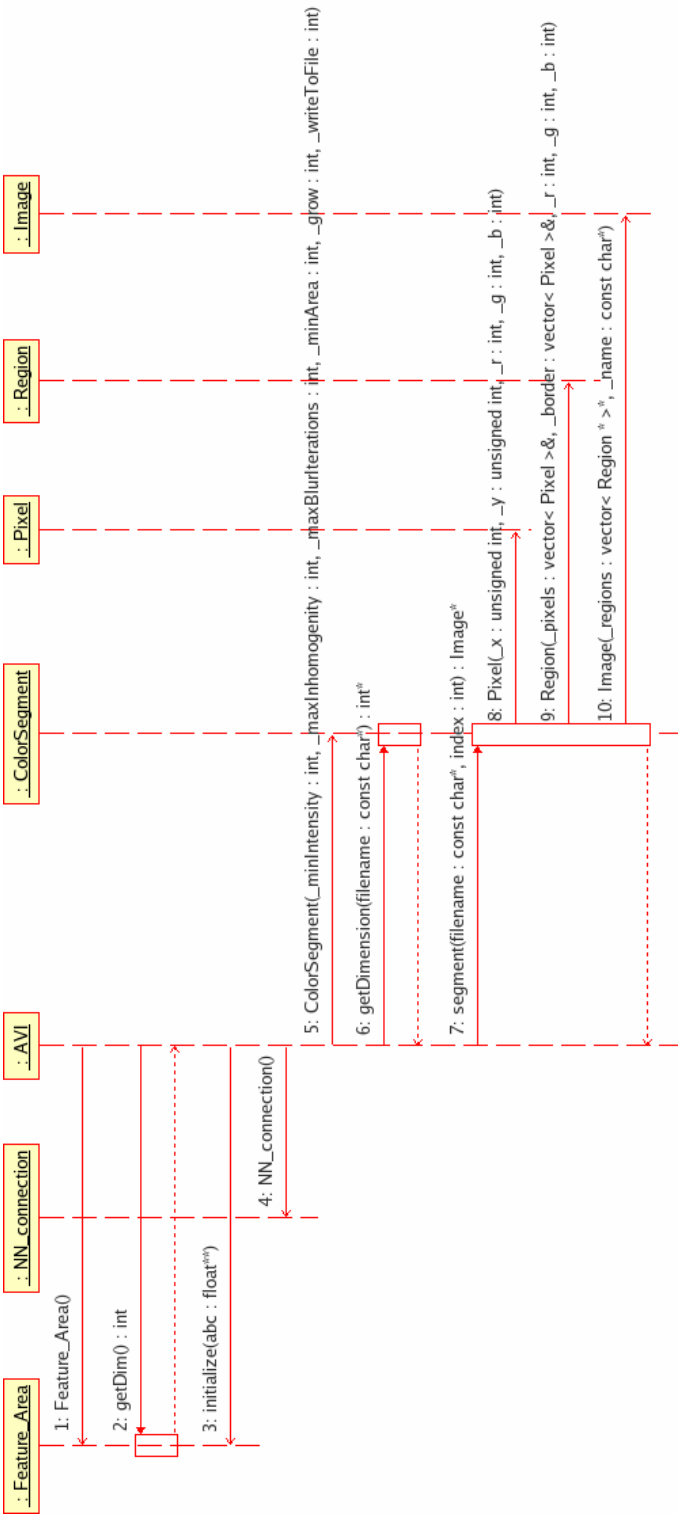
og *AVI.calculateRegionData()* beregner den lille egenskapsmengden for de segmenterte regionene uavhengig om disse er funnet egnede eller ikke. *AVI.-calculateOnlineImageData()* beregner dernest resterende egnede egenskapskandidater for regionene, og inputvektorer til det nevrale nettverket. *NN-_ connection.execute()* klassifiserer til slutt de aktuelle bildene.

7.2.2 Sekvensdiagrammer

Dette avsnittet illustrerer samhandling mellom klassene i inspeksjonssystemet. Alle diagrammer er tegnet ved hjelp av *Umbrello UML modeller*. Det er et viktig poeng at diagrammene bare presenterer metodekall fra klasse A til klasse B, der $A \neq B$. For presentasjonens skyld deles trenings- og eksekveringsfasen henholdsvis inn i seks og tre sekvensdiagrammer.

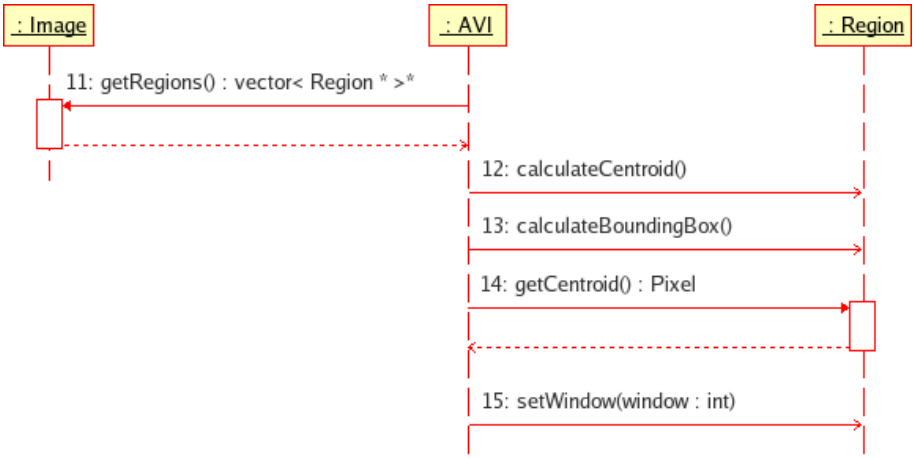
Treningsfase

Figur 7.12 viser gangen i systemets treningsfase fra oppstart til og med segmentering. Disse tilsvarer punktene 1-5 i figur 7.11. Punkt 1 og 2 oppretter og henter ut vektordimensjonen til egenskapskandidaten *areal* (*Feature_ Area*). Tilsvarende gjelder også for resterende egenskapskandidater, som ikke vises i figuren. Punkt 3 initialiserer egenskapskandidaten, og punkt 4 oppretter et objekt som representerer det nevrale nettverket. Punkt 5 oppretter et *ColorSegment*-objekt, som brukes til å hente bildedimensjoner i punkt 6 og segmentering i punkt 7. Bildedimensjonene er vesentlig informasjon for inndeling i vinduer og iterasjoner over bildet. Segmenteringsmetoden i *ColorSegment* oppretter pikselobjekter i punkt 8, regionobjekter i punkt 9 og et bildeobjekt i punkt 10.



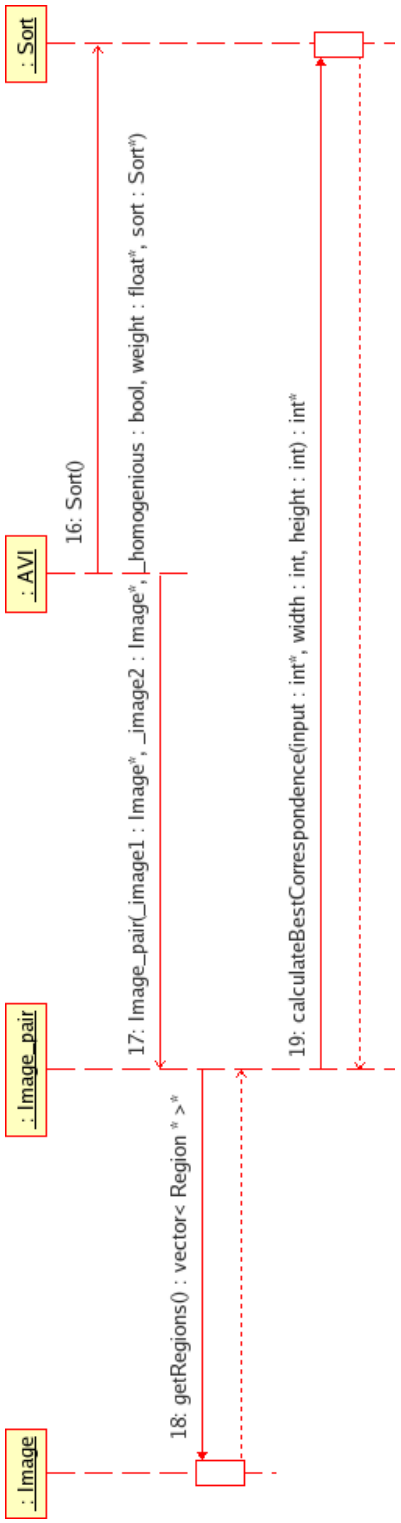
Figur 7.12: Treningsfase, steg 1

Figur 7.13 illustrerer sekvensen som tilsvarer punkt 6 (beregning av lille egenskapsmengde) i figur 7.11. Punkt 11 henter ut regioninformasjon fra et bildeobjekt. Punkt 12 og 13 beregner henholdsvis massesenter og omsluttende rektangel til disse regionene (arealet er allerede kjent som antall piksler i regionen). Punkt 14 henter ut massesenterinformasjon for en region, som er nødvendig for at systemet i punkt 15 skal være i stand til å sette hvilket vindu en region tilhører.



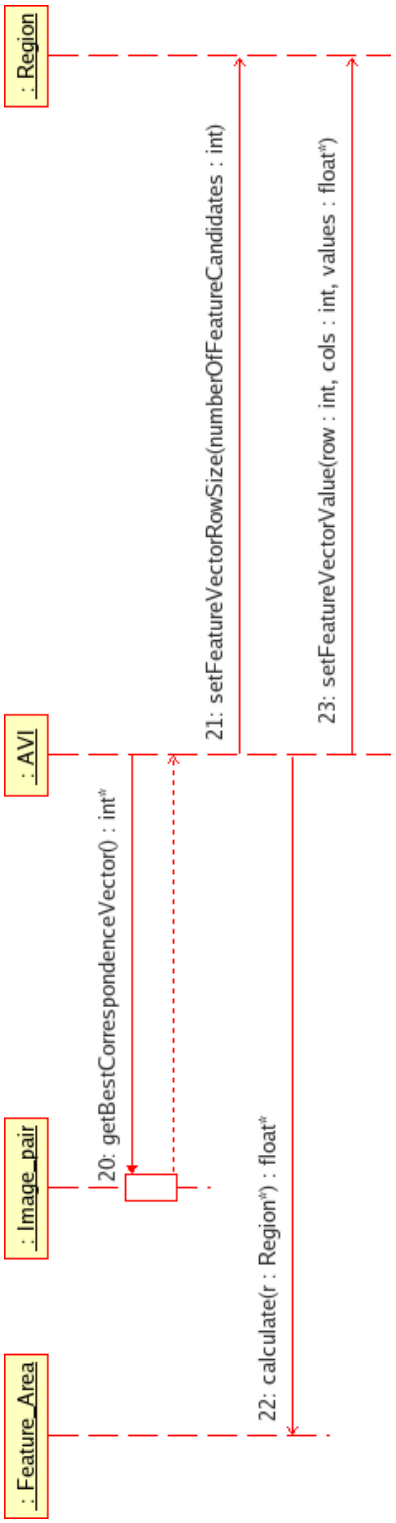
Figur 7.13: Treningsfase - steg 2

Figur 7.14 viser den delen av systemet som oppretter bildepar og mapper regioner internt i disse. Disse tilsvarer punktene 7 og 8 i figur 7.11. Punkt 16 oppretter et objekt med sorteringsfunksjonalitet. Punkt 17 oppretter bildeparobjekter av bildene. Hvert bildeparobjekt aksesserer samtlige av bildeparets regioner i punkt 18, før sorteringsobjekter i punkt 19 beregner beste korrespondans/mapping mellom disse.



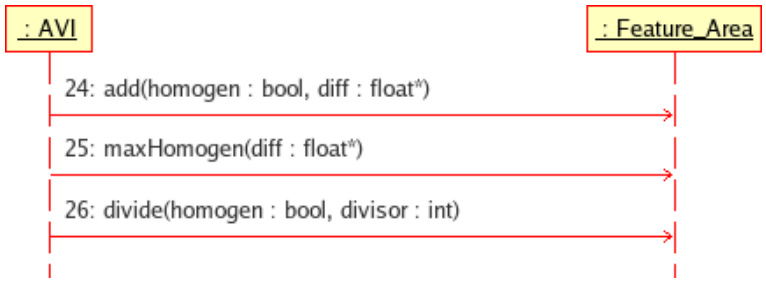
Figur 7.14: Treningsfase - steg 3

Figur 7.15 beregner det store utvalget av egenskapskandidater, og tilsvarer punkt 9 i figur 7.11. Punkt 20 henter ut informasjon om hvilke regioner som er mappet i bildeparet. Punkt 21 setter vektordimensjonen, som holder av rom for alle egenskapskandidatverdier, til slike regioner. Punkt 22 beregner verdier for en aktuell egenskapskandidat på denne regionen, og punkt 23 følger opp med å registrere resultatene på riktig posisjon i regionens vektor.



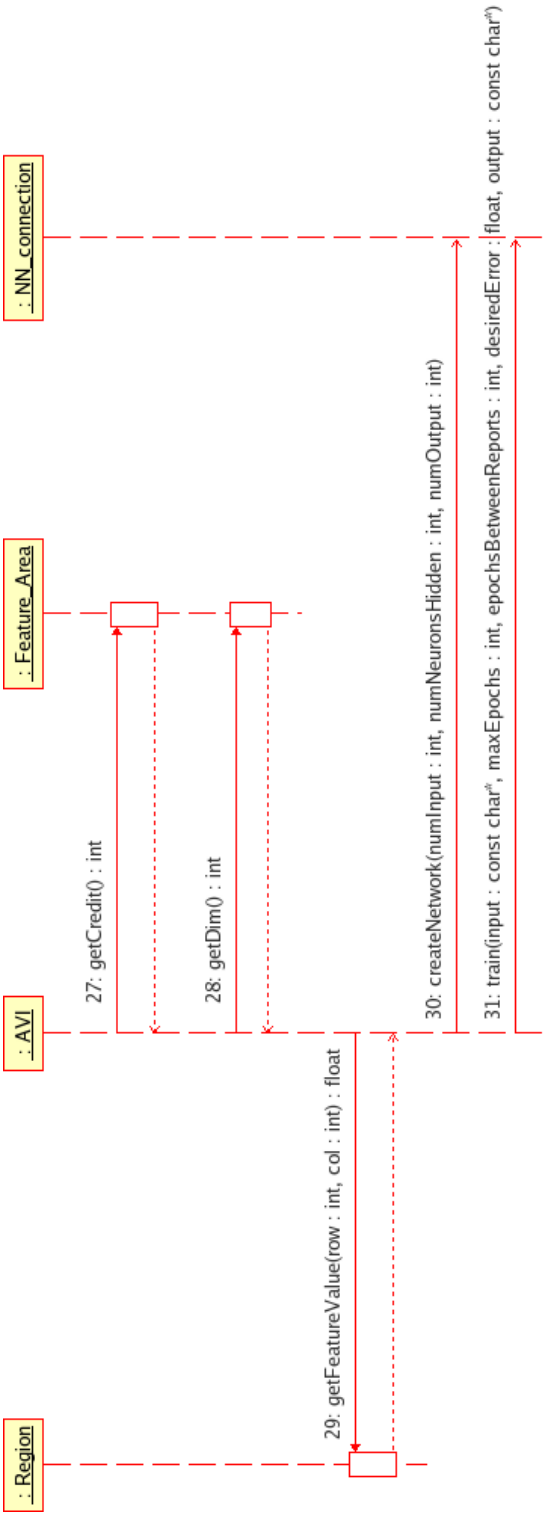
Figur 7.15: Treningsfase - steg 4

Figur 7.16 tar for seg beregningen av nøkkeldata for de ulike egenskapskandidatene. Sekvensen tilsvarer punkt 10 i figur 7.11. Her vises kun areal frem som egenskapskandidat, men tilsvarende gjelder også for alle de andre egenskapskandidatene. Punkt 24 legger til den aktuelle egenskapskandidatens differanse mellom to mappede regioner i egenskapskandidatobjektet. Tilsvarende gjøres i punkt 25 dersom de mappede regioner er et homogent (godkjent,godkjent) bildepar, og differansen er større enn tidligere. Punkt 26 følger opp med å dividere på antall regionpar som har bidratt til summen. Dette gjøres for å beregne egenskapskandidatens gjennomsnittlige differanseverdier for mappede regioner.



Figur 7.16: Treningsfase - steg 5

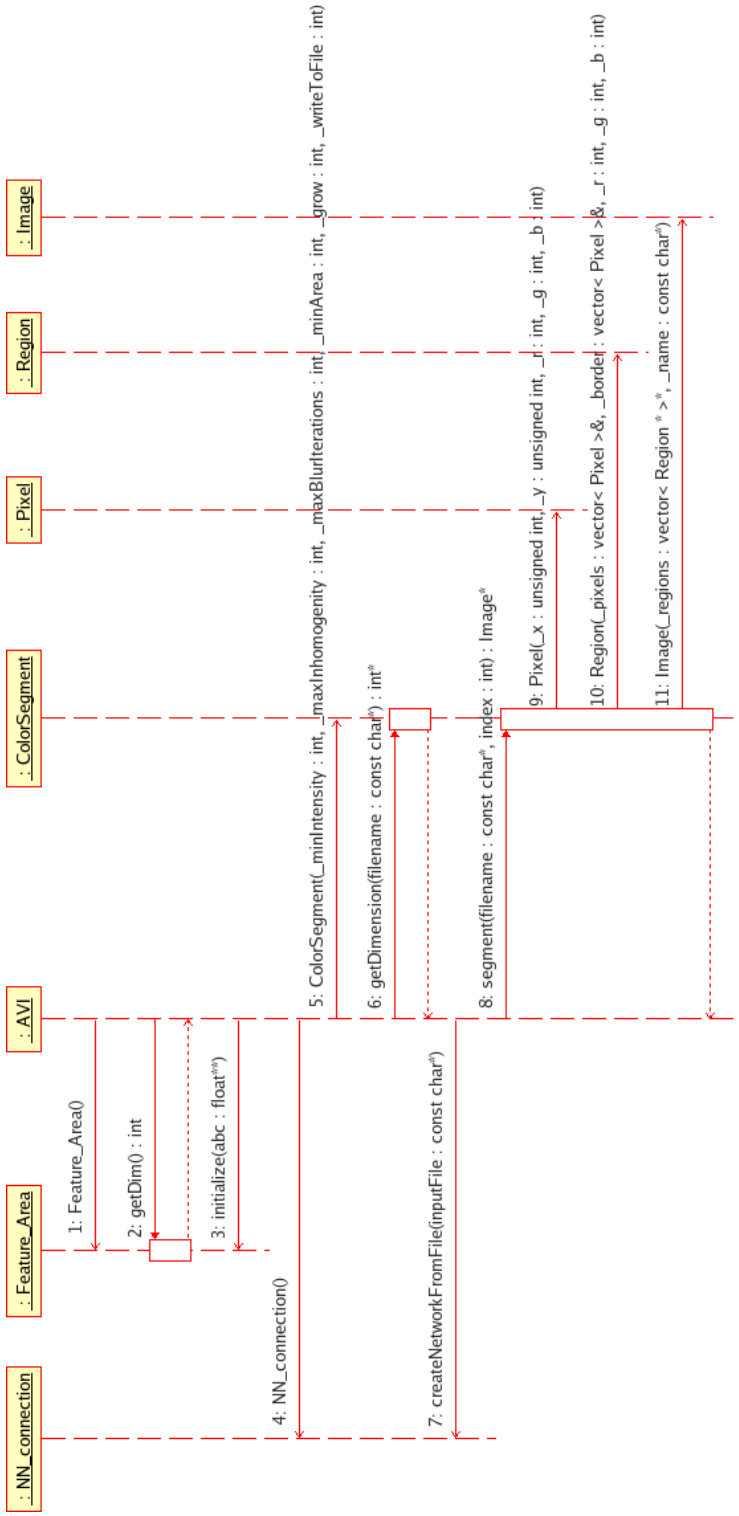
Figur 7.17 illustrerer siste del av systemets treningsfase. Sekvensene tilsvarer punkt 11 og utover (med unntak av punkt 12) i figur 7.11. Sekvensen starter med punkt 27, som henter ut informasjon angående prestaskjonen til egenskapskandidater (igjen brukes areal som et representativt eksempel). Denne informasjonen brukes til å avgjøre hvilke egenskapskandidater som er egnet. Punkt 28 og 29 henter dimensjon og verdier til egenskapskandidater. Disse treningsdataene skrives så til fil. Punkt 30 oppretter et objekt som representerer det nevrale nettverket, og punkt 31 trener opp dette med aktuelle treningsdata.



Figur 7.17: Treningsfase - steg 6

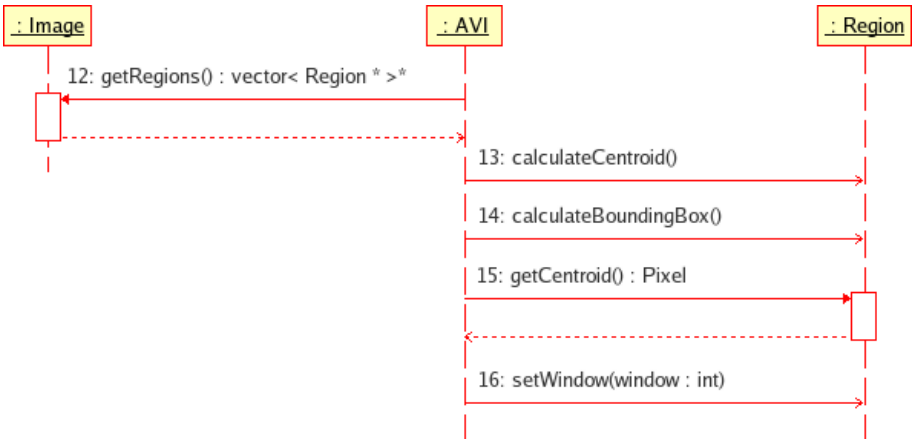
Eksekveringsfase

Figur 7.18 viser sekvenser i systemets eksekveringsfase. Disse tilsvarer punktene 1-7 i figur 7.11. Punkt 1, 2 og 3 oppretter, henter dimensjonen til og initialiserer et egenskapskandidatobjekt (her vises areal som eksempel). Punkt 4 instansierer et objekt med koblinger mot funksjonalitet i nevrale nettverk, og punkt 5 instansierer et objekt med fargesegmenteringsfunksjonalitet. Punkt 6 henter ut bildedimensjoner, som er vesentlig informasjon for å kunne dele inn i vinduer og iterere over bildet. Punkt 7 oppretter et ferdigtrenet nevralt nettverk ved å lese inn opplysninger fra fil som ble lagret i konfigurasjonsfasen. Segmenteringsmetoden kalles i punkt 8. Denne følger opp med å opprette pikselobjekter i punkt 9, regionobjekter i punkt 10 og et bildeobjekt i punkt 11.



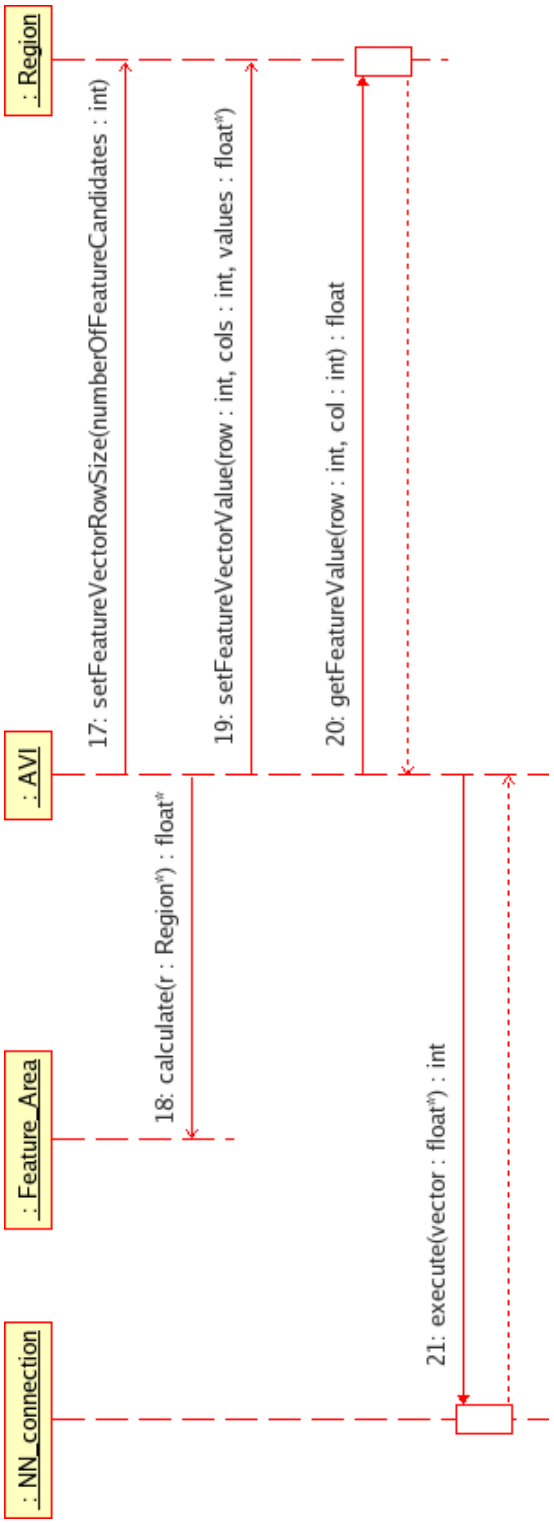
Figur 7.18: Eksekveringsfase - steg 1

Figur 7.19 illustrerer sekvensen som tilsvarer punkt 8 (beregning av lille egenskapsmengde) i figur 7.11. Punkt 12 henter ut regioninformasjon fra et bildeobjekt. Punkt 13 og 14 beregner henholdsvis massesenter og omsluttende rektangel til disse regionene (arealet er allerede kjent som antall piksler i regionen). Punkt 15 henter ut massesenterinformasjon for en region, som er nødvendig for at systemet i punkt 16 skal være i stand til å sette hvilket vindu en region tilhører, som er viktig for å bestemme posisjon i inputvektoren til det nevrale nettverket. Det er verdt å nevne at den lille egenskapsmengden ikke beregnes for at systemet skal være i stand til å mappe regioner, som i treningsfasen. Grunnen til at disse brukes her er rett og slett for å dra nytte av metodene som allerede er implementert for treningsfasen. Følgelig trenger bare den lille mengden av egenskapskandidater implementeres til å hente resultater fra *AVI*-klassen, fremfor å beregne disse selv.



Figur 7.19: Eksekveringsfase - steg 2

Figur 7.20 tar for seg den siste delen av systemets eksekveringsfase. Sekvensene tilsvarer punktene 9 og 10 i figur 7.11. Punkt 17 setter dimensjonen til regionobjektenes egenskapsberegninger i samsvar med antall egnede egenskapskandidater. Punkt 18 følger opp med å beregne aktuelle verdier for alle egnede egenskapskandidater på en region, og punkt 19 oppdaterer regionobjektet med disse verdiene. Videre henter punkt 20 ut sistnevnte resultater når disse behøves av hovedklassen. Til slutt eksekveres det nevrale nettverket med disse dataene i punkt 21.



Figur 7.20: Eksekveringsfase - steg 3

Kapittel 8

Implementasjon

Dette kapitlet beskriver implementasjonsrelatert informasjon tilknyttet noen utvalgte deler av systemet. Hensikten er å gjøre det lettere for utenforstående å forstå og ta i bruk systemet. Avsnitt 8.1 gir en oversikt over inspeksjonssystemets konfigurasjonsfil. Videre fortsetter avsnitt 8.2 med å gi pseudokode for valgt segmenteringsmetode. Avsnitt 8.3 presenterer bruk av objektorientering. Implementasjon av egenskapskandidater forklares i avsnitt 8.4. Utskrift til skjerm og fil presenteres i avsnitt 8.5, og dokumentasjon av kode beskrives i avsnitt 8.6.

8.1 Konfigurasjon

For å "fyre" i gang inspeksjonssystemet må brukeren angi to paramtere sammen med *exe*-filen, der den ene er konfigurasjonsfilen og den andre bestemmer systemets modus (treningsfase eller eksekveringsfase). All konfigurasjon må samles i en slik konfigurasjonsfil for hvert bildesett. Denne filen leses tidlig i systemets oppstart, og kan inneholde både felles innstillinger for generelle anvendelsesområder, samt spesifikke innstillinger for det aktuelle bildesettet. Dette gjør det mulig for brukere å operere med skreddersydde konfigurasjonsfiler for ulike bildesett.

Når det gjelder konfigurasjonsfilens innhold tar systemet kun høyde for innstillinger, som står til høyre for et likhetstegn. Alle andre linjer i betraktes som kommentarer. Det er også vesentlig at rekkefølgen av innstillingene i filen er fastlagt, da inspeksjonssystemet mapper linjevis fra konfigurasjonsfilen til systemvariabler. Konfigurasjonsfilen kan deles inn i 5 grupper av innstillinger. Disse presenteres i avsnittene 8.1.1 til 8.1.5, og følger systemets fastlagte rekkefølge.

8.1.1 Vekter og vinduinndeling

Følgende punktliste beskriver parametere som setter vekter for den lille egenskapsmengden og antall vinduinndelinger. Disse innstillingene må sies å være spesifikke for hvert bildesett. For eksempel vil bilder som er preget av store objekter kreve helt andre arealvekter, enn bilder som er preget av små objekter. Tilsvarende vil bilder med mange små objekter sannsynligvis kreve flere vinduinndelinger enn bilder med få og store objekter.

- Vekt areal: Bestemmer hvor mye arealet skal vektlegges i den lille egenskapsmengden.
- Vekt massesenterpunkt: Bestemmer hvor mye massesenterpunktet skal vektlegges i den lille egenskapsmengden.
- Vekt omsluttende rektangel: Bestemmer hvor mye omsluttende rektangel skal vektlegges i den lille egenskapsmengden.
- Vinduinndeling: Bestemmer hvor mange vinduer bildene skal deles inn i.

8.1.2 Segmenteringsparametere

Det er nødvendig at konfigurasjonsfilens segmenteringsparametere må settes spesifikt for hvert bildesett. Dette for viktig for å oppnå gunstige resultater, som omfatter det vesentligste fra de ulike bildesettene. Nedenfor følger en liste over disse parameterne.

- Minimal intensitet: Bestemmer minstekrav til fargeintensitet.
- Maksimal inhomogenitet: Bestemmer maksimalt tillatte inhomogenitet i regioner.
- Maksimalt antall glatteiterasjoner: Bestemmer øvre grense for antall iterasjoner som glatter bildet.
- Minimalt areal: Bestemmer nedre grense for areal med tanke på akseptable regionsegmenteringer.
- Utføre regionvekst: Bestemmer om *region growing* skal anvendes for å spise opp usegmenterte piksler eller ikke (henholdsvis 1 og 0).
- Skrive resultater til fil: Bestemmer om segmenteringsresultatene skal skrives ut til fil eller ikke (henholdsvis 1 og 0).

8.1.3 Sti og filnavn

Følgende instillinger bestemmer kataloger og filnavn for systemet. Disse kan settes generelle for alle bildesett.

- Katalog for godkjente bilder.
- Katalog for underkjente bilder.
- Katalog for ubestemte bilder (de bilder som skal klassifiseres i eksekveringsfasen).
- Klassifiseringsfil: Navn på fil som klassifiseringsresultatene skrives til.
- Treningsdata fil: Navn på fil som treningsdata skrives til.
- Eksekveringsfil: Navn på fil som representerer det ferdigtrenede nevrale nettverket.
- Egenskapsseleksjon fil: Navn på fil som inneholder egnede egenskapskandidater.

8.1.4 Parametere for nevralt nettverk

Følgende punktliste presenterer parametere som konfigurerer det nevrale nettverket. Disse er i utgangspunktet tenkt å kunne settes generelt for alle bildesett, men kan selvsagt settes spesifikt dersom bruker ønsker dette.

- Antall noder i de skjulte lagene.
- Øvre akseptable feilgrense ved trening for det nevrale nettverket.
- Maksimalt antall treningsepoker.
- Antall epoker mellom hver utskrift av statusrapport til skjerm.

8.1.5 Parametere for egenskapskandidater

Egenskapskandidatenes parametere er generelle og felles for alle bildesett. Selvsagt kan disse også settes spesifikt for hvert bildesett hvis ønskelig. Dette er vel og merke ikke vår intensjon, siden mye av generalitetskravet da går i vasken. Systemet er altså tenkt å kunne prestere godt nok uten at disse parameterne er optimalisert for hvert bildesett. Punktlisten under presenterer restende linjer av konfigurasjonsfilen med implementerte egenskapskandidater. Hver linje består av 2 parametere som setter tersklene for hovedkrav 1 og 2 (se 6.1.3).

- Statistiske momenter (7 linjer: moment 1-7)

- Fourier Descriptors (2 linjer: reell og imaginær del)
- Perimeter (1 linje)
- Areal (1 linje)
- Massesenterpunkt (2 linjer: x og y-komponent)
- Omsluttende rektangel (4 linjer: nord, øst, sør og vest)
- Sirkularitet (1 linje)
- Symmetri (1 linje)
- Radius (3 linjer: minimal, maksimal og gjennomsnittlig)
- Gjennomsnittsfarge (3 linjer: rød, grønn og blå komponent)

8.2 Segmentering

Dette avsnittet supplementerer teorien omkring *Richard Blake's fargesegmenteringsmetode* i avsnitt 2.3.5 med pseudokode. Først ut er algoritme 8, som viser hvordan systemet normaliser et bilde og eliminierer bort støyfulle lavintensitetspiksler.

```
1: procedure NORMALISERING(t)
2:     for all piksel P i bildet do
3:         if r^2 + g^2 + b^2 > t then
4:             skaler slik at max(r, g, b) = 255
5:         else
6:             sett r, g, b til 0
7:         end if
8:     end for
9: end procedure
```

Algoritme 8: Richard Blake's fargesegmentering steg 1 - normalisering [9]

Som beskrevet i 2.3.5 fortsetter en konturbevarende glattingsmekanisme som øker homogeniteten innad i regioner. Algoritme 9 viser hvordan dette gjøres.

Regioner kartlegges som tidligere nevnt ved å spore "vegger" av høy varians. Vår *region growing*-variant følger så opp for å redusere oversegmentering og la regioner adoptere regionløse piksler. Algoritme 10 presenterer essensen i denne prosessen. Her symboliserer B en dynamisk vektor over regionløse konturpiksler. Denne vektoren oppdateres ved at *fusjoner*-metoden legger inn nye piksler etter hvert som gamle skrelles av og overfører konturstemplet

videre til sine naboer. Etter hvert som prosessen går vil det naturligvis bli færre og færre regionløse piksler, slik at prosedyren konvergerer. *finnBesteNaboRegion*-metoden leter opp den naboregionen N som samsvarer best med fargeverdier til den aktuelle pikselen P, og *fusjoner*-metoden lar N spise opp P.

```
1:  procedure GLATTING(t1, t2)
2:      for i = 0..t2 do
3:          bool stop = true
4:          for all piksel P i bildet do
5:              v1 = varians over 3 * 3 maske med senter i P
6:              v2 = varians over 5 * 5 maske med senter i P
7:              if v2 < v1 + t1 then
8:                  glatt piksel P med naboer i 3 * 3 maske
9:                  stop = false
10:             end if
11:         end for
12:         if (stop) break;
13:     end for
14: end procedure
```

Algoritme 9: Richard Blake's fargesegmentering steg 2 - glatting [9]

```
1: procedure REGIONVEKST
2:     while B > 0 do
3:         P = trekk ut fremste piksel (element) fra B
4:         N = finnBesteNaboRegion(P)
5:         fusjoner(N, P)
6:     end while
7: end procedure
```

Algoritme 10: Richard Blake's fargesegmentering steg 3 - regionvekst

8.3 Objektorientert programmering

Inspeksjonssystemets implementasjon følger konstruksjonen med en objektorientert struktur. Målet er at systemet skal ha stor fleksibilitet og modularitet, og holde dørene åpne for flere egenskapskandidater. Eventuelle nye egenskapskandidater kan kort og godt legges til ved å lage klasser som blir satt til å arve funksjonalitet fra *Feature*-klassen, samt spesifisere deres identifikasjon, navn og dimensjon. I tillegg må de nye klassene implementere en kandidatspesifikk beregningsprosedyre i *calculate*-metoden, og registreres i *AVI*-klassens *readConfiguration* og *getFeatureFromID* metoder.

I noen av klassene er det gjort et poeng av kodeinnkapsling ut over klassesynlighetsdeklarasjoner (*public*, *protected* og *private*). Dette gjelder i det vesentligste klassene *ColorSegment* og *Sort*. Her defineres det metoder utenfor klassedefinisjonen. På denne måten er disse usynliggjort for resterende klasser, og er kun lesbar for sin klasse fordi implementasjonen fysisk ligger i samme *cpp*-fil som klassedefinisjonen. "Moralen" følger prinsippet om å gjøre den "synlige" koden minst mulig, da dette gir mindre risiko for at ting kan gå galt i eventuell videreutvikling av systemet.

Implementasjonen avviker litt fra løsningsforslaget i kapittel 6 på enkelte områder. Tankene går da først og fremst på at tabellinformasjon er baket inn i objekter. For eksempel er tabellen *objektinformasjon* ikke en tabell i implementasjonen, men tilsvarer istedet regionobjekter assosiert med bildeobjekter og lignende.

8.4 Egenskapskandidater

Systemet er implementert med 10 egenskapskandidater, hvorav 3 av dem utgjør den lille egenskapsmengden. Disse er valgt først og fremst på grunn av deres enkelthet, men det ligger i løsningsforslaget natur at mange andre egenskapskandidater bør legges til i et videre arbeid. Implementasjonen av de 10 egenskapskandidatene er gjort i hver sin egen *cpp*-fil, og teori for disse finnes i kapittel 3. Hver egenskapskandidat er "hardkodet" med identifikasjon, navn og dimensjon, og førstnevnte må korrespondere med *AVI*-klassens *getFeatureFromID*-metode. Som nevnt i konstruksjonskapitlet utfører ikke kandidatene i den lille egenskapsmengden (*Feature_ Area*, *Feature_ BoundingBox* og *Feature_ Centroid*) beregningen selv, men henter resultatene fra *AVI*-klassen. Det samme gjelder for kandidatene *Feature_ Mean* og *Feature_ Perimeter*, som beregnes i segmenteringsprosessen. Vi har til tross for dette valgt å "spandere" egne klassedefinisjoner på disse for ordens skyld, i tillegg til at det letter kodingen å la hver egenskapskandidat representeres av et objekt. Når det gjelder seleksjon av egenskapskandidater, så

gjøres dette kort og godt ved å bare velge kandidater som tilfredsstiller både hovedkrav 1 og 2 (6.1.3). Delavsnitt 8.4.1 til 8.4.5 beskriver kort implementasjonen av de andre egenskapskandidatene.

8.4.1 Feature_Circularity

Sirkularitetegenskapen (se avsnitt 3.2) beregnes som $\frac{perimeter^2}{areal}$. Denne gir en skalar verdi, og egenskapskandidaten er følgelig "hardkodet" med vektordimensjon lik 1.

8.4.2 Feature_FD

Dette er en implementasjon av *Fourier Descriptors*, der kun den høyeste frekvensen betraktes (bruker kun første koeffisient). For mer teori henvises leseren til avsnitt 3.1. Resultatet av beregningen blir et komplekst tall, og for å representere både reell og imaginær del "hardkodes" følgelig egenskapskandidatens vektordimensjon til 2.

8.4.3 Feature_Moments

Egenskapskandidaten implementerer 7 statistiske momenter som følger teorien fra avsnitt 3.2. Dette innebærer at vektordimensjonen er "hardkodet" til 7.

8.4.4 Feature_Radius

Undertegnde har valgt å innføre en egenskapskandidat som måler radius fra regionens massesenter til kontur (se avsnitt 3.2). Beregningene gjøres ved å iterere over regions konturpiksler og måle euklidsk distanse fra massesenteret. Variabler holder styr på minimal, maksimal og gjennomsnittlige distanse. Følgelig er egenskapskandidaten "hardkodet" med vektordimensjon lik 3.

8.4.5 Feature_Symmetry

Symmetriimplementasjonen (se avsnitt 3.2) måler et forholdstall over antall symmetriske konturpiksler om x og y-aksen i forhold til regionens totale antall konturpiskler. Egenskapskandidaten "hardkodes" derfor med vektordimensjon lik 1. Strategien er først å mappe alle regionens konturpiksler til posisjoner i en matrise. Dette gjøres ved å ta utgangspunkt i regionens omsluttende rektangel og massesenterpunkt. Konturpiksler merkes av i matrisen, som er initialisert med umerkede verdier. Neste steg er å iterere over matrisen, og summere opp symmetrier, som finnes ved å sammenligne matrisemerkinger om x og y-aksen. Siste steg er å dividere summen på regionens antall konturpiskler.

8.5 Utskrift til skjerm og fil

Det er lagt inn instrukser for å gjøre utskrift til skjerm på visse steder i koden. Hensikten er å gi tilbakemeldinger til brukeren om hvor prosessesringen befinner seg under eksekvering, da det i store treningssett med mange regioner kan gå med noe tid. Først ut er tilbakemeldinger på segmenteringsprosessen. Systemet skriver da ut: "Segmentering ferdig av godkjent/underkjent bilde: <relativ sti og filnavn>" etter hver bildesegmentering er unnagjort, og resultatene er representert i objektstruktur (bilde, region og pikselobjekter). For treningsfasen fortsetter systemet med å utskriften: "Beregner egenskapskandidater.....". Her tilsvarer hvert punktum en region som er prosessert ferdig (dvs alle egenskapsberegninger er gjort på denne). Mot slutten på treningsfasen skrives det ut treningsinformasjon om det nevrale nettverket. Her presenteres feilrater etter hvert som nettverket lærer. I eksekveringsfasen gis det i tillegg til tilbakemeldinger på segmenteringsprosessen, også klassifisringsresultater. Tilsvarende skrives også informasjon til fil. Dette gjøres naturligvis for å lagre informasjon, som for eksempel kan være treningsdata, egnede egenskapskandidater, ferdigtrenet nevralt nettverk og klassifiseringsresultater.

8.6 Doxygen

Alle kode er konsistent dokumentert med stil som støttes av *Doxygen* (se vedlegg). Det gis forklaring til samtlige klasser, attributter og metoder. I tillegg kommenteres det mer omfattende i *Sort.cpp*-filen sammenlignet med de andre filene, da vi fant det gunstig å gi ekstra forklaring her (for å forenkle lesbarheten av *Quicksort*, grådighetsmappingen og sammenhengen mellom disse).

Kapittel 9

Resultat

Dette kapitlet beskriver bildesett som er testet på inspeksjonssystemet, og resultater fra disse testene. Avsnitt 9.1 introduserer bildesettene. Videre fortsetter avsnitt 9.2 med å beskrive de aktuelle konfigurasjonsinnstillinger som ble brukt i testene. Avsnitt 9.3 avslutter med å presentere resultater som egnede egenskapskandidater, eksekveringstider og klassifiseringer.

9.1 Bildesett

Jeg prøvde tidlig i dette arbeidet å søke opp fritt tilgjengelige databaser over web. Hensikten var å finne reelle bilder fra industrien, som kunne brukes til å teste ut inspeksjonssystemet. Dette lykktes jeg derimot ikke med. En alternativ løsning ble derfor å lage egne bildesett. Disse ble tatt med et *Creative NX Ultra* webkamera, som sammen med en lyspære var festet i bunnen på en bøtte. Innsiden av bøtten var dekket med hvite A4-ark. Hensikten var å få frem gode lysforhold for de aktuelle objektene som skulle avbildes. Figur 9.1 viser utsiden og innsiden av denne innretningen.

Figur 9.1: Avbildningsinnretning

Undertengnede valgte 8 ulike sett av objekter til avbilding. Grunnen til disse ble valgt var først og fremst tilgjengelighet samt muligheten til å enkelt kunne manipulere objektene. Sistenevnte var særlig viktig for å skaffe frem variasjon i defekte objekter. Alle bildesett er lagret som flere 352 x 288 bitmap bilder, og disse bildene er plassert i en *godkjent* eller *underkjent* kategori. Tanken bak denne inndelingen er naturligvis at brukeren i treningsfasen skal kunne formidle hvilke type bilder som skal klassifiseres til godkjent og underkjent av systemet. En tredje kategori med navn *ubestemt* består av bilder som systemet selv skal klassifisere. Tabell 9.1 presenterer bildesettene og deres størrelse. Vi observererer at samtlige bildesett består av flere defekte enn ikke-defekte objekter. Det må presiseres at dette ikke er et mål i seg selv, men var mer bekvemmelig i form av å slippe å skaffe frem mange flere ikke-defekte objekter.

Bildesett	**Godkjent**	**Underkjent**	**Ubestemt**
Bolle	20	21	7
Fyrstikkeske	2	19	6
Gulrot	5	13	7
Hengelås	4	8	4
Karamell	5	34	10
Kjeks	2	28	14
Skrusett	3	14	4
Tunfisk	2	17	5
Sum	**43**	**154**	**57**

Tabell 9.1: Bildesett med antall bilder

Bildesettene varierer både i form, størrelse, farger og antall objekter i bildene, og det har vært en målsetning å oppnå realitet i disse. Bildesettene *hengelås* og *skrusett* avviker mer fra de andre settene, som har en mer opplagt forklaring av godkjente og underkjente objekter. I *hengelås*-bildene symboliserer en ulåst hengelås et defekt objekt, mens en låst hengelås symboliserer et ikke-defekt objekt. Når det gjelder *skrusett*, så representerer bildene som mangler ett eller flere verktøy et underkjent objekt, mens de som ikke mangler verktøy representerer et ikke-defekt objekt. Figureksempler på bilder fra de ulike bildesettene presenteres i avsnitt 9.3.

9.2 Konfigurasjon og parameterverdier

Inpseksjonssystemet er testet med alle 8 bildesett. De anvendte konfigurasjonsinnstillingene var felles for alle av disse, med unntak av 4 parametere som varierer fra sett til sett. Disse presenteres i tabell 9.2. Her representerer *vinduer* vinnduinndelingen (se 8.1.1), og fargesegmenteringsparameterne *intensitet*, *inhomogenitet* og *areal* henholdsvis minimale intensitet, maksimale

inhomogenitet og minimale areal for regioner (se 8.1.2). Foruten disse 4 parameterne kan ytterligere 3 (vektene for den lille egenskapsmengden, se 8.1.1) nevnes som ikke-generelle, i den forstand at også disse generelt må settes spesifikt for hvert bildesett. I våre 8 bildesett viste dette seg ikke å være nødvendig, da objektene ikke hadde nok variasjon i størrelse og antall til å "forlange" andre verdier av disse.

Bildesett	**Vinduer**	**Intensitet**	**Inhomogenitet**	**Areal**
Bolle	10	50	6	400
Fyrstikkeske	5	140	60	400
Gulrot	5	5	12	400
Hengelås	10	100	80	150
Karamell	5	60	20	400
Kjeks	9	140	10	500
Skrusett	10	196	40	400
Tunfisk	10	60	15	900

Tabell 9.2: Konfigurasjonsvarians

Følgende punktliste presenterer noen felles konfigurasjonsinnstillinger for de 8 bildesettene.

- Vekt areal: 0,001
- Vekt massesenterpunkt: 1
- Vekt omsluttende rektangel: 0,15
- Maksimalt antall glatteiterasjoner: 2
- Antall noder i de skjulte lagene: 10
- Øvre akseptable feilgrense ved trening for det nevrale nettverket: 0,00001
- Maksimalt antall treningsepoker: 10000
- Antall epoker mellom hver utskrift av statusrapport til skjerm: 1000

I tillegg til disse kommer egenskapskandidatenes generelle parameterverdier. Disse presenteres i figur 9.2 og 9.3, og presenterer de ulike resultatene av egenskapskandidatenes beregninger for bildesettene. Her står T_1 for terskelkravet som er assosiert med hovedkrav 1, og T_2 for terskelkravet som er assosisiert hovedkrav 2 (se 6.1.3). Gode egenskapskandidater må tilfredsstille både høy T_1 og høy T_2. Figurene er klippet ut fra et *Microsoft Excel* regneark (se vedlegg). Her kan man bestemme hvor mye valgte parameterverdier skal avvike fra gjennomsnittet. Disse er her satt til 75% for T_1 og 25% for T_2. Figurenes blå verdier representerer altså parameterverdiene som

er brukt i bildesettenes konfigurasjonsfiler. For hovedkrav 1 ble avviket satt såpass stort som 75% for at systemet i det hele tatt skulle klare å komme frem til egnede egenskapskandidater. Avviket for hovedkrav 2 kunne settes mye mindre uten å eliminiere for mange egenskapskandidater.

t1	Statistical moments							Fourier descriptor		Perimeter	Area	Centroid	
	M1	M2	M3	M4	M5	M6	M7	re	im			x	y
Bolle	0,194	0,115	0,143	0,125	0,072	0,126	0,046	0,110	0,110	0,170	0,151	0,115	0,128
Fyrstikkeske	0,500	0,500	0,500	0,500	0,500	0,500	0,500	0,991	0,991	1,000	1,000	0,667	0,556
Gulrot	0,184	0,253	0,282	0,401	0,262	0,272	0,173	0,231	0,231	0,285	0,351	0,253	0,289
Hengelås	0,124	0,114	0,136	0,187	0,169	0,137	0,173	0,108	0,108	0,131	0,123	0,208	0,392
Karamell	0,123	0,233	0,138	0,116	0,085	0,123	0,088	0,118	0,118	0,159	0,244	0,219	0,143
Kjeks	0,337	0,333	0,333	0,334	0,333	0,333	0,333	0,392	0,392	0,667	0,667	0,667	0,833
Skrusett	0,044	0,044	0,045	0,047	0,043	0,043	0,042	0,109	0,109	0,092	0,092	0,394	0,354
Tunfisk	0,500	0,500	0,500	0,500	0,500	0,500	0,500	0,989	0,989	0,995	1,000	0,833	0,750
Min	0,044	0,044	0,045	0,047	0,043	0,043	0,042	0,108	0,108	0,092	0,092	0,115	0,128
Snitt	0,251	0,262	0,260	0,276	0,246	0,254	0,232	0,381	0,381	0,437	0,453	0,419	0,431
Max	0,500	0,500	0,500	0,500	0,500	0,500	0,500	0,991	0,991	1,000	1,000	0,833	0,833
Valg	**0,063**	**0,065**	**0,065**	**0,069**	**0,061**	**0,064**	**0,058**	**0,095**	**0,095**	**0,109**	**0,113**	**0,105**	**0,108**

t1	Bouding Box				Circularity	Symmetry	Radius		Color			
	n	e	s	w		min	min	max	mean	r	g	b
Bolle	0,126	0,102	0,117	0,106	0,121	0,163	0,159	0,204	0,150	0,077	0,106	0,127
Fyrstikkeske	0,500	0,500	0,500	0,500	-1,#IND	0,676	0,581	0,549	0,712	0,500	0,500	0,500
Gulrot	0,311	0,196	0,258	0,178	0,239	0,276	0,411	0,405	0,412	0,190	0,288	0,308
Hengelås	0,296	0,285	0,371	0,164	0,120	0,142	0,171	0,199	0,130	0,116	0,120	0,121
Karamell	0,151	0,197	0,174	0,142	0,257	0,149	0,268	0,186	0,158	0,178	0,180	0,187
Kjeks	0,619	0,500	0,600	0,556	0,333	0,618	0,499	0,427	0,467	0,667	0,667	0,500
Skrusett	0,399	0,304	0,411	0,379	0,139	0,147	0,202	0,218	0,165	0,155	0,138	0,135
Tunfisk	0,500	0,500	0,500	0,500	0,750	0,952	0,693	0,882	0,975	0,900	0,643	0,650
Min	0,126	0,102	0,117	0,106	0,120	0,142	0,159	0,186	0,130	0,077	0,106	0,121
Snitt	0,363	0,323	0,367	0,316	0,245	0,390	0,373	0,381	0,396	0,348	0,330	0,316
Max	0,619	0,500	0,600	0,556	0,750	0,952	0,693	0,882	0,975	0,900	0,667	0,650
Valg	**0,091**	**0,081**	**0,092**	**0,079**	**0,061**	**0,098**	**0,093**	**0,095**	**0,099**	**0,087**	**0,083**	**0,079**

Figur 9.2: Parameter for egenskapskandidater - hovedkrav 1

t2	Statistical moments							Fourier descriptor		Perimeter	Area	Centroid	
	M1	M2	M3	M4	M5	M6	M7	re	im			x	y
Bolle	0,691	0,316	0,239	0,332	0,856	0,591	0,142	0,556	0,556	0,445	0,542	0,283	0,276
Fyrstikkeske	0,835	0,719	0,868	0,870	0,136	0,815	1,185	0,978	0,978	0,979	0,990	0,811	0,402
Gulrot	0,569	0,970	0,969	0,693	0,646	0,981	0,512	0,499	0,499	0,241	0,409	0,880	0,539
Hengelås	0,944	0,943	0,947	0,933	0,998	0,969	0,998	0,600	0,600	0,862	0,810	0,515	0,710
Karamell	0,463	0,446	0,523	0,409	0,733	0,633	0,273	0,622	0,622	0,623	0,705	0,487	0,013
Kjeks	0,927	0,865	0,693	0,541	0,940	0,625	0,866	0,961	0,961	0,973	0,957	0,895	0,645
Skrusett	0,321	0,420	0,583	0,251	0,506	0,574	0,063	0,233	0,233	0,629	0,671	0,501	0,454
Tunfisk	0,514	0,977	0,930	0,977	1,000	1,000	1,000	0,694	0,694	0,582	0,569	0,688	0,941
Min	0,321	0,316	0,239	0,251	0,136	0,574	0,063	0,233	0,233	0,241	0,409	0,283	0,013
Snitt	0,658	0,707	0,719	0,626	0,727	0,773	0,630	0,643	0,643	0,667	0,707	0,632	0,497
Max	0,944	0,977	0,969	0,977	1,000	1,000	1,185	0,978	0,978	0,979	0,990	0,895	0,941
Valg	**0,493**	**0,530**	**0,539**	**0,469**	**0,545**	**0,580**	**0,472**	**0,482**	**0,482**	**0,500**	**0,530**	**0,474**	**0,373**

t2	Bouding Box				Circularity	Symmetry	Radius			Color		
	n	e	s	w			min	max	mean	r	g	b
Bolle	0,199	0,401	0,347	0,397	0,396	0,186	0,499	0,356	0,505	0,633	0,547	0,791
Fyrstikkeske	0,539	0,926	0,572	0,912	1,000	0,471	0,733	0,892	0,966	0,926	0,913	0,862
Gulrot	0,049	0,538	0,109	0,613	0,113	1,000	0,217	0,106	0,067	0,468	0,232	0,241
Hengelås	0,778	0,738	0,630	0,466	0,824	0,359	0,707	0,693	0,719	0,783	0,752	0,753
Karamell	0,122	0,629	0,144	0,352	0,259	0,254	0,538	0,516	0,652	0,357	0,338	0,348
Kjeks	0,841	0,931	0,741	0,935	0,955	0,811	0,754	0,801	0,948	0,927	0,956	0,974
Skrusett	0,490	0,546	0,498	0,476	0,380	0,115	0,488	0,359	0,521	0,515	0,586	0,616
Tunfisk	0,787	0,965	0,782	0,688	0,920	0,911	0,838	0,366	0,605	0,899	0,936	0,904
Min	0,049	0,401	0,109	0,352	0,113	0,115	0,217	0,106	0,067	0,357	0,232	0,241
Snitt	0,476	0,709	0,478	0,605	0,606	0,513	0,597	0,511	0,623	0,689	0,657	0,686
Max	0,841	0,965	0,782	0,935	1,000	1,000	0,838	0,892	0,966	0,927	0,956	0,974
Valg	**0,357**	**0,532**	**0,358**	**0,454**	**0,454**	**0,385**	**0,448**	**0,383**	**0,467**	**0,516**	**0,493**	**0,515**

Figur 9.3: Parameter for egenskapskandidater - hovedkrav 2

9.3 Resultater

Tabell 9.3 viser hvilke egenskapskandidater som systemet fant egnede for de aktuelle bildesettene (x symboliserer egnet). I tabellen representeres egenskapskandidatenes med siffer fra 0-9, som henholdsvis symboliserer *statistiske momenter*, *Fourier Descriptors*, *perimeter*, *areal*, *massesenterpunkt*, *omsluttende rektangel*, *sirkularitet*, *radius*, *symmetri* og *gjennomsnittsfarge*. Det er verdt å merke seg at systemet finner hver av de 10 egenskapskandidatene egnet for minst 4 av bildesettene.

Bildesett	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Bolle	-	-	-	x	-	-	-	-	-	-
Fyrstikkeske	x	x	x	x	x	x	x	x	x	x
Gulrot	x	x	-	-	x	-	-	-	x	-
Hengelås	x	x	x	x	x	x	x	x	-	x
Karamell	-	x	x	x	-	-	-	x	-	-
Kjeks	x	x	x	x	x	x	x	x	x	x
Skrusett	-	-	-	-	x	x	-	-	-	-
Tunfisk	x	-	x	x	x	x	x	-	x	x
Sum	**5**	**5**	**5**	**6**	**6**	**5**	**4**	**4**	**4**	**4**

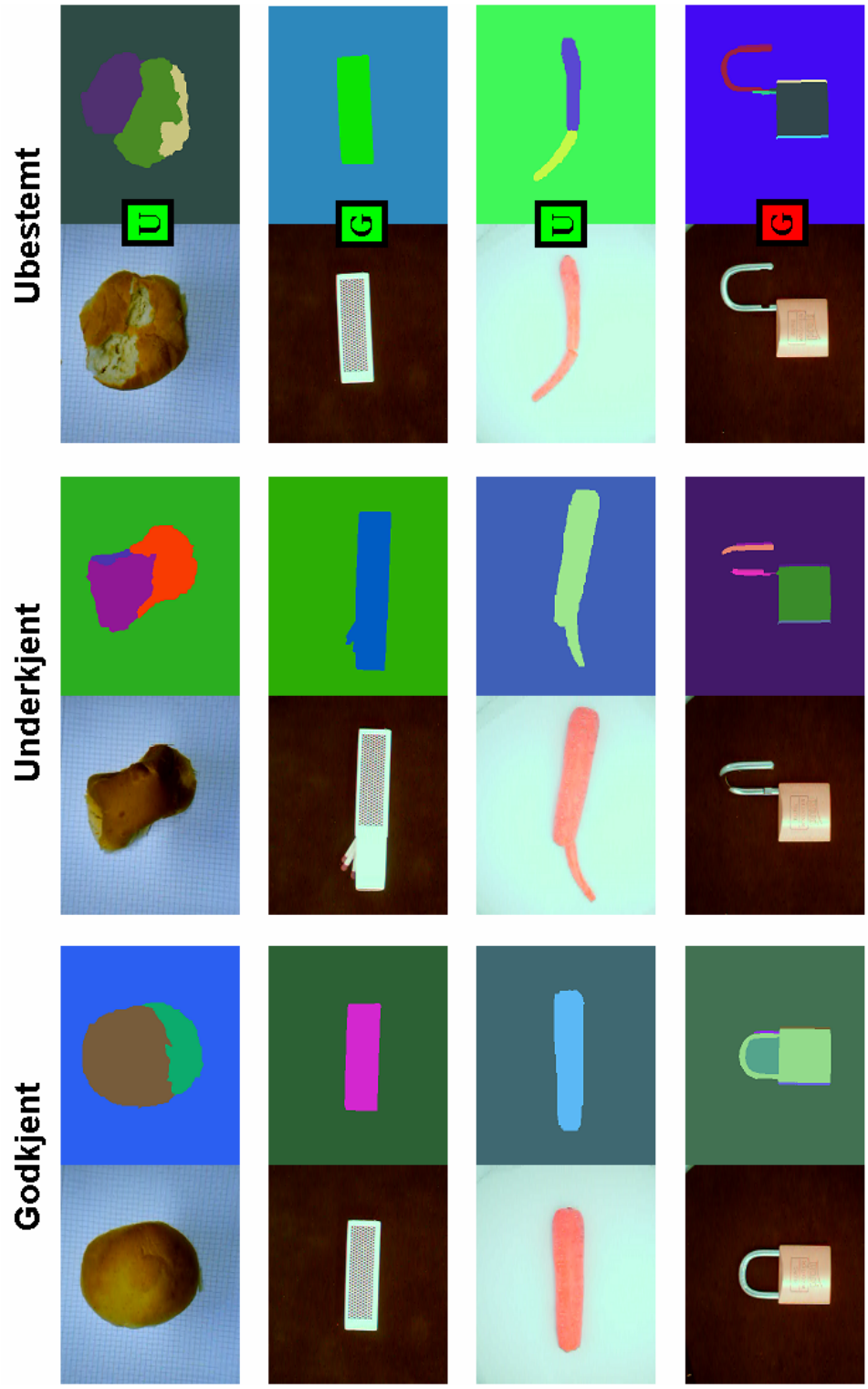
Tabell 9.3: Utvalgte egenskapskandidater

Tabell 9.4 presenterer kjøretider for inspeksjonssystemet. For eksempel ser vi at bildesettet *bolle* kjørte på 38 sekunder i treningsfasen, som tilsvarer 1,1 bilder/sek. Tilsvarende var tiden under eksekveringsfasen 5 sekunder, som betyr 1,4 bilder/sek. Maskinvaren som er brukt er en 3 GHz Pentium 4 prosessor. Vi observerer at våre bildesett gjennomsnittlig kjører med 1,2 bilder per sekund i eksekveringsfasen. Et viktig poeng som kommer frem i tabellen er at *skrusett* bruker lang tid på treningsfasen. Dette henger sammen med mange segmenterte regioner i disse bildene kontra andre bildesett.

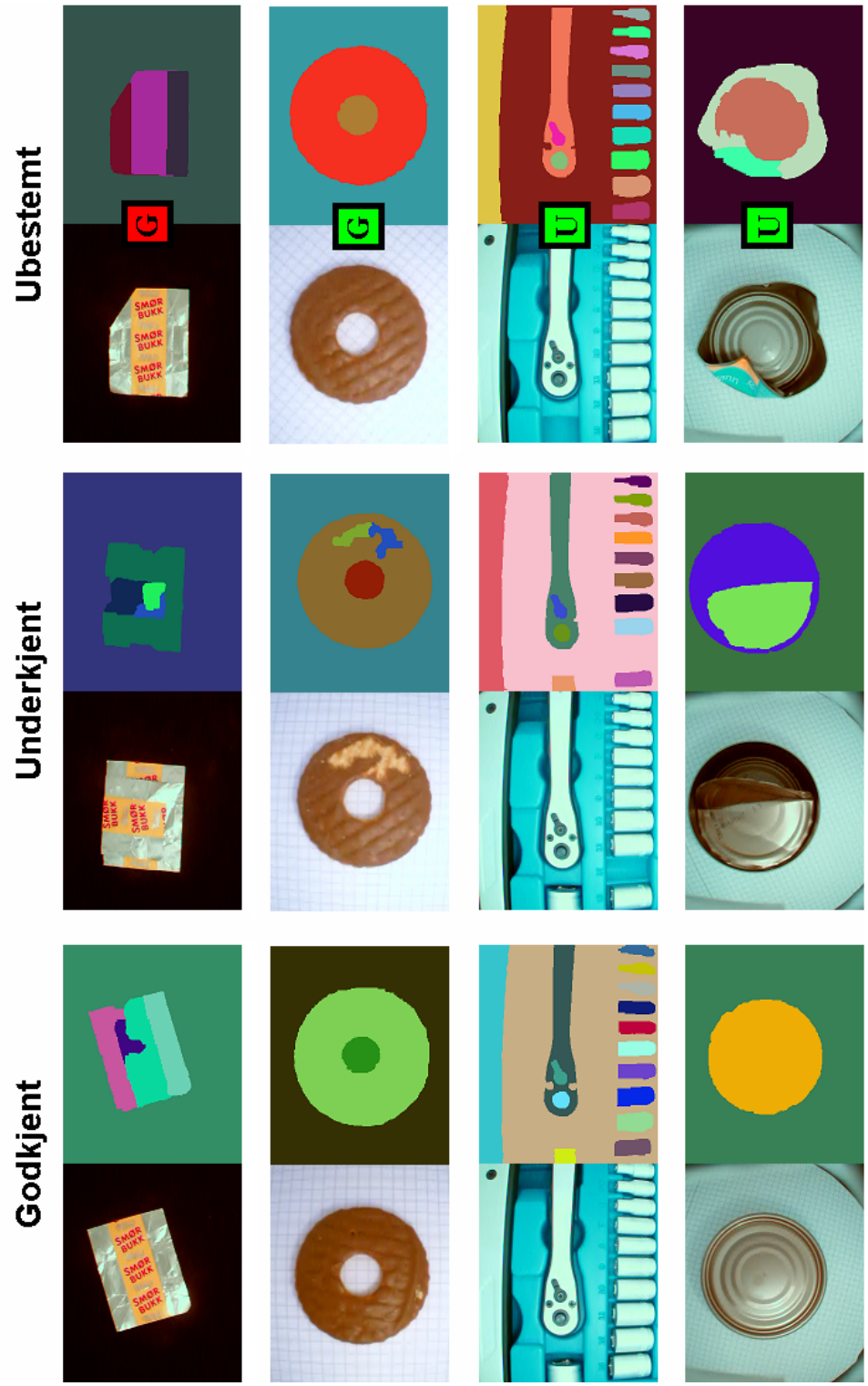
Bildesett	**Treningsfase**	**Eksekveringsfase**
Bolle	38 / 1,1	5 / 1,4
Fyrstikkeske	18 / 1,2	5 / 1,2
Gulrot	18 / 1,0	5 / 1,4
Hengelås	10 / 1,2	4 / 1,0
Karamell	36 / 1,1	7 / 1,4
Kjeks	32 / 0,9	15 / 0,9
Skrusett	52 / 0,3	3 / 1,3
Tunfisk	21 / 0,9	5 / 1,0
Snitt	**28 / 1,0**	**6 / 1,2**

Tabell 9.4: Kjøretider

Fra tabell 9.1 kan vi lese at inspeksjonssystemet er testet ut med 57 ubestemte bilder. Klassifiseringen er gjort på bakgrunn av treningsinformasjon bestående av 43 godkjente og 154 underkjente bilder. Dette blir til sammen 254 bilder. Systemet genererer et segmentert bilde for hvert av disse. Følgelig blir det totalt sett 508 bilder, som er behandlet eller generert av systemet. For fulle og hele oversikt henvises leseren til elektronisk vedlegg som inneholder alle bildene systemet er testet med. Figur 9.4 og 9.5 presenterer et lite utvalg fra bildesettene. På disse figurene vises bilder og segmenteringer av godkjente, underkjente og ubestemte objekter. På de ubestemte objektene vises også et symbol som forteller hvordan systemet i en av testene klassifiserte bildene. Der står G for godkjent og U for underkjent. Grønn bakgrunnsfarge symboliserer at systemet klassifiserte riktig, mens rød symboliserer at systemet klassifiserte galt.



Figur 9.4: Noen resultater fra bildesettene: *bolle*, *fyrstikkeske*, *gulrot* og *hengelås*



Figur 9.5: Noen resultater fra bildesettene: *karamell*, *kjeks*, *skrusett* og *tunfisk*

Tabell 9.5, 9.6 og 9.7 viser samlede resultater fra inspeksjonssystemet. Hver tabell representerer 3 tester av samtlige bildesett. Tallene sier hvor mange bilder som er riktig klassifisert. Vi observerer at resultatene varierer noe. Dette skyldes trolig ulike initialiseringer av vekter i det nevrale nettverket, da disse velges tilfeldig. De ulike tabellene skiller seg fra hverandre ved å initialisere vektene i det nevrale nettverket med ulike nedre og øvre grenser. Det er interessant å observere at gjennomsnittet av alle disse 9 gjennomførte testene resulterer i 52 av 57 riktige, altså 91% riktig klassifisering.

Bildesett	**Test 1**	**Test 2**	**Test 3**	**Mulige**
Bolle	7	7	7	7
Fyrstikkeske	6	6	6	6
Gulrot	5	5	7	7
Hengelås	4	2	3	4
Karamell	8	7	6	10
Kjeks	14	14	13	14
Skrusett	3	3	3	4
Tunfisk	5	5	5	5
Samlet/prosent	**52/91**	**49/86**	**50/88**	**57/100**

Tabell 9.5: Testresultater med vekter i nevralt nettverk initialisert tilfelding mellom -0,1 og 0,1

Bildesett	**Test 4**	**Test 5**	**Test 6**	**Mulige**
Bolle	7	7	7	7
Fyrstikkeske	6	6	6	6
Gulrot	7	7	5	7
Hengelås	4	4	4	4
Karamell	9	9	6	10
Kjeks	14	14	14	14
Skrusett	3	4	4	4
Tunfisk	5	5	5	5
Samlet/prosent	**55/96**	**56/98**	**51/89**	**57/100**

Tabell 9.6: Testresultater med vekter i nevralt nettverk initialisert tilfelding mellom -0,2 og 0,2

Bildesett	**Test 7**	**Test 8**	**Test 9**	**Mulige**
Bolle	7	7	7	7
Fyrstikkeske	6	6	6	6
Gulrot	7	5	6	7
Hengelås	4	3	4	4
Karamell	9	9	7	10
Kjeks	14	10	10	14
Skrusett	4	3	3	4
Tunfisk	5	5	4	5
Samlet/prosent	**56/98**	**48/84**	**47/82**	**57/100**

Tabell 9.7: Testresultater med vekter i nevralt nettverk initialisert tilfelding mellom -0,25 og 0,25

Kapittel 10

Analyse

Dette kapitlet trekker frem viktige analytiske trekk ved inspeksjonssystemet. Noe av dette ble introdusert i løsningsforslags, implementasjons og resultatkapitlene, men kategoriseres og utdypes ytterligere her. I tillegg til dette gjøres nye analyser som ikke er nevnt tidligere i rapporten. Avsnitt 10.1 beskriver vekter og vinduinndeling, og avsnitt 10.2 presenterer viktige trekk i segmenteringen. Videre følger avsnitt 10.3 med egenskapskandidater, avsnitt 10.4 med nevrale nettverk, avsnitt 10.5 med kjøretider, og til slutt avsnitt 10.6 med kort beskrivelse av inspeksjonssystemets nytteverdi.

10.1 Vekter og vinduinndeling

Det er helt essensielt å sette vekter for den lille egenskapsmengden samt vinduinndelinger spesifikt for hvert bildesett. Ulike avbildninger krever ulike parameterverdier. Som nevnt tidligere krever eksempelvis bilder, som er preget av store objekter, helt andre arealvekter enn bilder, som er preget av små objekter. Tilsvarende vil bilder med mange små objekter sannsynligvis kreve flere vinduinndelinger enn bilder med få og store objekter. Tabell 9.2 viser varians over gunstige vinduinndelinger av bildesettene. Når det gjelder vektene av den lille egenskapskandidatmengden viste det seg at disse kunne brukes felles for alle 8 bildesett. Dette skyldes at objektene ikke hadde stor nok variasjon i størrelse og antall til at disse vektene måtte spesialtilpasses hvert bildesett.

10.2 Segmentering

På tilsvarende måte som for parametere for vekter og vinduinndeling er det nødvendig å sette segmenteringsparametere spesifikt for hvert bildesett. Samtlige tester av inspeksjonssystemet er etter min vurdering basert på nær optimale parameterinnstillinger, som får frem det mest vesentligste i disse bildesettene. Tabell 9.2 viser hvordan de tre segmenteringsparameterne

minimal intensitet, *maksimal inhomogenitet* og *minimalt areal* varierer over bildesettene.

Det er ikke til å stikke under en stol at segmenteringsmetoden kan komme til kort, og særlig når den mates med objektavbildninger der mange små detaljer er vikitge. Det er vel og merke også et poeng at segmenteringsresultatene ikke nødvendigvis trenger å være perfekte, dog "bare" konsistente og tilstrekkelig gode for at systemet skal kunne klassifisere riktig på bakgrunn av disse.

10.3 Egenskapskandidater

Som nevnt i avsnitt 6.1.3 og 9.2 må gode egenskapskandidater tilfredsstille systemets to hovedkrav. Parameterverdiene som bestemmer egenskapskandidaters terskelverdier for disse hovedkravene, T_1 og T_2, er tiltenkt generelle for alle bildesett. Det er et vesentlig poeng at kravene til egenskapskandidatenes prestasjon blir større jo høyere T_1 og T_2 settes. For å holde disse på "jordnært" nivå, valgte vi å sette T_1 og T_2 på henholdsvis 75 og 25 prosent avvik fra gjennomsnittet. Såpass store avvik viste seg nødvendig for at systemet i det hele tatt skulle klare å komme opp med egnede egenskapskandidater for flere av bildesettene. Det er riktignok grunn til å tro at implementasjon av flere egenskapskandidater kan føre til at terskelverdiene T_1 og T_2 kan settes nærmere gjennomsnittet. Dette fordi flere egenskapskandidater sannsynligvis medfører større sjanse for at systemet kommer opp med noen egnede egenskapskandidater. Det er også rimelig å anta at den store men nødvendige forskjellen mellom gjennomsnittsavikene til hovedkrav 1 og 2 skyldes en ekstra sterk innflytelse fra ett spesielt homogent bildepar. Dette kommer tydelig frem i uttrykket for hovedkrav 1 (se 6.1.3), der vi ser nevneren representerer maksimal differanse mellom godkjente bilders objekter i det aktuelle bildesettet. I uttrykket for hovedkrav 2 (se 6.1.3) ser vi at alle bilder i bildesettene har like stor innflytelse (inneholder bare ledd som symboliserer gjennomsnitt). Følgelig kan det være grunn til å tro at ett bildepar i bildesettet lettere kan "ødelegge" egnede egenskapskandidater for hovedkrav 1 enn 2, og derav nøvendigvis større gjennomsnittsavvik i T_1 enn T_2.

Det er naturlig å tenke seg at en økning av antall egenskapskandidater kan føre til forbedringer prestasjonsmessig, samt kanskje også et mer robust inspeksjonssystem. Økt prestasjon er som sagt først og fremst mulig med sterkere krav til T_1 og T_2, og økt robusthet på grunn av større mulighet til å fange essensielle egenskaper i objektene. Medaljens bakside kommer frem ved at flere egenskapskandidater også kan finne "ødeleggende" likhetstrekk i objektene som strengt talt skal være ulike. Dette begrenser naturligvis den posistive effekten ved å innføre flere egenskapskandidater noe. Det er deri-

mot helt klart at en økning av antall egenskapskandidater vil medføre økt beregningstid for systemet. Den gode nyheten er at denne belastningen hovedsakelig vil være knyttet til treningsfasen, da eksekveringsfasen bare går gjennom de egenskapskandidater som er funnet egnede, mens treningsfasen må betrakte alle. Det er vanskelig å analytisk estimere et svar på spørsmålet om hvor mange egenskapskandidater som trengs. Vi kan ikke si annet enn at ytterligere testing helt klart er et viktig stikkord i denne forbindelse.

Det er verdt å merke seg at vi har valgt konturbasert fremfor regionbasert implementasjon av flere egenskapskandidater. Grunnen til dette er ene og alene en betydelig besparelse av beregninger og følgelig tidsbruk i systemet, da regionbaserte egenskapskandidater naturligvis krever langt flere pikseliterasjoner enn konturbaserte egenskapskandidater. Flere regionbaserte egenskapskandidater hadde imidlertid vært gunstige for systemet, da fargesegmenteringens *region growing*-prosess nok i mange tilfeller ikke virker fullt så drastisk på disse, som på konturbaserte regioner.

10.4 Nevralt nettverk

Parameterinnstillinger for det nevrale nettverket er tiltenkt generelle og felles for alle bildesett. 10 noder i ett sjult lag ser ut til å fungere temmelig bra for alle de åtte bildesettene som er testet, til tross for at antall inputnoder i bildesettene varierer fra 100 til 2400. Vi vil derimot ikke utelukke at andre oppbygninger av antall noder og skjulte lag kan gi bedre resultater. Dette bør testes ut ytterligere.

Det er rimelig å anta at en betydelig økning av antall bilder i hvert bildesett kan forbedre systemets klassifiseringsresultater betraktelig. Kilden til resultatenes variasjon i de 9 testene antas å være tilfeldige initialiseringer av vekter i det nevrale nettverket. Det virker som om systemet ikke får nok treningsbilder til å la vektene i det nevrale nettverket konvergere tilstrekkelig. Et viktig poeng som støtter opp under dette er at det kun er disse initialiseringene som er ikke-deterministisk i systemet, og følgelig kan bare variasjon i klassifiseringsresultatene på ett og samme bildesett skyldes disse (forutsatt at alle parametere holdes like selfølgelig). Det er derfor heller ingen løsning å stille sterkere krav til feilraten i det nevrale nettveket, da det rett og slett er mangel på treningsbilder som antas å være problemet.

10.5 Kjøretid

Fra tabell 9.4 kan vi lese at inspeksjonssystemet på bakgrunn av våre 8 bildesett gjennomsnittlig behandler 1,2 biler/sek i eksekveringsfasen. I treningsfasen er behandlingstiden lavere, men dette har mindre betydning. Det

er et viktig poeng at antall regioner har stor innvirkning på behandlingstiden. Dette er spesielt tydelig i treningsfasen, der hver region beregnes av alle egenskapskandidater. Andre faktorer som virker på treningsfasen er selvsat antall bilder i bildesettet. Jo flere bilder, desto mer bildepar, regionmappinger og egenskapskandidatberegninger.

Det er klart at systemets kjøretid kan effektiviseres på mange måter. For eksempel kan systemet velge et utvalg bilder til mapping, i stedet for å mappe alle godkjente bilder mot hverandre, samt alle godkjente mot underkjente bilder. Størrelsen på utvalget må da være tilstrekkelig stort nok til at essensielle trekk i bildesettene ikke går tapt. Omstrukturering og optimalisering av koden kan også være en god kilde til å utbedre systemet. Det kan for eksempel være fristende å file på modularitetsprinsippet for beregningen av egenskapskandidater til fordel for en alternativ strategi, som av effektivitetsmessige hensyn samler beregninger av ulike egenskapskandidater på regioner i felles metoder. Det er ikke til å stikke under en stol at dette ville spart på antall pikseliterasjoner, kontra en egen *calculate*-metode for hver egenskapskandidater, som hver gjør sine egne iterasjoner. Vi har dog valgt å fryse ned denne ideen til fordel for modularitet og intuitiv lesbarhet av koden.

10.6 Nytteverdi

Som nevnt i 10.2 ligger det en fare for at segmenteringsdelen av systemet ikke får med seg det vesentligste fra bilder med mange små detaljer. Følgelig har resten av systemet liten mulighet til å prestere godt på slike eksempler. Ingen av de 8 bildesettene som inspeksjonssystemet er testet med kan riktignok sies å tilhøre en slik kategori.

Det har hele tiden vært en vikig målsetning å holde antall parametere på et relativt beskjedent nivå. Systemet kan sies å være fleksibelt og raskt konfigurerbart for de ulike anvendelsesområder/bildesett. Det at systemet lærer fra reelle avbildninger i de omgivelser det selv skal operere, er en attaktiv tanke, og for industrielt bruk burde det også være relativt enkelt å skaffe frem bildesett for treningsfasen. Vi kan driste oss til å si at det ligger potensiale i systemet og dets metoder. På bakgrunn av 8 ganske tilfeldige bildesett klarte systemet å gjennomsnittlig klassifiserere 91% riktig. Da det er muligheter for forbedringer med større treningssett og flere egenskapskandidater, kan systemet utbedre klassifiseringsriktigheten ytterligere.

Kapittel 11

Videre arbeid

Dette kapitlet følger opp analysen i forrige kapittel med å beskrive potensielle utbedringer av inspeksjonssystemet. Avsnitt 11.1 til 11.5 beskriver henholdsvis hvordan bildesett, egenskapskandidater, dynamikk, effektivisering og andre tiltak kan drive frem forbedringer i systemet.

11.1 Bildesett

En økning av antall bilder per bildesett kan som nevnt i 10.4 sannsynligvis forbedre systemets klassifiseringsriktighet. Vi mener derfor at systemet absolutt bør testes ut med større bildesett. I tillegg til bildesettenes størrelse bør det også tas i bruk flere bildesett. Flere bildesett bidrar til å øke generaliteten for systemets bruksområder. Her antas at mange bildesett brukes til å stille inn krav til egenskapskandidatene en gang for alle. Det er dog ingen ting i veien for å stille disse spesifikt for ett bruksområde dersom brukeren ønsker dette. Dette er imidlertid ikke vår intensjon, da mye av generaliteten til systemet da går i vasken. Systemet bør også testes ut med vanskeligere bildesett, og gjerne i form av mer eller mindre detaljpregede bilder som kan skape vanskeligheter for segmenteringsdelen. Et annet poeng er å gjøre tester som tar sikte på å avdekke et gunstig forhold mellom antall godkjente og underkjente bilder med tanke på systemets treningsfase.

11.2 Egenskapskandidater

Det er helt klart viktig å implentere flere egenskapskandidater. Dette vil kunne føre til forbedringer prestasjonsmessig, da som nevnt i 10.3 flere egenskapskandidater gir større mulighet til å fange essensielle egenskaper i objektene. Med flere egenskapskandidater kan vi også tilltate oss å sette strengere krav (T_1 og T_2) til egenskapskandidatene. Det oppfordres derfor til å teste ut systemet med ulike verdier for å avgjøre hvor strenge disse kravene bør

være. Det kan også være en ide å "frigi" hver dimensjon i egenskapskandidater, som for eksempel for *statsistiske momenter*. Med "frigi" menes her at de momenter som tilfredsstiller egenskapskandidatkravene kan brukes, fremfor "alle eller ingen"-prinsippet som er implementert.

11.3 Dynamikk

Inspeksjonssystemet kan trolig forbedres ved å innføre dynamiske trekk i inspeksjonssystemet. En enkel mulighet er å automatisere verdier for krav til egenskapsparametere i konfigurasjonsfilen på bakgrunn av en mengde bildesett. Dette kan gjøres etter samme strategi som brukes i reknearket (avbildet i figur 9.2 og 9.3), der terskelverdiene bestemmes etter ett fast prosentavvik fra gjennomsnittet. Videre kan man tenke seg at systemet selv bestemmer vinduinndelinger i henhold til antall regioner og deres posisjoner, form og størrelse. Samme tanke går også for vektene i den lille egenskapsmengden. Ytterligere kan det være gunstig å gjøre kredittering og seleksjon av egenskapskandidater på en dynamisk måte. Dette gjelder særlig i tilfeller der mange egenskapskandidater vurderes som egnede (forutsetter implementasjon av mange egenskapskandidater). Tanken er da at systemet bare velger ut de beste av de beste. Kapittel 4 beskriver en mulig strategi her. For "ekstreme" utfordringer kan selvkonfigurerende segmenteringsparametere og topologi i det nevrale nettverket nevnes som attraktive utbedringer.

11.4 Effektivisering

Effektivisering av inspeksjonssystemet er en viktig del i det videre arbeidet. Økt effektivitet gir blant annet rom for implementasjon av flere egenskapskandidater og bruk av større bildesett. Som beskrevet i 10.5 kan dette for eksempel gjøres ved å anvende et utvalg bildepar fremfor "alle til alle"-mappinger, samt samle beregninger av egenskapskandidater i felles metoder for å spare pikseliterasjoner. Det kan i tillegg være gull verdt å utforske muligheter for å videreutvikle systemet for flerprosessorsystemer. "Parallelle"-implementasjoner kan skape enorme forbedringer ytelsesmessig, og det burde være fullt mulig å tilpasse systemet til på slike arkitekturer. Det må sies at det ikke har lagt stor vekt på optimalitet i kodingen. Derfor ligger det også et stort potensiale i utvikling av bedre algoritmer, og optimalisering ved hjelp av assemblykode.

11.5 Andre tiltak

Det er viktig å utbedre fargesegmenteringsmetoden til å ta høyde for bilder med små og viktige detaljer. Alternativt bør systemet testes med andre segmenteringsmetoder. Når det gjelder det nevrale nettverket, så er det fortsatt

en utfordring å finne ut topologier (antall noder og lag) som viser seg å være best tilpasset inspeksjonssystemet. Å innføre *region adjacency graph* [5] kan bidra til bedre mappinger i bildepar, og til syvende og sist bedre klassifiseringer. En annen tanke kan være å anvende metoder for transformasjonsinvarians for å avlaste kompleksiteten i det nevrale nettverket.

Vedlegg

Dette vedlegget beskriver ressurser som er anvendt i selve systemet og dets utvikling, samt arbeidet med å dokumentere systemet. Disse blir presentert i følgende to avsnitt, som henholdsvis vektøy og biblioteker.

Verktøy

Seks verktøy er anvendt i arbeidet med denne hovedoppgaven. Disse har hjulpet til med både utarbeidelsen av rapporten og selve inspeksjonssystemet, og kan kort nevnes på følgende måte:

Microsoft Visual Studio .NET 2003

Systemet er implementert ved hjelp av utviklingsmiljøet *Microsoft Visual Studio .NET 2003* [53].

Doxygen

Doxygen [54] er brukt til å generere dokumentasjon av koden. Vi har valgt å generere resultatet i *HTML*-format [55], slik leseren kan aksessere dette på en interaktiv måte.

Umbrello UML modeller

Umbrello UML modeller [56] er et *unix*-basert [57] verktøy for å lage UML [58] modeller. Alle klasse- og sekvensdiagrammene som fremgår i denne rapporten er produsert ved hjelp av dette verktøyet.

Microsoft Visio

Microsoft Visio [59] er et modelleringsverktøy som vi har anvendt til utarbeidelsen av flere figurer i denne rapporten.

TeXnicCenter

TeXnicCenter [60] er grafisk brukergrensesnitt på *Microsoft Windows* [61] plattformen for utvikling av *LaTex*-dokumenter [62]. Denne rapporten er utviklet ved hjelp av dette verktøyet.

Microsoft Excel

Microsoft Excel [63] er et verktøy for å utvikle regneark. Programvaren brukes i dette arbeidet for å presentere nedre, snitt og øvre grenser, samt valg av egenskapskandidaters parameterverdier på bakgrunn av snittavvik.

Biblioteker

For både å konsentrere arbeidets omfang og ressursbruk til størst mulig grad av "bildebehandling av oppgavens interesse", valgte vi å ta i bruk fritt tilgjengelig kode fra andre prosjekter. Inspeksjonssystemet har endt opp med å anvende åpen kode fra to biblioteker. Det være seg *Fast Artificial Neural Network Library* [64], og *Image Processing Library 98* [65]. Tidlig i arbeidet ble også et tredje bibilotek med navn *Insight Segmentation and Registration Toolkit* [66] anvendt. Dette ble senere tatt bort grunnet uakseptable resultater kjøretidsmessig. Til tross for dette velger vi å spandere noen ord på sistnevnte bibliotek, da dette ved effektiviserering av kjøretiden kan utgjøre en alternativ segmenteringsmodul i systemet.

Fast Artificial Neural Network Library

Fast Artificial Neural Network Library (FANN) [64] er et biliotek for nevrale nettverk. Dette støtter flerlagsakitektur og *backpropagation*-algoritmen med mer, og går for å være enkelt i bruk og raskt kjøretidsmessig. Inpseksjonssystemet sender og mottar all informasjon tilknyttet nevrale nettverk til og fra dette biblioteket.

Image Processing Library 98

Image Processing Library 98 (IPL98) [65] er et bibliotek for bildebehandling. Biblioteket støtter standard metoder for å skape, prosessere, vise og lagre bildeinformasjon med mer. Vi bruker bibloteket til å lese og skrive *BMP* og/eller *PNG*-formater.

Insight Segmentation and Registration Toolkit

Insight Segmentation and Registration Toolkit (ITK) [66] er et bibliotek som implementerer metoder og teknikker i bildebehandling (kanskje særlig

rettet mot medisinsk bruk). Vi prøvde tidlig i arbeidet ut en *watershed*-segmentering i dette biblioteket, men fant ut at den anvendte metoden i gjeldene versjon ikke tilfredstiller kjøretidsmessige krav til industrielle inspeksjonssystem.

Bibliografi

[1] Per Gunnar Bårdsen. Automatisk visuell inspeksjon for generelle objekter [online]. Available from: `http://www.idi.ntnu.no/~pergunb/fordypningsprosjekt.pdf`.

[2] K.S. Fu; J.K. Mui. A survey on image segmentation. *Pattern Recognition*, 13(1), 1981.

[3] N.R. Pal; S.K. Pal. A review on image segmentation techniques. *Pattern Recognition*, 26(9), 1993.

[4] P. Dulyakarn; P. Thitimajshima; Y. Rangsanseri. Image segmentation through a multithresholding based on gray-level co-occurrence. *Proceedings of the SPIE - The International Society for Optical Engineering*, 4067(pt.1-3), 2000.

[5] Milan Sonka; Vaclav Hlavac; Roger Boyle. *Image Processing, Analysis and Machine Vision*. Thomson-Engineering, 1998.

[6] Rafael C. Gonzalez; Richard E. Woods. *Digital Image Processing*. Prentice Hall, 2002.

[7] T.W. Ridler; S. Calvard. Picture thresholding using an iterative selection method. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-8(8), 1978.

[8] M. Sezgin; B. Sankur. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, 13(1), 2004.

[9] Richard Blake. Fargesegmentering - tdt4265 datasyn, ntnu [online]. Available from: `http://dionysus.idi.ntnu.no/~cv/COLSSEGA.html,http://rover.idi.ntnu.no/~cv/COLSSEGAA.shtml`.

[10] Thomas H. Cormen; Charles E. Leiserson; Ronald L. Rivest; Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2001.

[11] Luc Vincent; Pierre Soille. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6), 1991.

[12] Ida-Maria Sintorn. *Segmentation methods and shape descriptions in digital images*. PhD thesis, Swedish University of Agricultural Sciences, 2005.

[13] Hai Gao; Wan-Chi Siu; Chao-Huan Hou. Improved techniques for automatic image segmentation. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(12), 2001.

[14] V. Grau; A.U.J. Mewes; M.Alcaniz; R.Kikinis; S.K. Warfield. Improved watershed transform for medical image segmentation using prior information. *IEEE Transactions on Medical Imaging*, 23(4), 2004.

[15] Tim McInerney; Demetri Terzopoulos. Deformable models.

[16] Michael Kass; Andrew Witkin; Demetri Terzopoulos. Snakes: Active contour models. Proceedings - First International Conference on Computer Vision, IEEE, New York, NY, USA, 1987.

[17] Chenyang Xu; J.L. Prince. Snakes, shapes, and gradient vector flow. *IEEE Transactions on Image Processing*, 7(3), 1998.

[18] J.A. Sethian. Level set methods: An act of violence.

[19] Dengsheng Zhang; Guojun Lu. Review of shape representation and description techniques. *Pattern Recognition*, 37(1), 2003.

[20] Sven Loncaric. A survey of shape analysis techniques. *Pattern Recognition*, 31(8), 1998.

[21] Luciano da Fontoura Costa; Roberto M. Cesar Junior. *Shape Analysis and Classification*. CRC Press, 2000.

[22] Donald Hearn; M. Pauline Baker. *Computer Graphics with OpenGL*. Prentice Hall Professional Technical Reference, 2003.

[23] C. H. Jr. Edwards; David E. Penney. *Computer Graphics with OpenGL*. Prentice Hall, 1988.

[24] Lihong Zheng; Xiangjian He. Classification techniques in pattern recognition. Proceedings of The Thirteenth International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2005 (WSCG 2005), UNION Agency - Science Press, 2005.

[25] Anil Jain; Douglas Zongker. Feature selection: Evalutaion, application and small sample performance.

[26] Luigi Portinale; Lorenza Saitta. Feature selection, 2002.

[27] M. Dash; H. Liu. Feature selection for classification.

[28] S. Piramuthu. Evaluating feature selection methods for learning in data mining applications. *European Journal of Operational Research*, 156(2), 2004.

[29] Huan Liu; Lei Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17(4), 2005.

[30] H. Liu; L.Yu. Feature selection for data mining.

[31] Jens Strackeljan. Feature selection methods for softcomputing classification.

[32] Juha Reunanen. A pitfall in determining the optimal feature subset size. In *Proc. of the 4th Int. Workshop on Pattern Recognition in Information Systems (PRIS 2004)*, pages 176–185, Porto, Portugal, 2004.

[33] Selçuk Gören. Branch - and - bound procedures and beam search in scheduling.

[34] J. Yang; V. Honavar. Feature subset selection using a genetic algorithm. *IEEE Intelligent Systems And Their Applications*, 13(2), 1998.

[35] Manoranjan Dash; Huan Liu; Hiroshi Motoda. Consistency based feature selection. In *PADKK '00: Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Current Issues and New Applications*, pages 98–109, London, UK, 2000. Springer-Verlag.

[36] J.W. Foster III; P.M. Griffin; S.L. Messimer; J.R. Villalobos. Automated visual inspection: a tutorial. *Computers & Industrial Engineering*, 18(4), 1990.

[37] M. Fuss; P. Scharf; F.-J. Luecke. Image processing for automated visual surface inspection. *Proceedings of the SPIE - The International Society for Optical Engineering*, 3100, 1997.

[38] M. Bariani; R. Cucchiara; P. Mello; M. Piccardi. Data mining for automated visual inspection. 1997.

[39] Timothy S. Newman; Anil K. Jain. A survey of automated visual inspection. *Computer Vision and Image Understanding*, 61(2):231–262, 1995.

[40] E. Bayro-Corrochano. Review of automated visual inspection 1983 to 1993. i. conventional approaches. *Proceedings of the SPIE - The International Society for Optical Engineering*, 2055, 1993.

[41] R.T. Chin. Automated visual inspection: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-4(6), 1982.

[42] C. Fernandez; J. Suardiaz; C. Jimenez; P.J. Navarro; A. Toledo; A. Iborra. Automated visual inspection system for the classification of preserved vegetables. *ISIE 2002. Proceedings of the 2002 IEEE International Symposium on Industrial Electronics*, (1), 2002.

[43] YunKoo Chung; KiHong Kim. Automated visual inspection system of automobile doors and windows using the adaptive feature extraction. *Second International Conference on Knowledge-Based Intelligent Electronic Systems*, 1998.

[44] Masao Takatoo; Yoichi Takagi; Tadao Mori. Automated fabric inspection system using image processing technique. *Automated Inspection and High Speed Vision Architectures II*, 1004, 1988.

[45] G. Dounias; G. Tselentis; V. Moustakis. Machine learning based feature extraction for quality control in a production line. *Integrated Computer-Aided Engineering*, 8(4), 2001.

[46] M.L. Soborski. Machine vision inspection for improved quality and productivity in the food processing environment. *Food Processing Automation IV. Proceedings of the FPAC IV Conference*, 1995.

[47] G. Brown; P. Forte; R. Malyan; P. Barnwell. Perceptual feature based object recognition for automatic inspection. *Proceedings of the SPIE - The International Society for Optical Engineering*, 2064, 1993.

[48] A. Konig; H. Genther; M. Glesner; A. Korn; F. Quint; N. Waleschkowski; W. Henrich; M. Decker; M. Schahn. A generic dynamic inspection system for visual object inspection and industrial quality control. *IJCNN '93-Nagoya. Proceedings of 1993 International Joint Conference on Neural Networks (Cat. No.93CH3353-0)*, (2), 1993.

[49] R. Sablatnig. *A Highly Adaptable Concept for Visual Inspection.* PhD thesis, Vienna University of Technology, 1997.

[50] R. Sablatnig. Increasing flexibility for automatic visual inspection: the general analysis graph. *Machine Vision and Applications*, 12(4), 2000.

[51] R. Sablatnig. Flexible automatic visual inspection based on the separation of detection and analysis. *Proceedings of the 13th International Conference on Pattern Recognition*, 3(3), 1996.

[52] Wikipedia. Artificial neural network [online]. Available from: `http://en.wikipedia.org/wiki/Artificial_neural_network`.

[53] Microsoft visual studio developer center [online]. Available from: http://msdn.microsoft.com/vstudio/products/default.aspx.

[54] Doxygen [online]. Available from: http://www.stack.nl/~dimitri/doxygen/.

[55] W3C. Hypertext markup language (html) home page [online]. Available from: http://www.w3.org/MarkUp/.

[56] Umbrello uml modeller [online]. Available from: http://uml.sourceforge.net/index.php.

[57] Wikipedia. Unix [online]. Available from: http://en.wikipedia.org/wiki/Unix.

[58] Wikipedia. Unified modeling language [online]. Available from: http://en.wikipedia.org/wiki/Unified_Modeling_Language.

[59] Microsoft Norge. Microsoft office visio 2003 [online]. Available from: http://www.microsoft.com/norge/office/visio/prodinfo/overview.mspx.

[60] Texniccenter.org [online]. Available from: http://www.toolscenter.org/.

[61] Wikipedia. Microsoft windows [online]. Available from: http://en.wikipedia.org/wiki/Microsoft_windows.

[62] Latex: A document preparation system [online]. Available from: http://www.latex-project.org/.

[63] Microsoft Norge. Microsoft office excel 2003 [online]. Available from: http://www.microsoft.com/norge/office/excel/prodinfo/overview.mspx.

[64] Fast artificial neural network library (fann) [online]. Available from: http://leenissen.dk/fann/.

[65] Image processing library 98 [online]. Available from: http://www.mip.sdu.dk/ipl98/.

[66] Nlm insight segmentation & registration toolkit [online]. Available from: http://www.itk.org/.