Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

MASTEROPPGAVE

Studieprogram/spesialisering: Kybernetikk	Vårsemesteret, 2012 Konfidensiell
Student/studenter: Thomas Ivesdal-Tronstad	.... signatur(er)
Fagansvarlig: Kjersti Engan (UiS) Veileder(e): Kjersti Engan, Tom Ryen (UiS)	
Tittel på oppgaven: Klassifisering av typeskilt. Engelsk tittel: Classification of tag plates.	
Studiepoeng: 30	
Emneord: Klassifisering, Parzen-vinduer, kn-nærmeste-nabo, MLE, Bayes regel, Bildebehandling, Co-occurence-matrise, MATLAB, Statistikk, Interpolering	Sidetall: 51 + vedlegg/annet: CD-ROM Stavanger, 14 juni 2012

Sammendrag

Prosjektet går ut på å klassifisere bilder av typeskilt. Dette er løst ved å benytte en Bayes klassifiserer med normalfordelt klassespesifik tetthetsfunksjon sammen med en optimal egenskapskombinasjon. En egenskapskombinasjon som har gitt god ytelse: blokkbasert gjennomsnittspikselverdi av gradienten til bildene, se kap[5].

Innhold

Figurer

Tabeller

1 Introduksjon

Her følger en introduksjon til prosjektet. Hva hensikten med prosjektet er, og en generell oversikt av hvordan rapporten er satt sammen.

Oppgaven ble gitt av Verico AS høsten 2012, og er utført i samarbeid med Universitetet i Stavanger.

Takk til Verico AS, Veilederene, og fagansvarlig for all hjelpen de har bistått med gjennom prosjektet.

1.1 Om Verico AS

Basert på [3]:

Verico AS ble stiftet i 1999. De tilbyr IT-relaterte tjenester til andre firma innenfor fagområdene drift og vedlikehold.

Verico har store kunder innen elkraftbransjen som Lyse AS og Statnett.

I dag tilbyr Verico blant annet en fotobasert løsning for dokumentering og innsamling av data fra fysisk utstyr. I forbindelse med dette har Verico gitt flere bachelor- og masteroppgaver.

1.2 Hensikten med prosjektet

Prosessen med å lage en oversikt/database over detaljer om alt fysisk utstyr en bedrift eier kan være enormt omfattende. I dag foregår dette i stor grad manuelt. Det er ikke vanskelig å forstå at denne prosessen er svært arbeidskrevende og kan ta svært mange timer å fullføre for en bedrift. Formålet med oppgavene som Verico har gitt er å få automatisert de mest arbeidskrevende delene av denne prosessen.

Den fullstendige prosessen er delt opp i flere mindre deler, og hver av disse har blitt gitt som en bachelor- eller masteroppgave. Fig[2] viser en oversikt.

Delprosjektene er fordelt slik:

1. Lokalisering: parallelt.
2. Warp: ferdig.
3. Klassifisering: dette prosjektet.
4. OCR: ferdig.

Målet med klassifiseringstrinnet er å kunne velge ut riktig forhåndsdefinert mal som definerer områdene for avlesning av data (via OCR).

1.3 Problembeskrivelse

Oppgaven ble gitt med det relativt generelle målet "å klassifisere typeskilt". Et konkret hovedmål må defineres basert på det generelle.

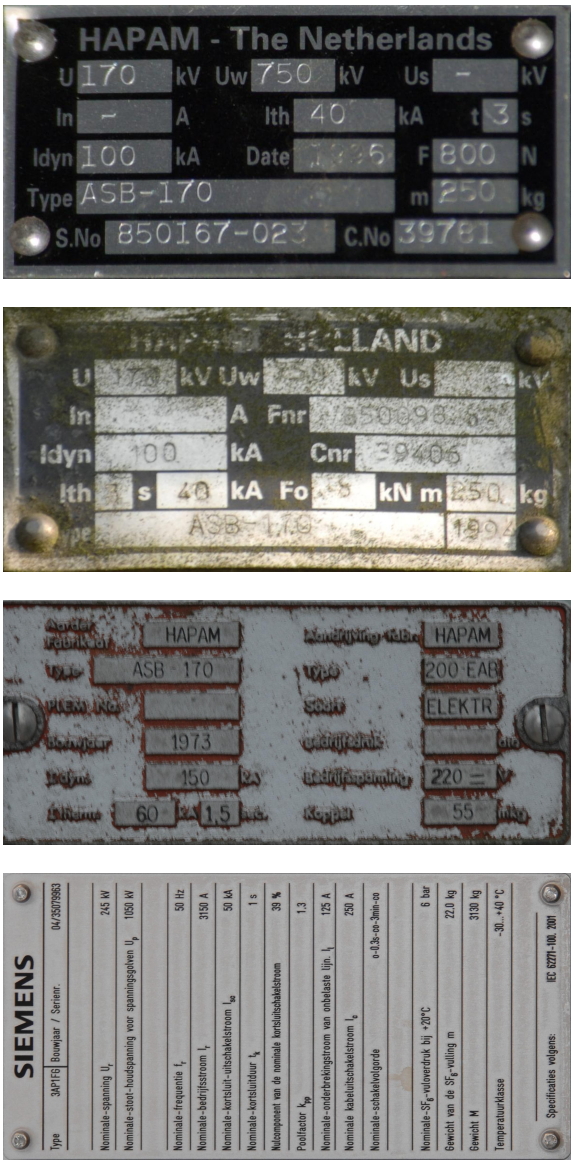
Dette prosjektet får data fra warping-steget som vist i fig[2]. Klassen som tilordnes et gitt bilde brukes så videre i OCR-steget i forbindelse med automatisk data-avlesning.

Hovedmålet konkretiseres ved å se på prosessen rundt:

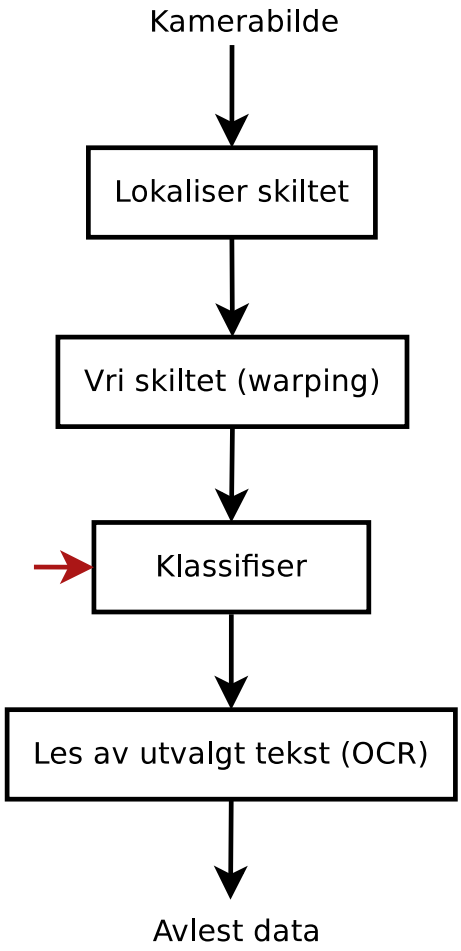
Warp Prosessen "warpinggir ut et rektangulært bilde med kun et skilt som motiv (ingen bakgrunn).

OCR OCR står for "Optical Character Recognition"og er et viktig trinn i en prosess for å lese for en datamaskin. Det antas i OCR prosjektet at koordinatene til teksten er kjent, med andre ord at det finnes en mal med felter for lesing manuelt definert.

For å kunne velge ut riktig mal for et gitt bilde må bildet først sorteres til en kjent kategori. Det er her klassifisering kommer inn.



Figur 1: Eksempler på skilt fra trinn 2 (warping).



Figur 2: Prosjektoversikt.

1.3.1 Hovedmål

I lys av hva som antas som input, og de resultatene som forventes ut, formuleres et konkret hovedmål.

Inn antas det at bildene er utelukkende av skilt med ingen annen bakgrunn, i tillegg antas det også at bildet er blitt tatt perpendikulært (skiltet står rett i bildet). Fig[1] viser noen eksempler på dette.

Neste delprosjekt, OCR, velger ut en mal basert på resultatet fra klassifiseringen. Malen beskriver de korrekte områdene for avlesing av data fra skiltet. Plasseringen av disse områdene vil naturligvis gå igjen i samme type skilt.

Med disse premisser kan et konkret hovedmål formuleres.

Hovedmålet med dette klassifiseringsprosjektet er å utvikle en metode for å automatisk velge ut en passende mal til et gitt skilt, ettersom det er antatt at koblingen mellom en mal og en skilttype allerede finnes vil hovedmålet begrense seg til:

Å klassifisere skilt til korrekt type, der typen gjenspeiler plasseringen av datafeltene på skiltet.

Merknad Pga. dagens implementering av trinn 2 har jeg basert på dagens løsning implementert min egen. Valget falt på å lage min implementering fordi jeg regnet med å kun bruke et par dager til sammen på dette, og det virket som den raskeste veien å gå.

Dagens implementasjon av trinn 2 er designet for å manuelt velge ut korresponderende punkter fra kildebildet. Det er da benyttet et referansebilde for å vise hvor disse punktene skal ligge [3].

Prosjektet er definert til å starte ved ferdig warpede bilder, for å konstruere datasettene som trengs har jeg derfor tatt et lite steg bakover i forkant av utførselen av prosjektet.

Se app[A] for detaljer.

1.4 Avgrensninger

For å danne en ramme for prosjektet er følgende antakelser gjort:

- Bildenes motiv er antatt å være utelukkende av skiltene, og tatt perpendikulært i forhold til skiltet.
- Antallet typer skilt er antatt å være stor.
- Skiltene antas å være av god kvalitet (ikke malt over eller redusert av vær og vind).
- Dimensjonene til bildene er ikke like, også innenfor samme klasse kan størrelsesforholdene være forskjellig.
- Lysforholdene er antatt å variere innenfor samme klasse.

1.5 Organisering av rapporten

Rapporten er delt opp i flere distinkte kapitler med hver sin funksjon.

1. **Teori** - Gjennomgang av all relevant teori i forbindelse med oppgaven og eksperimentene.
2. **Forhåndsprosessering av datasett** - Preparering og oppsett av forskjellige datasett.
3. **Egenskaper** - Beskrivelse av egenskapene.
4. **Implementering** - Implementeringsdetaljer.
5. **Eksperiment** - Definering av eksperimenter og presentasjon av resultater.
6. **Diskusjon** - Reflektering av resultater og erfaringer gjort i løpet av gjennomføringen av prosjektet.

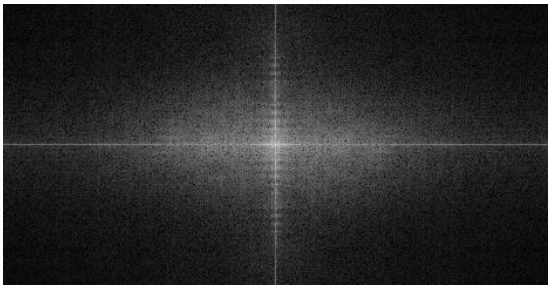
Kildehenvisninger er slik: [9].
Ligninger refereres til slik: ligning[1].
Kapiter og underkapitler refereres til slik: kap[3].
Appendix refereres til slik: app[C].
Figurer og plott refereres til slik: fig[3].
Tabeller refereres til slik: tab[1].

1.6 Ord og begreper

Listing av ord og begreper brukt i rapporten.

- **A posteriori sannsynlighet** - $P(\omega|\underline{x})$, sannsynligheten for at en egenskapsvektor $\underline{x}$ tilhører klassen ω.
- **A priori sannsynlighet** - $p(\underline{x}|\omega)$, tetthetsfunksjonen for klassen ω, estimerer klassespesifik sannsynlighet.
- **Desisjonsgrense** - Grensen i egenskapsrommet som skiller mellom to klasser, et klassifiseringssystem kan ha flere desisjonsgrenser.
- **Desisjonsregel** - Denne regelen beskriver hvordan klassifiseringen foregår.
- **Diskriminantfunksjon** - Funksjonen som gir ut et mål basert på egenskaper beregnet for et gitt bilde. Denne brukes så sammen med Desisjonsregelen til å klassifisere et objekt.
- **Egenskapsrom** - Det euklidiske rommet som inneholder egenskapsvektorene til objekter. Dimensjonen til dette rommet er lik antall egenskaper i egenskapsvektoren.
- **Egenskapsvektor** - Et eller flere mål/størrelser som kan beregnes/estimeres fra et objekt som kan brukes til å korrekt klassifisere objektet.
- **Klasse** - En gruppe av objekter som hører sammen definerer en klasse.
- **Klassespesifik sannsynlighet** - Se *A priori sannsynlighet*.
- **Klassifisering** - Å sortere/gruppere et objekt mellom kjente grupper/klasser.
- **Objekt** - Objekt referer til gjenstanden som skal klassifiseres, i dette prosjektet er dette motiv fra bilder.
- **Prosjektet** - Refererer til dette delprosjektet.
- **Tilordnet klasse** - Klassen et bilde klassifiseres til.

(a) Eksempelbilde.



(b) Fouriertransformen til bildet til venstre, kun amplituder.

Figur 3: Eksempel på en fouriertransform, skalert ved å ta logaritmen til hvert piksel. DC-komponenten er her plassert i sentrum vha. MATLAB's fftshift funksjon.

2 Teori

Her er en oppsummerende gjennomgang av relevant teori innenfor mønstergjenkjenning og bildebehandling.

Hovedmålet med denne oppgaven løses ved hjelp av en klassifiseringsprosess. Denne prosessen inneholder disse sentrale blokkene:

1. Bildebehandling
2. Beregning av egenskaper
3. Klassifiser bilder vha. egenskapene

Jeg går først gjennom rene bildebehandlingsalgoritmer og filtre, og til slutt går jeg inn på teori angående mønstergjenkjenning.

All teori er hentet fra diverse kilder,oversatt og gjengitt her.

2.1 Fouriertransformen

Den todimensjonale diskrete fouriertransformen er definert slik (DFT2) [5]:

$$\mathcal{F}(u,v) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I(m,n)\mathsf{e}^{-j2\pi(um/M+vn/N)} \tag{1}$$

Her starter indekseringen av bildet på null.

Frekvensbildet består av komplekse tall som representerer amplituden og fasen til forskjellige frekvenser. I punktet $(1,1)$ ligger DC-frekvensen, frekvensene øker så utover frekvensbildet i begge retninger.

Fig[3] viser et eksempel på et frekvensspektrum. Det viser kun styrkene til de forskjellige frekvensene ved at absoluttverdien er beregnet for hver frekvens. De to komponentene er funnet slik:

$$\mathcal{F}_{mag}(u,v) = \mathsf{abs}\left[\mathcal{F}(u,v)\right] \tag{2}$$

$$\mathcal{F}_{ang}(u,v) = \arctan\left[\frac{\mathcal{F}_{imag}(u,v)}{\mathcal{F}_{real}(u,v)}\right] \tag{3}$$

En effektiv algoritme for beregning av DFT kalt FFT deler DFTen opp i stadig mindre biter. Det gir en betydelig hastighetsforbedring ved beregning av DFTen. Se [4] for mer om FFT.

Det er FFT som er implementert i MATLAB for beregning av fouriertransformen [2].

2.2 Massesenter

I fysikken er massesenteret til et system av partikler definert på følgende vis (fra side 306-307 i [9]):

$$\underline{\mathsf{R}} = \frac{1}{M}\sum_i m_i \underline{\mathsf{r}}_i \tag{4}$$

Der $\underline{\mathsf{R}}$ er punktet for massesenteret, M total masse, m_i massen til partikkel i, og $\underline{\mathsf{r}}_i$ er posisjonen til partikkel i.

Vi tenker oss så at pikslene i et bilde er partikler plassert i et rutenett med masse lik pikselverdiene. Da kan vi beregne et slags massesenter for bildet.

For bilder blir da massesentrum i begge retninger slik:

$$x_m = \frac{\sum_{m=1}^{M}\sum_{n=1}^{N} I(m,n)\cdot m}{\sum_{m=1}^{M}\sum_{n=1}^{N} I(m,n)} \tag{5}$$

$$x_n = \frac{\sum_{m=1}^{M}\sum_{n=1}^{N} I(m,n)\cdot n}{\sum_{m=1}^{M}\sum_{n=1}^{N} I(m,n)} \tag{6}$$

Her er det fare for å dele med 0 i tilfeller der bildet er helt svart (et system av masseløse partikler i fysikken).

2.3 Lineær romlig filtrering - bilder

De filtrene som for hver piksel i utgangsbildet beregner verdien fra et nabolag i inngangsbildet kalles for romlige filtre".

De lineære romlige filtrene implementeres vha. maskeoperasjoner.

Noen romlige filtre:

- Uniformt lavpassfilter.
- Gaussisk lavpassfilter.
- Sobel operator.

Maskeoperasjoner Hver pikselverdi i utgangsbildet beregnes fra et nabolag i inngangsbildet og masken vha. korrelasjon [5].

Korrelasjonen beregnes som summen av produktene av maskevektene og de korresponderende inngangspikselverdiene.

$$w(x,y) \star f(x,y) = \sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t) f(x+s, y+t) \tag{7}$$

Der w(x,y) er et element fra filtermasken, f(x,y) er et piksel fra inngangsbildet, når masken har størrelsen $[m,n]$ er $[a,b] = [(m-1)/2, (n-1)/2]$. Her antar vi at inngangsbildet har blitt *paddet* tilstrekkelig.

Rekkefølgen har betydning for resultatet, for å utføre korrelasjonen riktig er det filtermasken som skal transleres over bildet (se [5] for detaljer).

Eksempel fra fig[4] Anta at masken har følgende vekter:

Da regnes verdien for utgangspikselet $O(3,3)$ (samme posisjon som det blå senterpikselet) ut slik:

$$\begin{aligned} O(3,3) = & \\ & \frac{1}{9}14 + \frac{1}{9}8 + \frac{1}{9}12 + \\ & \frac{1}{9}10 + \frac{1}{9}9 + \frac{1}{9}13 + \\ & \frac{1}{9}14 + \frac{1}{9}7 + \frac{1}{9}12 \\ & = \underline{11} \end{aligned}$$

15	10	12	8	10	2
15	14	8	12	20	18
20	10	9	13	9	0
11	14	7	12	8	13
12	12	10	5	6	7
8	5	10	15	16	2

$\frac{1}{9} \cdot$

1	1	1
1	1	1
1	1	1

Figur 4: Bilde med pikselverdier og en $3x3$ maske (senter er blå, naboer røde).

Tallene er tatt fra fig[4].
Denne summen av produkter utføres for hvert piksel i resultatbildet.
Masken trenger ikke nødvendigvis å være firkantet, rektangulær, eller symmetrisk.

Padding Når en maskeoperasjon skal utføres langs kantene i et bilde oppstår det et spesialtilfelle der vi leser av pikselverdier utenfor bildet. For å gjøre dette mulig må bildet paddes", dvs. utvides tilstrekkelig. Hva som skal puttes inn som verdier utenfor bildet varierer med hvilken padde-metode som brukes.

Det finnes flere metoder å velge mellom, her er noen av dem:

- **Zero-fill** - fyll inn med nuller.
- **One-fill** - fyll inn med enere (maks pikselverdi).
- **Gjenta** - gjenta pikselverdiene til kanten av bildet utover perpendikulært med kanten.
- **Speiling** - speil pikselverdier fra bildet symmetrisk om kanten av bildet.

Disse har alle sine bruksområder. "Zero-fill"og "One-fill"vil for eksempel gi best resultat henholdsvis i et max-/min- filter, mens Gjenta"og Speiling"vil virke bedre for lavpassfiltre siden resultatet da ikke dras mot svart (0) eller hvitt (1). [2] og [5]

2.3.1 Uniformt lavpassfilter (boxcar)

Et uniformt lavpassfilter beregner gjennomsnittet av pikselverdiene i et nabolag i inngangsbildet. Det er tilsvarende et filter kalt boxcar-filter"innen signalbehandling. [4]

En $3x3$ uniform maske ser slik ut:

$$\frac{1}{9} \cdot \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} \tag{8}$$

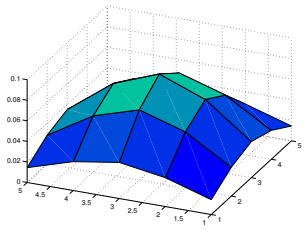
Den kan også ta andre former. Alle elementene i masken har samme verdi, og summen av dem skal være lik 1 for å unngå å tilføre eller trekke fra energi fra bildet.

Størrelsen på filteret avgjør hva knekkfrekvensen til filteret er. Større filter gir lavere knekkfrekvens, det gir kraftigere glatting/filtrering av bildet.

Som nevnt er dette filteret veldig simpelt og det gir ikke et spesielt godt resultat, som regel er det bedre å heller bruke et gaussisklavpassfilter. I applikasjoner der dimensjonen til et bilde økes er et boxcarfilter nyttig for interpolering.

2.3.2 Gaussisk lavpassfilter

Et gaussisk lavpassfilter har samme karakteristikk som den uniforme varianten ved at summen av maske-elementene er lik 1, og alle elementene er positive. Forskjellen er at det legges mer vekt på et element des nærmere senter det er. Elementene til masken tar verdier fra en normalfordeling med en spesifisert varians.



Figur 5: Illustrasjon av et gaussisk lavpassfilter.

Større varians legger mer vekt på elementene lengre vekk fra senter, mens en lavere varians legger mer vekt på elementene om maskesentrum.

Verdiene for masken er normalfordelt med middelverdi lik senter av masken $\begin{bmatrix} m \\ n \end{bmatrix}$ og en varians Σ:

$$\omega_{m,n}(\underline{x}) = N(\underline{x}| \begin{bmatrix} m \\ n \end{bmatrix}, \Sigma) \tag{9}$$

Der $\underline{x}$ er en maskekoordinat.

Summen av en normalfordeling er lik 1. Siden vi ikke tar med en hel normalfordeling, men heller sampler den ved heltallskoordinater, bør masken skaleres for at den skal få en sum lik 1. Etter å ha fylt ut en hel maske med tall direkte fra en normalfordelingsfunksjon kan en skalering utføres ved å dele hvert element på summen av den uskalerte masken. Da vil summen til masken etter skalering bli lik 1.

En ordentlig skalert gaussisk lavpass-maske av størrelse $5x5$ med $\Sigma = \begin{bmatrix} 1.5 & 0.0 \\ 0.0 & 1.5 \end{bmatrix}$ ser slik ut:

0.0144	0.0281	0.0351	0.0281	0.0144
0.0281	0.0547	0.0683	0.0547	0.0281
0.0351	0.0683	0.0853	0.0683	0.0351
0.0281	0.0547	0.0683	0.0547	0.0281
0.0144	0.0281	0.0351	0.0281	0.0144

Fig[5] viser et plott av denne masken.

Dette filteret gir som regel et bedre resultat enn et uniformt lavpassfilter da det legger mer vekt på naboer nærtliggende senteret.

I forhold til det uniforme glattefilteret vil ikke størrelsen nødvendigvis ha like stor effekt på knekkfrekvensen til filteret, et gaussisk lavpassfilter er også avhengig av parameteren Σ. Større verdi på Σ gir kraftigere glatting/filtrering og størrelsen på filteret vil da også bety mer.

Dersom $\Sigma \to \infty$ vil et gaussisk lavpassfilter gå mot et uniformt lavpassfilter av samme størrelse.

2.3.3 Sobel-operatoren

Sobel-operatoren estimerer gradienten til en posisjon i bildet. Dette er en høypassfiltrering. Filteret deriverer bildet i begge retninger ved en diskret approksimasjon. Derivatet estimeres i et punkt som differansen mellom de to naboene langs en retning.

For å estimere de partiell-deriverte i begge retningene kan følgende to masker benyttes:

−1	0	1

og

−1
0
1

(10)

Sobel-operatoren bruker et større nabolag og ser slik ut ([5] og [2]):

$$\begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \quad \text{og} \quad \begin{array}{|c|c|c|} \hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array} \tag{11}$$

Resultatene fra filtrering med maskene over gir en tilnærming til derivatet av inngangsbildet i henholdsvis horisontal og vertikal retning.

De to resultatene kan så kombineres videre for å finne styrke og vinkel til gradienten til bildet.

$$O_{mag}(m,n) = \sqrt{g_x^2 + g_y^2} \tag{12}$$

$$O_{ang}(m,n) = \arctan\left[\frac{g_y}{g_x}\right] \tag{13}$$

Der g_x og g_y er pikselverdier fra et gradientbilde i henholdsvis horisontal og vertikal retning ved punktet (m,n).

Gradient av fargebilder Sobel-operatoren fungerer fint for gråskala-bilder med kun en fargekanal. Fargebilder har derimot 3 individuelle bildeplan, for å estimere gradienten til et slikt bilde med en Sobel-operator må bildet først gjøres om til gråskala.

Et bedre resultat fås ved å beregne vektorgradienten i stedet [5]:

$$\bigtriangledown I(P) = \frac{\partial I}{\partial x}\underline{\mathtt{m}} + \frac{\partial I}{\partial y}\underline{\mathtt{n}} \tag{14}$$

Der I(P) er et piksel ved punkt P.

Den deriverte i hver retning estimeres og summeres for hver farge.

$$g_{xx} = \left|\frac{\partial R}{\partial x}\right|^2 + \left|\frac{\partial G}{\partial x}\right|^2 + \left|\frac{\partial B}{\partial x}\right|^2$$

$$g_{yy} = \left|\frac{\partial R}{\partial y}\right|^2 + \left|\frac{\partial G}{\partial y}\right|^2 + \left|\frac{\partial B}{\partial y}\right|^2$$

$$g_{xy} = \frac{\partial R}{\partial x}\frac{\partial R}{\partial y} + \frac{\partial G}{\partial x}\frac{\partial G}{\partial y} + \frac{\partial B}{\partial x}\frac{\partial B}{\partial y}$$

Gradienten finnes da ved hjelp av dekomponering.

$$\varphi = \frac{1}{2}\arctan\left(\frac{2g_{xy}}{g_{xx} - g_{yy}}\right) \tag{15}$$

$$\bigtriangledown I(P) = \sqrt{\frac{1}{2}\left[(g_{xx} + g_{yy}) + (g_{xx} - g_{yy})\cos(2\varphi) + 2g_{xy}\sin(2\varphi)\right]} \tag{16}$$

Gradienten beregnes to ganger, en for φ og $\varphi + \frac{\pi}{2}$. De to resultatene kombineres ved å velge største verdi fra hver posisjon, $\bigtriangledown I(P) = max[\bigtriangledown_1 I(P), \bigtriangledown_2 I(P)]$. (Se `colorgrad` fra appendix C i [5]).

2.3.4 Mer om filtre

Lavpassfiltre og høypassfiltre er nært beslektet og den ene kan implementere den andre og omvendt.

Dersom h_{lav} er et lavpassfilter, $h_{høy}$ et høypassfilter, og I er et bilde gjelder følgende forhold:

$$h_{høy} \star I = I - h_{lav} \star I \tag{17}$$

$$h_{lav} \star I = I - h_{høy} \star I \tag{18}$$

Et lavpassfilter kan brukes for å høypassfiltrere et bilde og omvendt. På denne måten kan kanter og høye frekvenser fremheves med et lavpassfilter.

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

Figur 6: Bilde.

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

Figur 7: "Co-occurencematrisen til bildet (hele dette eksempelet er tatt direkte fra [5] side 647).

Alternativ til korrelasjon/konvolusjon Et alternativ til korrelasjon og konvolusjon for utføring av maskeoperasjoner er å gå over til frekvensplanet, der vi heller multipliserer i stedet for å korrelere eller konvulvere.

$$h_{lav} \star I = H_{lav}\mathcal{F}\{I\} \tag{19}$$

Der H_{lav} er fouriertransformerte $\mathcal{F}\{h_{lav}\}$ versjonen av lavpassfilteret h_{lav}.

Avhengig av hvor rask implementeringen av fouriertransformen er og størrelsen på filteret, kan følgende prosess gi raskere filtrering enn via korrelasjon eller konvolusjon:

1. Fouriertransformer både bildet og filtermasken.
2. Multipliser bildet og filtermasken.
3. Invers-fouriertransformer resultatet.

2.4 Teksturegenskaper - "Co-occurencematrise

En måte å beskrive en region i et bilde er ved å beregne forskjellige mål av teksturene i området. En "co-occurencematrise er et verktøy som kan brukes til å estimere noen statistiske mål av teksturene [5].

"Co-occurencematrisen er fylt med data om hvor ofte spesifike nabolag forekommer i bildet. I likhet med histogrammet kan en "co-occurencematrise sees på som et estimat av en tetthetsfordeling, til forskjell fra histogrammet handler "co-occurencematrisen om mønstre (i motsetning til histogrammet som kun inneholder informasjon om sannsynligheten for enkeltpiskelverdier). Dersom matrisen normaliseres vil elementene fra matrisen tilsvare estimater av sannsynligheten for å finne de forskjellige mønstrene i bildet.

Fig[6] og fig[7] viser hvordan "co-occurencematrisen er bygget opp. En teller opp antall like kombinasjoner av naboer som forekommer. For eksempel i fig[6] er det 3 stykker av 6 2 nederst i høyre hjørne, dette gir elementet $(6, 2) = 3$ i matrisen. Denne matrisen er alltid kvadratisk.

Flere statistiske mål kan beregnes fra denne matrisen, jeg går her raskt gjennom 6 slike (de 4 første er implementert i MATLABs image processing toolbox [2]).

Anta følgende symboler:

- $\underset{\sim}{G}$ er "co-occurencematrisen
- K er størrelsen på $\underset{\sim}{G}$ (antall intensitetsnivåer)
- n er summen av alle elementene i $\underset{\sim}{G}$
- $p_{i,j}$ er elementet ved (i, j) i $\underset{\sim}{G}/n$

Følgende mål kan da beregnes fra $\underset{\sim}{G}$ (gjengitt direkte fra [5]).

Kontrast Dette er et mål av forskjellen i intensitet mellom to nabopiksler i hele bildet.

$$\sum_{i=1}^{K}\sum_{j=1}^{K}(i-j)^2 p_{i,j} \tag{20}$$

Skala: $[0,(K-1)^2]$. Kontrasten er lik 0 for et konstant bilde.

Korrelasjon Dette er et mål på hvor korrelert et piksel er til dets nabo i hele bildet.

$$\sum_{i=1}^{K}\sum_{j=1}^{K}\frac{(i-m_r)(j-m_c)p_{i,j}}{\sigma_r\sigma_c} \tag{21}$$

$$\sigma_r \neq 0 \quad \sigma_c \neq 0$$

Der:

$$m_r = \sum_{i=1}^{K} iP(i) \qquad \sigma_r^2 = \sum_{i=1}^{K}(i-m_r)^2 P(i)$$

$$m_c = \sum_{j=1}^{K} jP(j) \qquad \sigma_c^2 = \sum_{j=1}^{K}(j-m_c)^2 P(j)$$

$$P(i) = \sum_{j=1}^{K} p_{i,j} \qquad P(j) = \sum_{i=1}^{K} p_{i,j}$$

Skala: $[-1,1]$. Korrelasjon er ikke definert for et konstant bilde ($\sigma_r = 0$ og $\sigma_c = 0$).

Energi Dette er summen av alle elementene i $\underset{\sim}{G}$ kvadrert.

$$\sum_{i=1}^{K}\sum_{j=1}^{K} p_{i,j}^2 \tag{22}$$

Skala: $[0,1]$. Energien er lik 1 for et konstant bilde.

Homogenitet Dette er et mål på hvor nærme elementene til $\underset{\sim}{G}$ ligger diagonalen.

$$\sum_{i=1}^{K}\sum_{j=1}^{K}\frac{p_{i,j}}{1+|i-j|} \tag{23}$$

Skala: $[0,1]$. Homogeniteten til en diagonal $\underset{\sim}{G}$ er lik 1.

Sterkeste respons Dette er andelen av mønsteret av nabopiksler som oftest går igjen i bildet.

$$\max\left[\underset{\sim}{G}/n\right] \tag{24}$$

Skala: $[0,1]$. Lik 1 for et konstant bilde.

Entropi Dette er et mål på tilfeldighetentil pikselmønstrene i bildet.

$$-\sum_{i=1}^{K}\sum_{j=1}^{K} p_{i,j}\log_2\left[p_{i,j}\right] \tag{25}$$

Skala: $[0,\log_2\left(K^2\right)]$. Entropien for et konstant bilde er lik 0, og entropien for et bilde med hvit støy er (tilnærmet) lik $\log_2\left(K^2\right)$.

2.5 Hva er en egenskap?

For å skille et objekt fra andre objekter vil et menneske studere dem for å finne unike karakteristikker som så benyttes til å tilordne objektet en klasse. Disse karakteristikkene er det som i mønstergjenkjennings-sammenheng kalles for en egenskap.

En egenskap kan i prinsippet være et hvilket som helst mål beregnet fra objektet. Eksempel på egenskaper:

- Gjennomsnittspikselverdi.
- Varians/standardavvik.
- Høyere-ordens statistiske moment (gjennomsnitt og varians er 1. og 2. ordens henholdsvis).
- Tyngdepunkt.
- Kantinformasjon.
- Teksturkarakteristikker/statistikker (kontrast, energi, korrelasjon, homogenitet, maximum, entropi).
- Frekvensanalyser (energi i frekvensbånd,).
- ...

Det er som regel nødvendig med flere enn en egenskap for å få god ytelse. Flere egenskaper kan settes sammen til en egenskapsvektor som brukes direkte i en klassifiserer.

Ideelle egenskaper er:

- Tilnærmet like for objekter av samme klasse (lav varians).
- Svært ulike for alle andre objekter (god separabilitet).

Det er en fordel om egenskapene er skalert til ca. samme skala. Dette er for å unngå at en av egenskapene får større betydning enn de andre.

2.6 Klassifisering

Klassifisereren bestemmer klassentilhørigheten til et objekt basert på egenskapsvektoren beregnet fra objektet. Jeg utleder en del av prinsippene bak klassifiserere, men tar en del hopp for å fatte meg i korthet. Se [6] for en mer detaljert gjennomgang.

Alt gjengitt her i dette underkapittelet er basert på teori fra [6], kapitlene 2-4.

2.6.1 Diskriminantfunksjonen

For å skille mellom to klasser, og for å kunne bruke en desisjonsregel for å komme frem til en avgjørelse om klassetilhørighet, trenger vi å "måleet objekt på et vis. Her kommer diskriminantfunksjonen inn, den er en generell funksjon som kan defineres til flere former.

Hensikten med en diskriminantfunksjon er å beregne eller estimere noen faste mål gitt et objekt. Målet fra diskriminantfunksjonen brukes direkte til å bestemme klassetilhørighet til objektet (derav navnet, diskriminerer, eller skiller, mellom objekter). Navnet *diskriminantfunksjon* betegner en rolle, ikke en bestemt funksjon. Hva som kan være en diskriminantfunksjon er opp til designeren av klassifiseringssystemet.

I dette prosjektet har jeg tatt en statistisk tilnærming. Jeg bruker bayes lov til å beskrive sannsynligheten for at en gitt egenskapsvektor $\underline{x}$ tilhører den spesifiserte klassen ω_j i et spesifisert utvalg av klasser ω.

Bayes lov definerer forholdet:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} \tag{26}$$

Her er A og B begge stokastiske variabler. $P(B|A)$ leses sannsynligheten for B gitt A"[1].

Dersom vi setter inn for A og B:

$$g_j(\underline{x}) = P(\omega_j|\underline{x}) = \frac{p(\underline{x}|\omega_j)P(\omega_j)}{P(\underline{x})} \tag{27}$$

Her er ω_j klasse nr. j, $\underline{x}$ en egenskapsvektor, og $p(\underline{x}|\omega_j)$ den klassespesifike tetthetsfunksjonen for klasse j.

2.6.2 Anatomien til en bayes klassifiserer

En bayes klassifiserer består av tre sentrale komponenter:

- $p(\underline{x}|\omega_j)$ - klassespesifik tetthetsfordeling".
- $P(\omega_j)$ - "a priori sannsynlighet".
- $P(\omega_j|\underline{x})$ - "a posteriori sannsynlighet".

Nevneren, $P(\underline{x})$, er ikke så viktig å ta med i en klassifiseringssammenheng siden den bidrar likt for alle klasser. Det er vanlig å forenkle diskriminantfunksjoner på denne måten, altså ved å betrakte hvilke deler av utrykket som er uavhengig av klassene.

Klassespesifik tetthetsfordeling, $p(\underline{x}|\omega_j)$ I en klassifiseringssituasjon vil en generelt øke ytelsen ved å tilføre klassifiseringssystemet informasjon. Den klassespesifike tetthetsfordelingen representerer i den sammenhengen erfaring.

Gitt at vi har et sett med objekter som vi vet de sanne klassene til, da kan vi benytte denne informasjonen til å treneklassifisereren. Ved å beregne egenskapsvektoren for dette datasettet kan en tetthetsfunksjon estimeres. Dette er den klassespesifike tetthetsfunksjonen, $p(\underline{x}|\omega_j)$.

Den klassespesifike tetthetsfordelingen er med andre ord "maskinlæringsdelen"av klassifiseringssystemet. Her lærer klassifiseringssystemet å kjenne igjen mønstre i objekter vha. egenskaper.

A priori sannsynlighet, $P(\omega_j)$ Som det kommer av ordet; a priori sannsynlighet omfatter den informasjonen vi har på forhånd, det vi vet om et objekt før vi observerer det.

Utrykket $P(\omega_j)$ tilsier sannsynligheten for at et gitt objekt tilhører klasse nr. j uten å beregne noen egenskaper.

Dersom vi for eksempel i en klassifiseringssituasjon vet at klasse nr. j forekommer i ca. 10% av objektene er det naturlig å sette $P(\omega_j) = 0.1$.

Summen av alle a priori sannsynligheter i et klassifiseringssystem bør være lik 1.

$$\sum_{j=1}^{c} P(\omega_j) = 1.0 \tag{28}$$

Dette impliserer at vi antar at et gitt objekt definitivt tilhører en av klassene i systemet vårt.

A posteriori sannsynlighet, $P(\omega_j|\underline{x})$ For å kombinere informasjonen fra egenskapene og den bakgrunnsinformasjonen vi har (a priori sannsynlighet) benyttes Bayes lov (ligning[26]). Resultatet her kalles "a posteriori sannsynlighet"og kan brukes som en diskriminantfunksjon.

Som nevnt tidligere er det vanlig å forenkle deler av diskriminantfunksjoner ved å kutte klasseuavhengige elementer, dersom vi ikke har noen a priori informasjon å tilføre er det vanlig å sette $P(\omega_j$ lik for hver klasse. I slike tilfeller kan denne faktoren regnes som klasseuavhengig og dermed kan den kuttes. På lik linje kuttes også $P(\underline{x})$, og vi står da igjen med $P(\omega_j|\underline{x}) = p(\underline{x}|\omega_j)$ (altså kun tetthetsfunksjonen).

2.6.3 Desisjonsregel

En klassifiserer utfører selve klassifiseringen ved hjelp av en desisjonsregel. Denne regelen er tilpasset etter hva slags type diskriminantfunksjon som brukes.

Desisjonsregelen som brukes i denne oppgaven er som følger:

$$\text{Velg } \omega_i \text{ hvis } g_i(\underline{\mathrm{x}}) > g_j(\underline{\mathrm{x}}) \quad \text{for alle } j \neq i \tag{29}$$

Dette er såkalt "minimum-error-rateklassifisering når vi snakker om en Bayes klassifiserer, dvs. vi antar at å feilklassifisere er like kostbart for alle klasser.

Det går også an å benytte en kostnadsfunksjon som tar hensyn til kostnadene ved feilklassifisering for hver klasse, dette kalles for "minimum-costklassifisering.

2.6.4 Tetthetsfunksjoner og trening av en Bayes klassifiserer

Den klassespesifike tetthetsfunksjonen estimeres ved å beregne egenskapsvektorer fra treningsdata som vi vet den sanne klassetilhørigheten for. Prosessen å estimere denne tetthetsfordelingen kalles "å trene klassifisereren".

Det er to hovedgrupper estimatorer for tetthetsfunksjoner:

- Parametriske metoder.
- Ikke-parametriske metoder.

Jeg går gjennom en parametrisk metode (normalfordelinger) og 2 ikke-parametriske metoder (parzen-vinduer og kn-nærmeste-naboer).

2.7 Parametriske metoder

Antar vi en konkret form for tetthetsfordeling er vi innenfor parametriske metoder.

Dette er en enkel måte å benytte oss av eventuell informasjon vi har om hvordan egenskapene er fordelt (normalfordeling, uniform osv.).

I veldig mange tilfeller vil egenskapene være normalfordelt, den multivariable normalfordelingen er definert som følger:

$$p(\underline{\mathrm{x}}|\omega_j) = \frac{1}{(2\pi)^{\frac{d}{2}}|\underset{\sim}{\Sigma}_j|^{\frac{1}{2}}} \mathrm{e}^{-\frac{1}{2}(\underline{\mathrm{x}}-\underline{\mu}_j)^T \underset{\sim}{\Sigma}_j{}^{-1}(\underline{\mathrm{x}}-\underline{\mu}_j)} \tag{30}$$

Der ω_j impliserer tetthetsfunksjonen til klasse nr. j, d er antall egenskaper (dimensjonen til tetthetsfunksjonen), $\underline{\mu}_j$ og $\underset{\sim}{\Sigma}_j$ er snitt og varians, og $\underline{\mathrm{x}}$ er en egenskapsvektor.

For å estimere parametrene til en parametrisk tetthetsfordeling kan vi benytte såkalt "maximum likelihood estimation", eller MLE, til å finne estimatorer for hver av parametrene. ML estimatorene for en multivariabel normalfordeling er som følger:

$$\underline{\mu} = \frac{1}{n}\sum_{i=1}^{n} \underline{\mathrm{x}}_i \tag{31}$$

$$\underset{\sim}{\Sigma} = \frac{1}{n-1}\sum_{i=1}^{n} (\underline{\mathrm{x}} - \underline{\mu})^T (\underline{\mathrm{x}} - \underline{\mu}) \tag{32}$$

Estimatoren for $\underset{\sim}{\Sigma}$ er her modifisert til å være "unbiased"(estimatoren funnet direkte fra ML-metoden gir en liten systematisk feil). Faktoren $\frac{1}{n-1}$ er endret fra $\frac{1}{n}$.

2.8 Ikke-parametriske metoder

Når vi ikke vet hvordan egenskapene er fordelt er nødvendig å estimere selve tetthetsfordelingen inkludert form.

To mye brukte metoder:

- Parzen-vinduer
- Kn-nærmeste-naboer

Disse metodene tar utgangspunkt i følgende estimat for den ukjente tetthetsfordelingen (se [6] for en utdeldning):

$$p(\underline{x}) \approx p_n(\underline{x}) = \frac{k_n/n}{V_n} \tag{33}$$

Her plasseres et volum V_n oppå hvert av treningsdataene (egenskapsvektorene). k_n er antallet treningsdata tilhørende klasse j som havner innenfor dette volumet, og n er totalt antall treningsdata innenfor det samme volumet. $p_n(\underline{x})$ er et estimat av den sanne tetthetsfordelingen $p(\underline{x})$ med n stykker treningsdata.

Trening av klassifisereren her begrenser seg til å beregne og ta vare på egenskapsvektorer for hele treningssettet.

Ikke-parametriske metoder har en tendens til å trenge flere treningsdata enn parametriske metoder, dette er fordi vi har mindre informasjon om tetthetsfordelingen og dermed trenger mer informasjon fra treningsdata.

Parzen-vinduer Dette er en måte å angripe ligning[33] på.

Anta at vi har et konstant volum $V_n = h_n^d$, der d er dimensjonen (antall egenskaper). Ved å benytte en vindus-funksjon (som definerer et volum i egenskapsrommet) kan vi finne et analytisk uttrykk for antallet sampler som havner innenfor volumet V_n:

$$k_n = \sum_{i=1}^{n} \varphi\left(\frac{\underline{x} - \underline{x}_i}{h_n}\right) \tag{34}$$

Innsatt i ligning[33]:

$$p_n(\underline{x}) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{h_n^d}\varphi\left(\frac{\underline{x} - \underline{x}_i}{h_n}\right) \tag{35}$$

Her er $\varphi(\cdot)$ en vindusfunksjon som oppfyller følgende krav: $\int_{-\infty}^{\infty} \varphi(\underline{x})d\underline{x} = 1.0$, $\varphi(\underline{x}) >= 0.0$. Vindusbredden h_n er en parameter som må bestemmes når klassifisereren designes, for eksempel kan den settes til $h_n = \frac{h_1}{\sqrt{n}}$ med h_1 som parameter. d er dimensjonen til egenskapsvektoren $\underline{x}$, og x_i er egenskapsvektoren tilhørende treningsdata nr. i.

Det går an å se på denne metoden som en slags interpolering av vindusfunksjoner som tilnærming av en sann tetthetsfordeling, hvert av treningsdataene bidrar med sin vindusfunksjon.

Kn-nærmeste-naboer I stedet for å anta et fast volum V_n kan vi heller utvide volumet inntil det inneholder k_n antall treningsdata av samme klasse, med k_n ofte gitt som en funksjon av n (antall treningsdata for klassen).

Funksjonen for k_n bør være slik at når n går mot uendelig så går også k_n mot uendelig, samtidig bør k_n vokse tregt nok til at $\frac{k_n}{n}$ går mot null. $k_n = \sqrt{n}$ vil for eksempel være en slik funksjon.

Volumet V_n kan uttrykkes som volumet til en d-ball (hyperkule) med avstanden fra $\underline{x}$ til det k_n-te datapunktet som radius.

En d-ball kan beregnes rekursivt slik [8]:

$$V_n = V_d = \frac{2\pi r^2}{n} V_{d-2} \tag{36}$$

$$V_1 = 2r \tag{37}$$

$$V_2 = \pi r^2 \tag{38}$$

Merk at n og d er ikke samme variabel og har ingenting med hverandre å gjøre, n merker volumet tilhørende treningsdata nr. n, og d er dimensjonen i forhold til å beregne volumet til en d-dimensjonal kule.

Nå kan ligning[33] brukes for å estimere tetthetsfunksjonen innsatt for k_n og V_n.

En bør passe seg litt her for tilfeller der $V_n = 0$ når treningsdataene ligger oppå hverandre i samme punkt som egenskapsvektoren $\underline{x}$, ligning[33] er da ikke definert (gå mot uendelig).

2.8.1 Estimat av a posteriori sannsynlighet med Kn-nærmeste-naboer

Som nevnt har Kn-nærmeste-naboer den ulempen at estimatet kan gå mot uendelig hvis treningsdataene ligger oppå hverandre sammen med egenskapsvektoren. En måte å unngå dette på er å estimere a posteriori sannsynlighet med utgangspunkt i $p_n(\underline{x}|\omega_i)$.

Dersom k_i er antall datapunkter fra klasse nr. i som havner innenfor volumet V_n, og k er det totale antallet sampler innenfor det samme volumet. Da kan a posteriori sannsynlighet estimeres slik:

$$\begin{aligned} P_n(\omega_i|\underline{x}) &= \frac{p_n(\underline{x}|\omega_i)}{\sum_{j=1}^{c} p_n(\underline{x}|\omega_j)} \\ &= \frac{k_i}{k} \end{aligned} \tag{39}$$

Her trenger vi å finne de k_i nærmeste naboene til egenskapsvektoren $\underline{x}$ av samme klasse, radiusen til den ytterste naboen bestemmer da volumet, og så telles totalt antall sampler innenfor samme volum. A posteriori sannsynlighet er da estimert til andelen av naboer innenfor volumet V_n som tilhører klasse i.

Vi klassifiserer med en slags flertalls stemme-prosess, klassen med flest antall treningsdata innenfor det samme volumet vinner valget.

En fordel her er at vi slipper å beregne et volum V_n, kun radiusen til volumet brukes.

2.9 Ytelsesmål og analyseverktøy

For å evaluere de forskjellige egenskapene og klassifisererene trengs det noen ytelsesmål og metoder for å analysere stabilitet osv. Ytelsesmålene nevnt her er beregnet på et fler-klasse-system med mer enn 2 klasesr.

Her er en gjennomgang av målene brukt i prosjektet.

2.9.1 Klassifisering av kjent datasett

Dersom et testsett har kjent klassetilhørighet kan et enkelt ytelsesmål være å klassifisere settet og så sammenligne klassetilhørighetene fra klassifiseringen med de kjente sanne klasse-tilhørighetene, bilde for bilde og klasse for klasse.

Først trenes en klassifiserer, og klassifisereren brukes så til å klassifisere det kjente testsettet. Resultatene fra klassifiseringen sammenlignes med de kjente klassetilhørighetene. Basert på dette kan både en total ytelse og en per-klasse ytelse estimeres.

Dersom testsettet er stort nok vil dette enkle estimatet være et fullgodt statistisk sikkert ytelsesmål. Er testsettet stort nok bør det deles opp i flere sett som sammen kan validere hverandres resultater.

2.9.2 Kryssvalidering

Til vanlig arbeider en med faste datasett, og som regel minst 2 testsett. En litt grundigere test er å sette sammen testsett av tilfeldig valgt data.

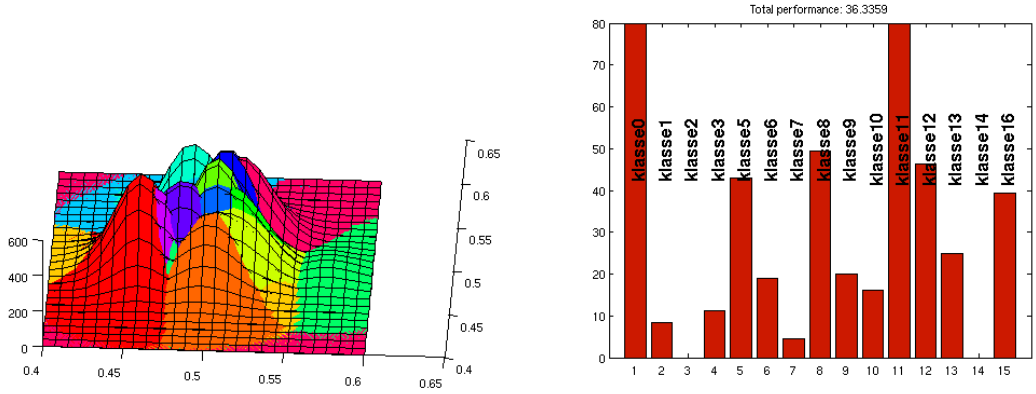
Fra flere kjøringer av slike tester kan vi for eksempel beregne gjennomsnittsytelse og variansen til ytelsene. Snittytelsen bør helst være så nærme 1.0 som mulig, og variansen så liten som mulig.

Boken [1] beskriver forskjellige algoritmer og statistiske prinsipper som kan brukes.

t-fordelingen og t-tester Med n uavhengige ytelsesmålinger kan et konfidensintervall estimeres. [1]

$$p \pm t_{\alpha/2,n-1} \frac{S}{\sqrt{n}} \tag{40}$$

Der p er gjennomsnittsytelse fra n uavhengige tester. $t_{\alpha/2,n-1}$ er t-fordelingen (tabelloppslag brukes).



(a) Plott av parzen-estimerte tetthetsfordelinger.

(b) Ytelse for verifiseringssettet.

Figur 8: Klassifiserer med parzen-estimering av tetthetsfunksjonen for hver klasse. Egenskaper: massesenter i x- og y-retning. Dette er ikke et godt resultat.

3 Egenskaper

Et viktig delmål i dette prosjektet er å finne gode egenskaper som skiller mellom de forskjellige typer skilt. Dette kapittelet går gjennom egenskapene jeg har vurdert, resultatene fra disse eksperimentene kommer senere i rapporten.

Egenskaper kan i prinsippet være hva som helst, men ofte kan det være lurt å lete etter karakteristikker i bildene som er unike for hver klasse, og dermed gir god separabilitet og ytelse. Fig[8] er et eksempel på en dårlig egenskapskombinasjon (massesenter i x- og y-retning), det illustrerer at det ofte kan være ganske utfordrende å komme på gode egenskaper som gir god ytelse for alle klassene.

Noen observasjoner:

- Siden bildene består utelukkende av skilt kan det ligge diskriminerende karakteristikker hvor som helst i bildet, objektet som skal klassifiseres er da hele bildet.
- Når vi ikke vet antallet av klasser bør egenskapene planlegges slik at eventuelle nye klasser også får god ytelse.
- Det er antatt at bildene innenfor samme klasse kan ha forskjellig størrelse, dermed bør ikke koordinater brukes direkte som koordinater. De bør heller normaliseres til en skala $[0, 1]$.
- Lysforholdene vil forstyrre noen av egenskapene.

Bildene kan ha ulik dimensjon og størrelsesforhold, også innenfor samme klasse (dette vil være forskjellig avhengig av hvor nært bildet ble tatt). Det er viktig å ta hensyn til dette når egenskapene planlegges og implementeres.

For å motvirke forskjeller i lysforhold kan bildet høypassfiltreres da mesteparten av lysinformasjonen legger seg i de lavere frekvensene. Jeg skiller her mellom to bildetyper; Gradient, og normal for henholdsvis høypassfiltrert bilde og ufiltrert.

3.1 Statistiske egenskaper

Disse egenskapene er basert på statistiske mål.

Gjennomsnittspikselverdi

$$x = \frac{1}{M \cdot N} \sum_{m=1}^{M} \sum_{n=1}^{N} I(m, n) \tag{41}$$

Der $I(m, n)$ er et piksel fra bildet I med dimensjonene $[M, N]$, og x er egenskapen.

Varians

$$x = \frac{1}{M \cdot N} \sum_{m=1}^{M} \sum_{n=1}^{N} [I(m,n) - \mu]^2 \tag{42}$$

Der μ er middelverdien til bildet I.

Variansen kan ta ganske store verdier sammenlignet med andre egenskaper, derfor bør variansen skaleres.

Entropi Entropi er et statistisk mål på tilfeldighet.

Førsteordens entropi er definert slik:

$$x = -\sum_{i=1}^{R} H(i) \log_2[H(i)] \tag{43}$$

Der H er histogrammet til et bilde, og R er antall mulige pikselverdier (og størrelsen til H).

Entropien til et bilde kan gi verdier i området $[0, \log_2(N)]$, der N er antall mulige pikselverdier (typisk 256).

Diagonal: gjennomsnitt I stedet for å beregne gjennomsnittet av hele bildet kan det være nok å kun se på diagonalen. Dette reduserer mengden beregninger en del.

MATLAB har en innebygget funksjon for å finne diagonalen til et bilde. Dette er funksjonen "improfile", den trenger minst 3 argumenter:

1. Bildedata
2. xi - X-koordinatene til endepunktene til diagonalen.
3. yi - Y-koordinatene til endepunktene til diagonalen.

Diagonal: varians Som egenskapen over, men i stedet for gjennomsnittet beregnes heller variansen til diagonalen.

Denne trenger å skaleres.

Tekstur-egenskaper Følgende egenskaper kan beregnes av "co-occurencematrisen:

- Kontrast
- Korrelasjon
- Energi
- Homogenitet
- Maksimum
- Entropi

Se kap[2.4] for mer detaljer.

3.2 Massesenter

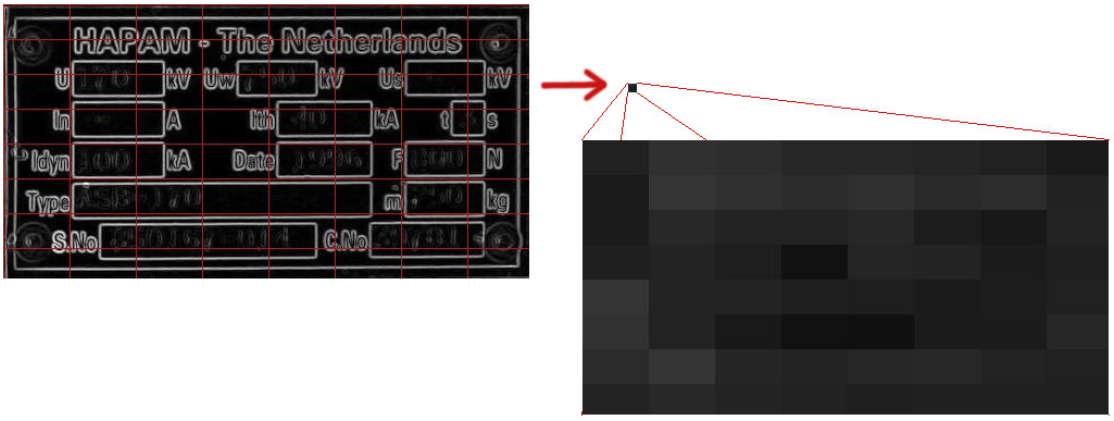
Dersom vi ser på pikslene i et bilde som partikler kan et massesenter beregnes (se kap[2.2]).

$$x_m = \frac{\sum_{m=1}^{M} \sum_{n=1}^{N} I(m,n) \cdot m}{\sum_{m=1}^{M} \sum_{n=1}^{N} I(m,n)} \tag{44}$$

$$x_n = \frac{\sum_{m=1}^{M} \sum_{n=1}^{N} I(m,n) \cdot n}{\sum_{m=1}^{M} \sum_{n=1}^{N} I(m,n)} \tag{45}$$

Her får vi to individuelle egenskaper.

Det er koordinater som beregnes her og disse bør normaliseres.



Figur 9: Inndeling av et gradientbilde. Lokal egenskap er her gjennomsnittspikselverdi. Dette gir en egenskapsvektor med en egenskap per blokk.

Massesentrum av frekvensspektrum I stedet for å beregne et massesentrum av selve bildet kan kanskje frekvensspekteret gi resultater. I tilfeller der bildene er litt translert vil fortsatt frekvensspekteret være nokså uendret.

Som med massesentrumet av originalbildene bør koordinatene skaleres. Teori om fourier-transformen kap[2.1].

3.3 Lokale egenskaper

Når en person prøver å skille en gjenstand fra en annen utføres dette ved at han studerer selve gjenstanden. Dersom han allerede vet hva han skal se etter er det nok å se på akkurat dette, men dersom det for han er en ukjent gjenstand vil han gjennomsøke hele gjenstanden helt til han finner en diskriminerende effekt ved gjenstanden.

Med inspirasjon av dette deles bildet opp i mindre blokker, for så å beregne en egenskap for hver blokk, dette vil forhåpentligvis gi god ytelse for et stort antall klasser samt nye klasser som senere dukker opp.

Fig[9] viser blokkinndeling av et bilde, og gjennomsnittsberegning per blokk. Hver blokk representerer en egenskap. Denne egenskapen kan være en av enkeltegenskapene beskrevet i forrige delkapittel, eller en kombinasjon av disse. Den totale egenskapsvektoren blir da det sammensatte settet av de lokale egenskapsvektorene fra blokkene.

4 Implementering

Denne delen av rapporten gir en gjennomgang av hvordan jeg har implementert eksperimentene.

Implementasjonen er laget med tanke på eksperimentering og prøving av mange forskjellige egenskaper og klassifiserere og derfor er koden skrevet med tanke på generalitet på bekostning av effektivitet. For flere blokkdiagrammer og en beskrivelse av kodefilene se app[C].

Jeg har valgt å benytte meg av MATLAB som programmeringsverktøy/språk. Dette fordi MATAB er et godt verktøy til utvikling og eksperimentering av matematiske prinsipper og algoritmer. MATLAB har mange nyttige funksjoner og algoritmer innebygget, dette sparer meg for mye tid og jeg kan fokusere mer på selve klassifiseringsproblemet.

Litt om matriser og vektorisering i MATLAB. Det grunnleggende objektet i MATLAB er matrisen, mange matematiske operasjoner kan vektoriseres ved å danne matriser og vektorer og benytte (raske) matriseoperasjoner i stedet for trege for-løkker. Eksempel:

$$l = \sqrt{\sum_i (\mathsf{p}_{1,i} - \mathsf{p}_{2,i})^2}$$

Dette er den euklidiske avstanden mellom to punkter $\underline{\mathsf{p}}_1$ og $\underline{\mathsf{p}}_2$. Denne operasjonen kan vektoriseres slik:

$$\underline{\mathsf{v}} = \underline{\mathsf{p}}_1 - \underline{\mathsf{p}}_2$$
$$l = \sqrt{\underline{\mathsf{v}}^T \underline{\mathsf{v}}}$$

Dersom vi trenger å lage lister av ikke-numeriske objekter kan såkalte **celle-array** benyttes. Dette er generelle lister som kan holde på alle typer objekter. Ingen matematiske operasjoner virker direkte på celle-array, her er vi avhengige av for-løkker for å iterere over elementene.

4.1 Rammeverk

Kodebasen er designet med tanke på generalitet. Dette betyr mindre forprosessering og noen ganger mer dobbelt arbeid", til gjengjeld kan flere forskjellige typer klassifiserere og egenskaper implementeres og enkelt plugges direkte inn i resten av klassifiseringssystemet.

4.1.1 Datakilde

Det er som regel lurt å ha et abstraksjonslag mellom systemet og de fysiske dataene (her bildene). Dette laget sørger for å finne data og sette dem sammen til forskjellige datasett.

I min implementasjon er datasettene som dette laget gir ut et celle-array av filstier til bildene tilhørende samme klasse og datasett. Argumentene inn er valg av klasser vha. mappenavn, og valg av type datasett (også mappenavn) (fig[10]). Bildene er antatt å være lagret som i fig[11]. Selv om bildene ligger sortert etter sann klassetilhørighet er denne informasjonen ikke brukt i klassifiseringsprosessen. Informasjonen er derimot nyttig i forhold til analyser og ytelsesmåling (se kap[2.9]).

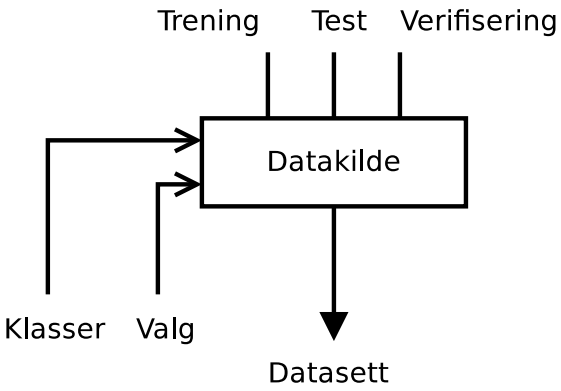
4.1.2 Klassifiseringssystem

Rammeverket er bygget opp rundt et sentralt objekt som representerer en klasse.

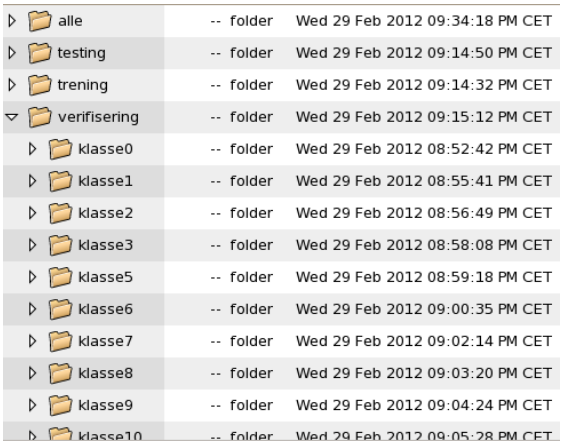
Representering av en klasse. Objektet inneholder alle relevante data og funksjoner som hører naturlig til en klasse. Fig[12] viser klassen med datamedelemmer og funksjonsmedlemmer.

Oversikt over klasse-objektet:

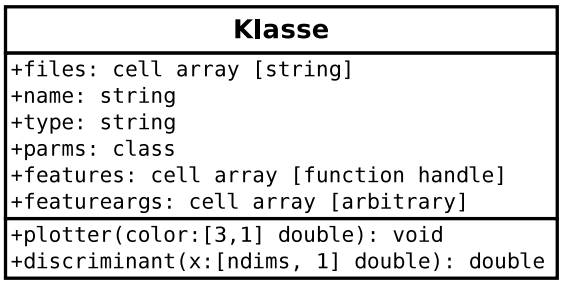
- **+files:** liste av bilder (tatt direkte fra datakilden) som til sammen danner treningssettet for klassen.



Figur 10: Datakilden - abstraksjonslaget mellom systemet og fysiske data. 'Klasser' er en liste over hvilke klasser som skal lastes, og 'Valg' velger mellom typene av datasett; Trening", Test", og "Verifisering".

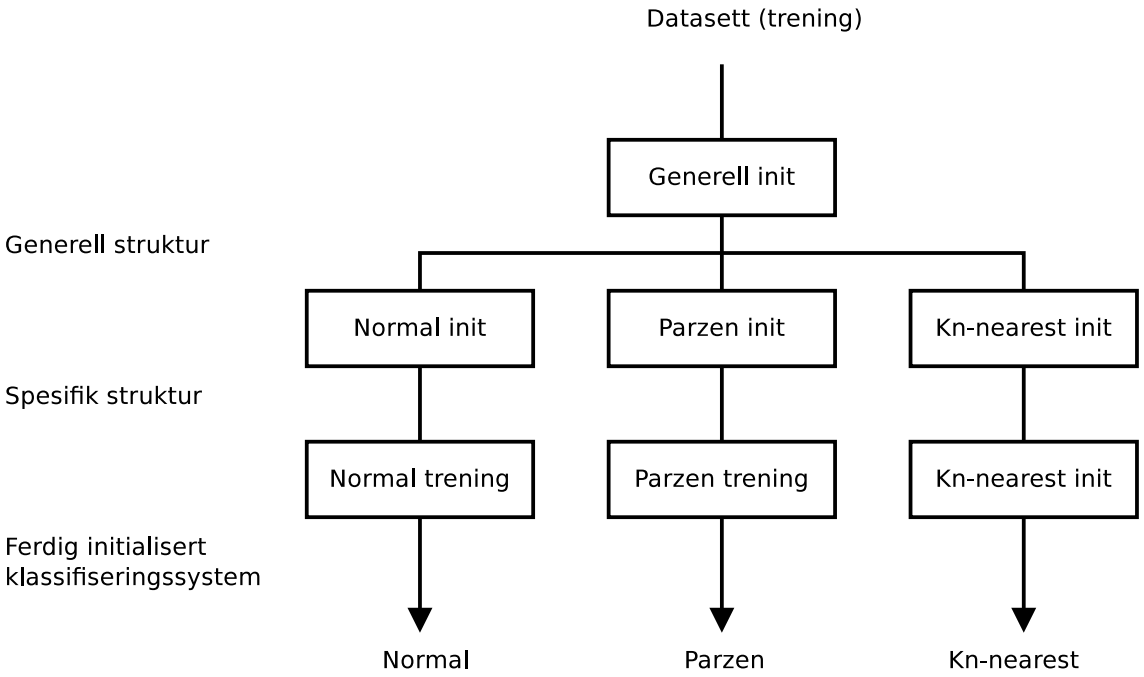


Figur 11: Mappestrukturen brukt for bildene.



Figur 12: Objekt som representerer en klasse.

Figur 13: Klassifiseringssystem satt opp som et celle-array av klasser.



Figur 14: Initialiseringsprosessen.

- **+name:** navn til klassen.
- **+type:** type klassifiserer, eks: 'bayes'.
- **+parms:** ekstra klassifiserer parametre, varierer etter type klassifiserer.
- **+features:** egenskapsvektor (lik for alle klasser i systemet).
- **+featureargs:** ekstra argumenter gitt til egenskapsfunksjonene.
- *+plotter:* funksjon som lager et plot av klassens diskriminantfunksjon, kan plotte 1-2 egenskaper.
- *+discriminant:* diskriminantfunksjon tilhørende klassen.

Oppsett av klassifisereren Siden alle klassene i en klassifiserer er avhengig av hverandre (for eksempel Pw i en Bayes klassifiserer) er det naturlig å samle alle klasse-objektene i et klassifiserer-objekt. Her har jeg i min implementering ganske enkelt samlet alle klasse-objektene vha. et celle-array, denne listen av objekter representerer da et fullstendig klassifiseringssystem. Fig[13] illustrerer klassifisereren.

En konsekvens av å sette opp klassifisereren som en enkel liste på denne måten er at hver klasse-objekt må ha en lik kopi av alle klassifiserer-spesifike elementer (som for eksempel listen av funksjoner for beregning av egenskaper).

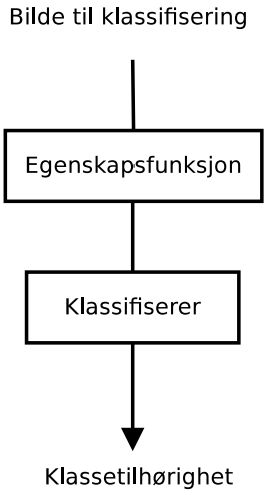
4.2 Initialisering og klargjøring av klassifisereren

Før klassifisereren kan brukes må et treningssett defineres for hver klasse, i tillegg må typen klassifiserer velges. Fig[14] viser initialiseringsprosessen.

Først settes den generelle klasse-strukturen opp, deretter settes datamedlemmet *parms* fra klasse-objektet opp med type-spesifike parametre som gir *spesifik struktur* i fig[14]. Til slutt trenes klassifisereren med treningsdataene listet i datamedlemmet *files*.

4.3 Klassifiseringsprosessen

Forutsatt at klassifisereren er initialisert og klar til bruk gjelder prosessen beskrevet i fig[15]. Først beregnes en egenskap fra et bilde, deretter brukes egenskapen til å bestemme bildets klassetilhørighet i klassifisererens diskriminantfunksjon.



Figur 15: Klassifiseringsprosessen,

4.3.1 Egenskapene

Egenskapsfunksjonene tar inn ett argument: en matrise med bildedata. Disse funksjonene utfører all nødvendig bildebehandlig, for eksempel høypassfiltrering for å redusere forskjeller i lysforhold.

Funksjonsprototypen for beregning av egenskaper ser slik ut:

```
function [egenskaper, N] = egenskap(bildedata, args)
```

Funksjonen gir ut en egenskapsvektor, samt antall egenskaper beregnet.
`args` inneholder ekstra argumenter som brukes ved beregning av noen egenskaper.

4.3.2 Klassifisereren

Klassifisereren implementerer følgende desisjonsregel (ligning[29]):

$$\text{Velg } \omega_i \text{ hvis } g_i(\underline{x}) > g_j(\underline{x}) \quad \text{for alle } j \neq i$$

Klassifisereren antar at diskriminantfunksjonene har følgende prototype:

```
function [gi] = diskriminant(klasse, x, klassifiserer)
```

Her er `gi` en skalar som benyttes for å utføre klassifiseringen (via desisjonsregelen).

Grunnen til at både klasse og klassifiserer gis inn som argument er at noen diskriminantfunksjoner trenger informasjon fra andre klasser (for eksempel kn-nærmeste-nabo som teller opp sampler fra alle klasser).

`x` er en enkelt (multidimensjonal) egenskapsvektor.

4.4 Implementering av klassifisereren

Her gis en detaljert beskrivelse av implementeringen av klassifisereren og diskriminantfunksjonene. I tillegg en av egenskapene.

4.4.1 Diskriminantfunksjon: normal

Her benyttes en Bayes diskriminantfunksjon, formelen for denne er gitt i teori-kapittelet i ligning[30].

Ligning[30] innsatt i ligning[27] gir:

$$g_j(\underset{\sim}{x}) = \frac{P(\omega_j)}{p(\underset{\sim}{x})(2\pi)^{\frac{d}{2}}|\underset{\sim}{\Sigma_j}|^{\frac{1}{2}}}\mathsf{e}^{-\frac{1}{2}(\underset{\sim}{x}-\underset{\sim}{\mu_j})^T\underset{\sim}{\Sigma_j}^{-1}(\underset{\sim}{x}-\underset{\sim}{\mu_j})}$$

Det er lov å utføre lineære operasjoner på uttrykket. For eksempel den naturlige logaritmen:

$$\ln[g_j(\underset{\sim}{x})] = \ln[P(\omega_j)] - \ln[p(\underset{\sim}{x})] - \frac{d}{2}\ln[2\pi] - \frac{1}{2}\ln\left[|\underset{\sim}{\Sigma_j}|\right] - \frac{1}{2}(\underset{\sim}{x}-\underset{\sim}{\mu_j})^T\underset{\sim}{\Sigma_j}^{-1}(\underset{\sim}{x}-\underset{\sim}{\mu_j})$$

Forenkling av diskriminantfunksjonen Det går an å optimalisere denne en del. Alle ledd som er konstante for alle diskriminantfunksjonene g_j kan fjernes, i dette tilfellet kan kun $\frac{d}{2}\ln[2\pi]$ og $\ln[p(\underset{\sim}{x})]$ droppes.

$$\begin{aligned}\ln[g_j(\underset{\sim}{x})] &= \ln[P(\omega_j)] - \frac{1}{2}\ln\left[|\underset{\sim}{\Sigma_j}|\right] - \frac{1}{2}(\underset{\sim}{x}-\underset{\sim}{\mu_j})^T\underset{\sim}{\Sigma_j}^{-1}(\underset{\sim}{x}-\underset{\sim}{\mu_j}) \\ &= \underset{\sim}{x}^T\underset{\sim}{W_j}\underset{\sim}{x} + \underset{\sim}{w_j}^T\underset{\sim}{x} + w_{j0} \qquad (46)\end{aligned}$$

$$\underset{\sim}{W_j} = -\frac{1}{2}\underset{\sim}{\Sigma_j}^{-1} \qquad (47)$$

$$\underset{\sim}{w_j} = \underset{\sim}{\Sigma_j}^{-1}\underset{\sim}{\mu_j} \qquad (48)$$

$$w_{j0} = -\frac{1}{2}\underset{\sim}{\mu_j}^T\underset{\sim}{\Sigma_j}^{-1}\underset{\sim}{\mu_j} - \frac{1}{2}\ln\left[|\underset{\sim}{\Sigma_j}|\right] + \ln[P(\omega_j)] \qquad (49)$$

Her unngår vi også eksponenter av $\underset{\sim}{x}$, dette gir bedre numerisk stabilitet for ekstreme verdier fra $-\frac{1}{2}(\underset{\sim}{x}-\underset{\sim}{\mu_j})^T\underset{\sim}{\Sigma_j}^{-1}(\underset{\sim}{x}-\underset{\sim}{\mu_j})$, eksponenten til e.

Siden jeg i dette prosjektet har antatt en stor mengde klasser gir dette en stor $\underset{\sim}{\sigma}$, for å unngå singularitet legges det til en konstant verdi til diagonalen til $\underset{\sim}{\Sigma}$. Dette har den effekten å øke bredden til tetthetsfunksjonene.

Ligning[46] - ligning[49] er implementert i `d_normal.m`.

Trening En normalfordeling har 2 parametre som må estimeres. Dette gjøres via estimatorer av typen MLE. Ligningene [31] og [32] brukes for å estimere parametrene (gjengitt under).

$$\underset{\sim}{\mu} = \frac{1}{n}\sum_{i=1}^{n}\underset{\sim}{x}_i$$

$$\underset{\sim}{\Sigma} = \frac{1}{n-1}\sum_{i=1}^{n}(\underset{\sim}{x}-\underset{\sim}{\mu})^T(\underset{\sim}{x}-\underset{\sim}{\mu})$$

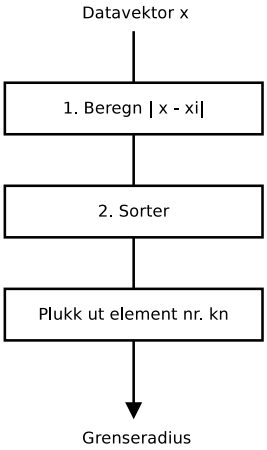
Parametrene $\underset{\sim}{\mu}$ og $\underset{\sim}{\Sigma}$ lagres for hver klasse, og klassen har da blitt trent er klar til å brukes.

Filen `k_normal_train.m` implementerer treningen av klassifisereren.

4.4.2 Diskriminantfunksjon: parzen

For å implementere parzen-estimatoren benyttes to funksjoner. Den ene implementerer ligning[35] (gjengitt under) og den andre implementerer vindusfunksjonen $\varphi(\cdot)$.

Vindusfunksjonen er i dette prosjektet satt til å være en multivariabel standard normalfordeling med standardavvik lik 1 (og dermed kan $\underset{\sim}{\Sigma}$ kuttes ut).



Figur 16: Beregning av grenseradius.

$$p_n(\underline{x}) = \frac{1}{nh_n^d} \sum_{i=1}^{n} \varphi\left(\frac{\underline{x} - \underline{x}_i}{h_n}\right)$$

Jeg bruker en standard normalfordeling som vindusfunksjon.

$$\varphi(\underline{x}) = \frac{1}{(2\pi)^{\frac{d}{2}}} e^{-\frac{1}{2}\underline{x}^T\underline{x}}$$

Der d er dimensjonen til $\underline{x}$.
Begge funksjonene beskrevet her er implementert i `d_parzen.m`.

Trening En parzen-estimator plasserer et vindu ved hvert av treningsdataene. Plasseringen av et treningsdata er definert av egenskapsvektoren.

Hver klasse lagrer en liste av slike egenskapsvektorer i treningsprosessen. Samtidig beregnes også parameteren h_n. Jeg har valgt å definere denne parameteren slik:

$$h_n = \frac{h_1}{\sqrt{n}}$$

Der n er antall treningsdata.
Filen `k_parzen_train.m` implementerer treningen av klassifisereren.

4.4.3 Diskriminantfunksjon: kn-nærmeste-nabo

Kn-nærmeste-nabo-estimatoren kommer i to former: en estimerer en tetthetsfunksjon (klassespesifik sannsynlighet, se kap[2.8]), den andre estimerer a posteriori sannsynlighet (se kap[2.8.1]).

Jeg har valgt å implementere formen som estimerer a posteriori sannsynlighet fordi denne unngår deling med 0 (når $V_n = 0$), den kan implementeres lettere og raskere (slipper å beregne et volum V_n siden det her kun er nødvendig å beregne radiuser).

Klassifiseringsprosessen består av to deler, først finnes det minste volumet som inneholder k_n sampler tilhørende samme klasse, deretter telles det opp hvor mange sampler som finnes totalt innenfor samme volum. Disse to tallene bestemmer da sannsynligheten for klassen.

Problemet med å finne de k_n nærmeste treningsvektorene kan enkelt implementeres ved å beregne den euklidiske avstanden mellom datavektoren som klassifiseres og alle treningsvektorene. Denne listen av avstander sorteres og element nr. k_n vil da holde radiusen til det gjeldende volumet. Fig[16] illustrerer beregningen av grenseradiusen.

Når grenseradiusen er funnet telles det totale antallet sampler som faller innenfor dette volumet opp. Denne prosessen er nesten lik som prosessen for å finne grenseradiusen. For alle

klasser beregnes avstander på samme måte, listen sorteres, og indeksen til det siste elementet som faller innenfor grenseradiusen legges til totalen.

Sannsynligheten for den gitte klassen estimeres da som gitt i ligning[39], gjengitt her:

$$p(\omega_i|\underline{x}) = \frac{k_i}{k} \tag{50}$$

k_i er kjent fra før, k telles opp.

Ligning[50] er implementert i `d_knearest.m`.

Trening Å trene en kn-nearest-klassifiserer er enkelt. Som med parzen-vindus-metoden lagres egenskapsvektorene beregnet av treningsbildene. I tillegg beregnes parameteren $k_n = \sqrt{n}$, der n er antall treningsbilder. Parameteren rundes til nærmeste heltall.

Filen `k_knearest_train.m` implementerer treningen av klassifisereren.

4.4.4 Egenskap: lokale gjennomsnitt

Her er implementeringen av den mest lovende egenskapen; *lokale gjennomsnitt*. Denne egenskaper består av to komponenter; blokkinndelingsfunksjonen, og egenskapen *gjennomsnitt*. Se kap[3] for flere detaljer.

Lokale egenskaper For å beregne lokale egenskaper deles bildet inn i blokker, en egenskap beregnes da for hver blokk. Alle egenskapene fra blokkene kombineres så til en egenskapsvektor.

MATLABs "image processing toolbox"har innebygget en funksjon kalt `blockproc` (se [2]), denne funksjonen deler bildet inn i [M,N] blokker og kaller en gitt funksjon på hver av blokkene. Den har følgende prototype:

```
function [x] = blockproc(A, [M,N], fun)
```

Funksjonen funskal ha følgende form:

```
function [a] = fun(blockstruct)
```

Argumentet `blockstruct` inneholder en blokk fra bildet A. Denne funksjonen kalles en gang for hver av de [M,N] blokkene.

Gjennomsnitt Den innebygde funksjonen `mean2`, benyttes for å beregne gjennomsnittet til et bilde.

Blokkfunksjonen er da implementert slik:

```
function [a] = fun(blockstruct)
    a = mean2(blockstruct.data);
end
```

Denne gir ut en egenskap per blokk.

4.5 Kryssvalidering

Implementeringen min av kryssvalidering bygger på datakilden fra [4.1.1]. Kryssvalideringen implementeres ved å velge ut tilfeldige elementer fra datasettet som datakilden gir ut.

1. Be om datasett fra datakilden.
2. Generer N antall tilfeldige tall mellom 1 og totalt antall data i datasettet.
3. Bruk de tilfeldige tallene som indekser til å plukke ut data.

Dataene blir da tilfeldig plukket ut uten tilbakelegging.

Funksjonen `k_crossvalid_init.m` erstatter `k_init.m` for å gi tilfeldige treningssett. Funksjonen `u_rand_datasett.m` implementerer stegene beskrevet over og erstatter `u_datasett.m` som datakilde.

4.6 Eksempel

Her er et eksempel på en mesterfil som måler ytelsen til en egenskap. Eksempelet er basert på filen `t_total_normal_mean.m`.

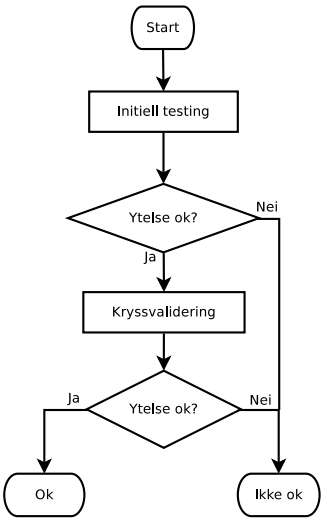
```
clear all;

% VARIABLER OG KONFIGURERING.
klasser = {'klasse0','klasse1','klasse2','klasse3','klasse5', ...
          'klasse6','klasse7', 'klasse8','klasse9','klasse10', ...
          'klasse11','klasse12','klasse13','klasse14','klasse16'};
egenskaper = {@e_snitt};
egenskapsargs = {[]};
t_klasse_sett = {'testing','verifisering'};

% KLARGJØRING OG INITIALISERING AV KLASSIFISERINGSSYSTEMET
a = k_init(klasser, egenskaper, egenskapsargs, klasser);
b = k_normal_init(a);
b = k_normal_train(b,b);

% KLASSIFISER OG ANALYSER RESULTATENE, RESULTATENE FRA KLASSIFISERINGEN
% LEGGES INN I STRUKTURENE dt1 og dt2.
Nklasser = numel(klasser);
dt1 = a_classify_datastructure(b, a_create_datastructure(klasser, ...
    repcell(Nklasser, 1, t_klasse_sett{1}) ));
dt2 = a_classify_datastructure(b, a_create_datastructure(klasser, ...
    repcell(Nklasser, 1, t_klasse_sett{2}) ));

% PRESENTER TOTAL YTELSE, DET GÅR OGSÅ AN Å FINNE
% YTELSE PER KLASSE MED a_analyse_per_class(dt1).
disp(['Performance for "mean" for test dataset 1 ...
     (testing) using normal density: ', num2str(a_analyse_total(dt1)), '%.'])
disp(['Performance for "mean" for test dataset 2 ...
     (verification) using normal density: ', num2str(a_analyse_total(dt2)), '%.'])
```

Først defineres variabler: Klassene som skal testes, egenskapsvektoren gitt som en liste av funksjoner. Egenskapen gjennomsnitttrenger ingen ekstra argumenter, derfor gis dette kun en tom matrise.

Så klargjøres et klassifiseringssystem via funksjonene `k_init`, `k_normal_init`, og `k_normal_train`. Treningsdataene hentes automatisk og kobles til klassene.

Selve klassifiseringen utføres av en analysefunksjon. Denne holder orden på sann klassetilhørigeter- og klassifisert klassetilhørighet- for hvert bilde fra hver klasse. Sann klassetilhørighet bestemmes av navnet på mappen bildet fysisk ligger i (se fig[11]).

Funksjonen `a_analyse_total` beregner andelen av korrekt klassifisering ved å sammenligne sann klassetilhørighet og klassifisert klassetilhørighet. Se app[C] for detaljer om de forskjellige kodefilene og funksjonene.



Figur 17: Arbeidsflyt, egenskapsutvikling.

5 Eksperiment

Dette kapittelet går gjennom eksperimentene som er utført i forbindelse med å løse hovedmålet til prosjektet. Egenskapene listet i kap[3] er blitt testet med en normalfordelt Bayes klassifiserer.

Da treningssettet er lite har jeg lagt mest vekt på den normalfordelte Bayes klassifisereren siden denne krever et lavere antall treningsdata enn de ikke-parametriske klassifisererene.

Først har jeg testet og indeksert alle de forskjellige egenskapene listet i kap[3]. Så testes alle mulige kombinasjoner av egenskapene (to og to).

Lokale egenskaper (fra kap[3.3]) har en parameter, denne testes ved å gradvis øke parameteren mens det testes for total ytelse.

5.1 Oppsett

Eksperimentene er fokusert på å finne kombinasjoner av egenskaper for å få optimal ytelse. Eksperimentene klassifiserer to kjente datasett og måler ytelsen ved å sammenligne resultatene fra klassifisereren med de kjente klassetilhørighetene innen datasettene. Alle eksperimentene er testet på to uavhengige datasett. Datasettene er oppsummert i appendix[B].

I utviklingen av forskjellige egenskaper er det benyttet en arbeidsflyt som vist i fig[17].

Også forskjellige typer klassifiserere prøves ut med denne metoden.

Ikke alle eksperimentene presentert her representerer fornuftige løsninger, De fleste eksperimentene er tatt med for å demonstrere hva som er blitt gjort, og viser hva som ikke virker. Se kap[6] for reflekteringer og konklusjoner om resultatene.

Initiell testing Det enkleste og kanskje naive å gjøre er å trene klassifisereren med de faste treningsbildene, klassifisere testsettet og verifiseringssettet, og ut fra resultatet fra klassifiseringen beregne prosentvis korrekt klassifisering. Dette gir et enkelt ytelsesmål, men det er ikke mulig å si så mye om stabiliteten til klassifisereren. Se kap[2.9.1].

Kryssvalidering Etter det er funnet egenskaper som gir gode resultater i den initielle testingen går jeg videre med kryssvalidering. Se kap[2.9.2].

5.2 Resultater

Her følger resultater fra forskjellige eksperimenter i den initielle testingen.

Parametrisk: normal, diagonalelementene til $\underset{\sim}{\Sigma}$ blitt lagt til en konstant verdi for å unngå singularitet (0.1 dersom ikke annet er nevnt).

	Egenskap	Bilde	Klassifiserer	Ytelse (%) testing	verifisering
1	Gjennomsnitt	Gradient	Parametrisk	6.30	6.11
2	Standardavvik	Gradient	Parametrisk	10.24	12.82
3	Middelverdi av diagonal	Gradient	Parametrisk	14.96	15.57
4	Standardavvik av diagonal	Gradient	Parametrisk	10.63	11.15
5	Entropi	Gråskala	Parametrisk	11.42	11.30
6	Massesenter	Gråskala	Parametrisk	23.62	26.56
7	Massesenter av frekvensspektrum	FFT av gråskala	Parametrisk	21.65	21.37
8	Tekstur: kontrast	Gråskala	Parametrisk	14.57	14.66
9	Tekstur: korrelasjon	Gråskala	Parametrisk	21.65	14.66
10	Tekstur: energi	Gråskala	Parametrisk	12.20	12.52
11	Tekstur: homogenitet	Gråskala	Parametrisk	6.69	9.92
12	Tekstur: maksimum	Gråskala	Parametrisk	13.78	13.28
13	Tekstur: entropi	Gråskala	Parametrisk	7.48	11.30

Tabell 1: Resultater fra enkle eksperiment

	1	2	3	4	5	6	7	8	9	10	11	12	13	
1	-	11.91	15.57	8.40	14.05	27.79	21.22	12.67	17.10	13.59	12.21	11.45	12.21	1
2	11.42	-	13.44	13.59	20.92	16.18	23.05	14.50	12.52	13.28	20.92	14.81	18.78	2
3	14.96	12.99	-	10.08	14.35	35.27	26.41	12.67	19.85	19.08	14.96	17.25	15.11	3
4	7.09	12.99	7.87	-	19.69	15.73	19.69	14.05	11.60	9.77	21.53	10.84	17.86	4
5	12.60	20.87	12.60	18.90	-	15.42	16.64	21.53	14.05	14.05	19.08	14.35	24.27	5
6	24.80	13.39	33.46	12.60	12.99	-	26.87	4.86	29.16	30.84	19.69	31.60	19.39	6
7	22.05	19.09	26.77	18.50	16.14	27.17	-	4.89	22.90	23.05	25.34	23.21	24.27	7
8	14.17	15.35	14.17	15.35	19.69	5.91	5.91	-	12.67	12.67	13.44	12.67	13.59	8
9	24.80	11.02	19.69	9.06	12.60	29.53	23.62	14.17	-	20.31	13.59	17.25	13.89	9
10	14.17	13.78	18.90	7.87	12.60	29.13	24.80	14.17	24.02	-	12.52	12.52	13.89	10
11	10.24	16.54	12.60	20.87	20.47	14.96	24.80	15.35	10.24	8.66	-	12.52	22.44	11
12	12.20	14.17	20.08	11.02	12.60	31.10	26.38	14.17	19.29	13.39	8.66	-	22.44	12
13	7.48	15.35	10.24	15.35	22.44	13.78	22.05	15.35	9.84	9.84	22.83	22.83	-	13
	1	2	3	4	5	6	7	8	9	10	11	12	13	

Tabell 2: Resultater fra alle mulige kombinasjoner av 2 enkle egenskaper fra tabell[1]. Tallene under diagonalen: resultater fra testsettet. Tallene over diagonalen: resultater fra verifiseringssettet.

Ikke-parametrisk: parzen, denne er har en fri variabel h_1. Se kap[5.2.2] for eksperimenter med parzen-vinduer.

5.2.1 Enkle egenskaper

Egenskapene listet i kap[3] er testet enkeltvis og i kombinasjon to og to.

Resultatene for single egenskaper er oppsummert i tabell[1]. Resultatene fra alle kombinasjoner av 2 egenskaper er oppsummert i tabell[2]. Resultatene under diagonalen er fra testsettet, resultatene over diagonalen er fra verifiseringssettet.

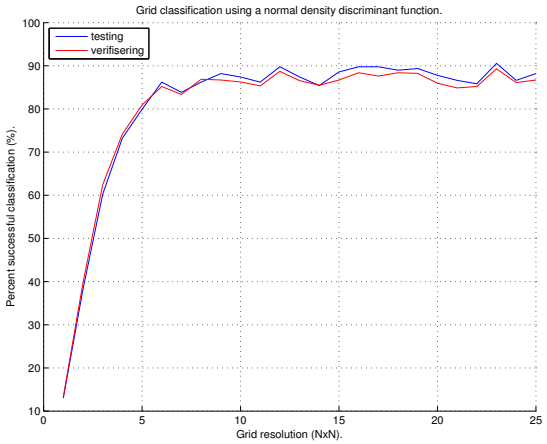
Lokale egenskaper Egenskapen *lokale egenskaper* har en parameter som bestemmer graden av blokkinndeling av et bilde. For å teste ytelsen utføres et eksperiment for hver grad av blokkinndeling.

Resultatene fra eksperimentene er oppsummert i fig[18]. Plottene viser hvordan ytelsen endres med økende grad av blokkinndeling av bildene.

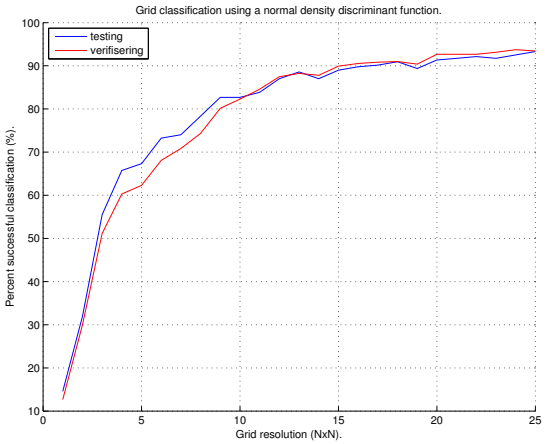
Som en kan se av plottene er ytelsen veldig god.

Plottene i fig[18] viser at ytelsen øker drastisk med økende grad av blokkinndeling. Vi kan konkludere med at egenskapsvektoren lokalt gjennomsnittkombinert med den enkle normalklassifisereren gir et særdeles godt resultat, og at egenskapen dermed faktisk er normalfordelt.

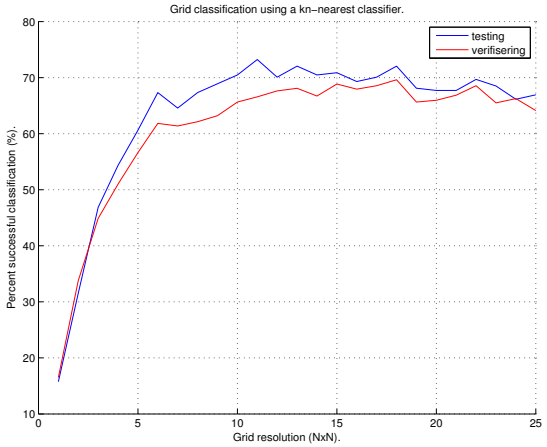
Kn-nærmeste-nabo-klassifisereren trenger flere treningsdata for å få god ytelse.



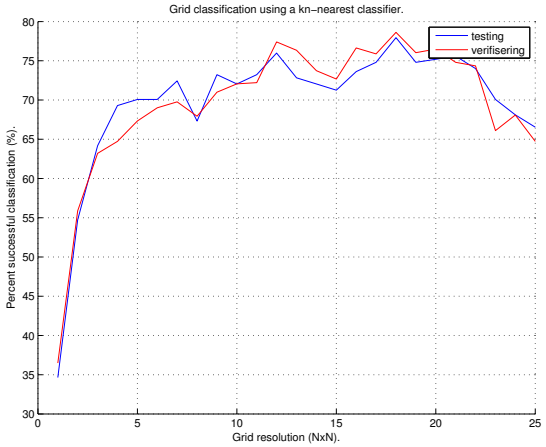
(a) 14, Klassifiserer: normal, Egenskap: Lokalt gjennomsnitt (med kantdeteksjon fra Sobel).



(b) 15, Klassifiserer: normal, Egenskap: Lokalt gjennomsnitt (med kantdeteksjon fra Colorgrad).

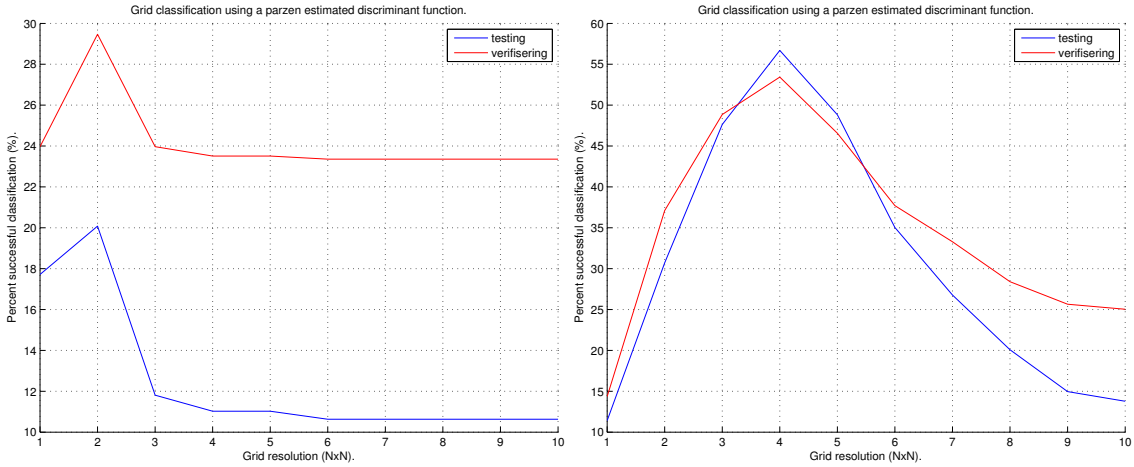


(c) 16, Klassifiserer: kn-nærmeste-nabo, Egenskap: Lokalt gjennomsnitt (med kantdeteksjon fra Sobel).



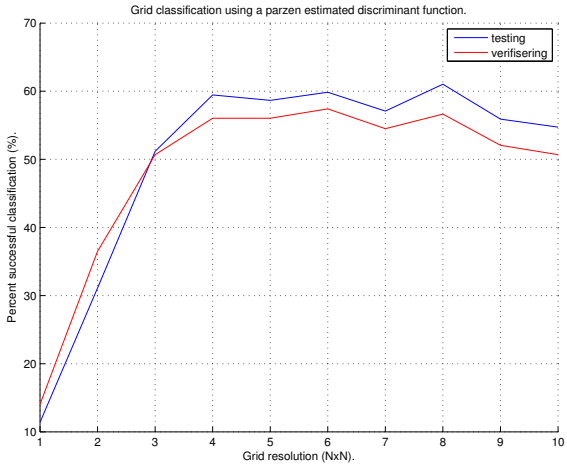
(d) 17, Klassifiserer: kn-nærmeste-nabo, Egenskap: Lokalt massesenter (originalbilde).

Figur 18: Resultater eksperimenter med lokale egenskaper. Plottene viser hvordan ytelsen endres med økende grad av blokkinndeling.

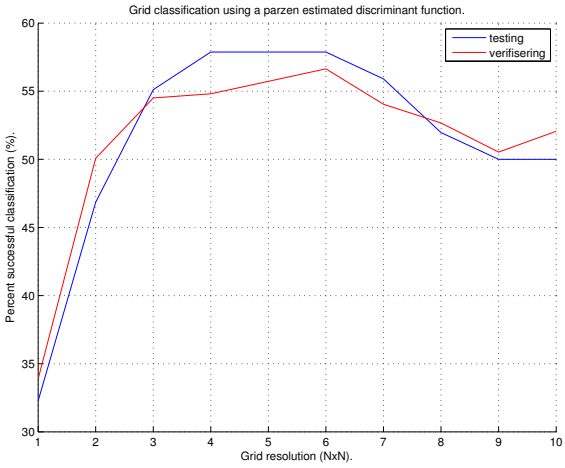


(a) Parzen-vindu, $h_1 = 0.1$.

(b) Parzen-vindu, $h_1 = 1.0$.



(c) Parzen-vindu, $h_1 = 3.0$.



(d) Parzen-vindu, $h_1 = 5.0$.

Figur 19: Egenskapen lokalt gjennomsnittplottet mot økende blokkinndeling. Plottene viser hvordan parameteren h_1 påvirker ytelsesforløpet.

5.2.2 Parzen-vinduer - parameteren h_1

Det er ikke så lett å finne gode parametre og vindusfunksjoner til en Parzen-vindu-estimator. Her er noen eksperimenter med parameteren h_1.

Fig[19] viser egenskapen lokalt gjennomsnitt"med forskjellig grad av blokkinndeling. Plottene viser hvordan ytelsesforløpet påvirkes av bredden til vindusfunksjonen $\varphi(\cdot)$.

Ytelsen er ganske dårlig, et større treningssett kan potensielt øke ytelsen.

5.2.3 Kryssvalidering

Kryssvalidering av de mest lovende egenskapskombinasjonene og klassifisererene fra den initielle testingen.

Jeg har desverre ikke hatt tid til å gjennomføre alle testene jeg har sett for meg.

Her skulle det vært plott med kjøringer av forskjellige tilfeldige treningssett/testsett. Fra disse hadde jeg tenkt å beregne konfidensintervall (se kap[2.9.2]). Dette konfidensintervallet ville vært gyldig for klassene jeg har i datasettene mine.

6 Diskusjon

I dette kapittelet reflekterer jeg over prosjektet, gjennomføringen, og resultatene jeg har komt frem til.

6.1 Om prosjektet

Da jeg valgte prosjektet som min masteroppgave var det den praktiske orienteringen og vanskelighetsgraden som først tiltrakk meg.

Å klassifisere mellom mange forskjellige klasser er ikke lett. Mange av klassene er svært like, noen av dem har svært små forskjeller. Å finne et godt ytelsesmål er utfordrende. Se app[B] for informasjon om datasettene og klassene.

Prosjektet er å finne metoder for å klassifisere typeskilt fra bilder. Bildene er antatt å være på en konsistent form (se kap[1.4]).

Prosjektet ble gitt av Verico AS i samarbeid med UiS.

6.2 Gjennomføringen

I forkant av oppstarten av selve prosjektet måtte jeg få tak i nødvendige data til å trene og teste på. Jeg måtte behandle dataene for å få dem på formen som prosjektet bruker i input. Forarbeidet er beskrevet i appendix[A]. Jeg regnet med å bruke opptil en uke til sammen på dette. Bildene til datasettene ble gitt av Verico AS.

Prosjektet handler om å klassifisere typeskilt, og for å løse dette hovedmålet har jeg fokusert på egenskapsutvikling. Jeg har i prosjektet lagt mest tid implementering og eksperimentering av masse forskjellige egenskaper.

Dersom jeg kunne gå tilbake og starte på nytt ville jeg nok ha planlagt litt bedre i forhold til implementeringen og tidsbruk. Det gikk mye tid med på utvikling og implementering/eksekvering av forskjellige egenskaper, flere enn dem nevnt i rapporten. De er ikke tatt med da de fleste av dem var svært dårlige og ustabile. Jeg forkastet da disse. I stedet for å skyte fra hoftaburde jeg på et tidlig tidspunkt heller studert problemstillingen bedre og komt frem til den endelige løsningen raskere.

Jeg fikk ikke tid til å utføre alle eksperimentene jeg ønsket. For å kunne danne et statistisk representativt resultat med konfidensintervall må flere uavhengige tester utføres. Jeg hadde planlagt å kjøre flere kryssvalideringer og bruke resultatene til å danne et konfidensintervall for ytelsen til klassifisereren (dette er da nevnt som videre arbeid).

6.3 Resultatene

I gjennom prosjektet har jeg brukt mye tid til å finne gode egenskaper som gir bra ytelse for alle klassene. Kap[5] presenterer resultatene fra utvalgte eksperimenter. Ikke alle kombinasjoner av egenskaper er tatt med i rapporten siden eksperimentene tar lang tid å kjøre, jeg har tatt med nok til å presentere løsningen på problemet.

Som tabell[1] viser er ikke de enkle egenskapene særlig gode i seg selv. Ytelsen til disse ligger rundt 10-30%.

Den endelige løsningen I mai kom jeg endelig på en god løsning etter å ha prøvd mange forskjellige egenskaper og kombinasjoner av disse. Da jeg gikk fort tom for ideer om nye egenskaper, bestemte je meg for en ny tilnærming på problemet. I stedet for å utvikle stadig flere og flere typer egenskaper med utgangspunkt i matematikken og statistikken la jeg til side bøkene og forsøkte å heller konkretisere hvordan mennesker ville utføre klassifisering av skiltene. Et menneske leter etter diskriminerende effekter, og med utgangspunkt i (kun) disse utfører klassifisering. I en egenskapsvektor trenger kun en av egenskapene representere en diskriminerende effekt. Dette søket etter diskriminerende effekter kan implementeres med å dele et bilde opp i blokker der det for hver blokk beregnes en egenskapsvektor. Dette vil da emulere hvordan et menneske sorterer bilder basert på små forskjeller.

Jeg kom frem til det jeg i rapporten kaller for lokale egenskaper"(beskrevet nærmere i kap[3.3]). Etter jeg implementerte dette fikk jeg momentant mye bedre ytelse enn alle andre egenskapskombinasjoner. Det enkle er ofte det beste, også i dette tilfellet. Ved å bruke en

Bayes klassifiserer med en klassespesifik normalfordeling kombinert med lokale gjennomsnitt fikk jeg en svært god ytelse.

Jeg har fått den beste ytelsen ved å beregne egenskapsvektoren fra kantbildene. Kantbildene vil ha mindre påvirkning av ytre faktorer som lysforhold, regndråper, slitasje osv.

Om klassifiseringsresultatene fra ikke-parametriske klassifiserere Jeg har gjennom hele prosjektet brukt et ganske lite treningssett med kun opp til 10 bilder per klasse (i noen tilfeller enda færre enn dette). De ikke-parametriske estimatorene (for de klassespesifike tetthetsfunksjonene) trenger flere treningsdata for å gi god ytelse. Dette kan sees i fig[18] der klassifisereren basert på klassespesifike normalfordelinger har best ytelse. Se app[B] for detaljer om datasettene og klassene.

6.4 Konklusjon

Oppgavens hovedmål gikk ut på å klassifisere bilder av typeskilt. Da det er mange forskjellige klasser av skilt har det vært vanskelig å finne en god egenskapskombinasjon som gir respektabel ytelse for alle klasser. Etterhvert fant jeg ut at å beregne egenskaper over hele bildene ikke kan gi tilstrekkelig ytelse, og jeg kom etterhvert frem til å beregne egenskapene fra forskjellige områder i bildene ved å foreta en blokkinndeling.

En blokk-basert egenskapsvektor har vist seg å kunne gi svært god ytelse. **Opptil over 90% korrekt klassifisering** av et datasett med ca. 600 bilder og 15 klasser på enkelte tester!

6.5 Videre arbeid

Med resultatene jeg har komt frem til vil jeg si at prosjektet er en suksess. Det er derimot fortsatt mer arbeid som kan gjøres med klassifiseringen av skiltene.

Mine forslag:

- Videre testing av løsningen for å få en bedre ide om hvorvidt løsningen er optimal. Se kap[2.9.2] og [1].
- Jeg har ikke i prosjektet implementert en applikasjon for vanlig bruk. Den mest optimale løsningen (for eksempel fig[**??**]) bør implementeres spesielt for å kunne optimalisere bedre enn det jeg har gjort i prosjektet.
- Jeg har kun prøvd ut 3 typer klassifiserere. Det finnes mange flere og mer avanserte enn de 3 jeg har prøvd.
- For å detektere potensielle nye klasser og dynamisk trening av klassifisereren finnes såkalte "clustering"algoritmer. Disse ser etter områder i egenskapsrommet med stor tetthet av data, dersom høy tetthet av data dukker opp utenfor de kjente områdene indikerer dette en ny klasse.
- Algoritmer som hindrer overtrening og undertrening bør brukes. Dette gir mer optimal trening enn det jeg har fått i prosjektet med et fast treningssett, og bør da gi økt ytelse.
- For parameterestimering av normalfordelingen har jeg brukt MLE (maximum likelihood estimation), andre metoder for parameterestimering finnes, blandt annet LMS (least means squared), RLS (rekursiv LMS med glemmefaktor), Kalman filter m.m.

Se boken *Pattern Classification* [6] for mer informasjon.

Referanser

[1] Per Chr. Hagen. Innforing i sannsynlighetsregning og statistikk. Cappelen akademisk forlag, 5th edition, 2007.

[2] The MathWorks Inc. Matlab documentation, 2012.

[3] Rune Johanssen. Effektivisering av datafangst fra merkeskilt med bruk av bilde-warping og mønstergjenkjenning. Bachelor's thesis, Universitetet i Stavanger, 2009.

[4] John G. Proakis and Dimitris G. Manolakis. Digital Signal Processing, Principles, Algorithms, and Applications. Pearson Prentice Hall, fourth edition, 2007.

[5] Richard E. Woods Rafael C. Gonzalez and Steven L. Eddins. Digital Image Processing using MATLAB. Gatesmark Publishing, second edition, 2009.

[6] Peter E. Hart Richard O. Duda and David G. Stork. Pattern Classification. Wiley-Interscience, second edition, 2001.

[7] Eric W. Weisstein. Cubic spline. From MathWorld – A Wolfram Web Resource. http://mathworld.wolfram.com/CubicSpline.html Acceessed 11/06 2012.

[8] Wikipedia. n-sphere. Wikipedia Foundation Inc. http://en.wikipedia.org/wiki/N-sphere Accessed 12/06 2012.

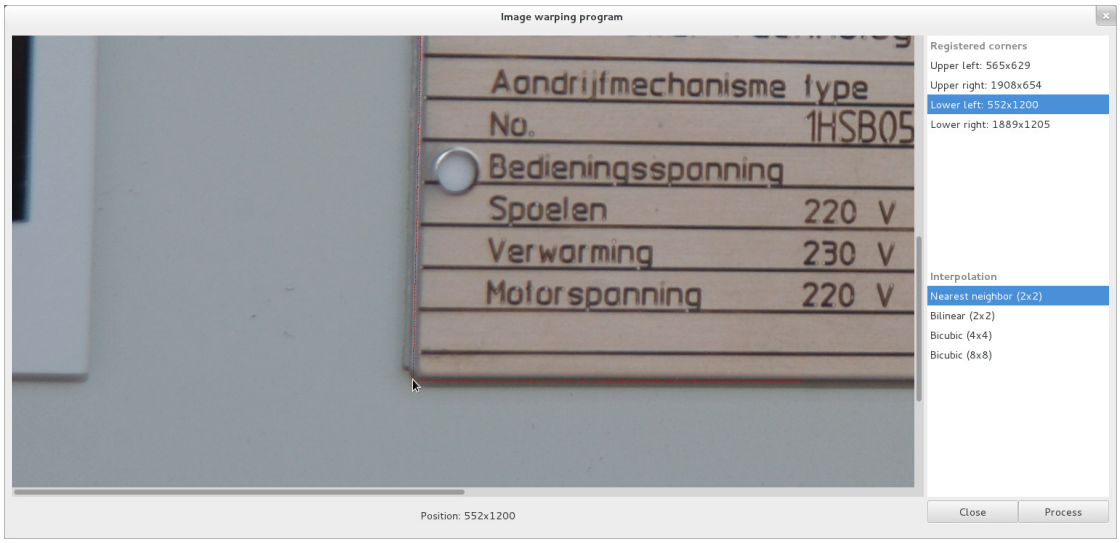
[9] Hough D. Young and Roger A. Freedman. University Physics with Modern Physics. Pearson Addison Wesley, 11th edition, 2004.

A Klargjøring og oppsett av datasettene

Prosjektet antar inndata i et spesifikt format; bilder som er utelukkende av skilt, uten bakgrunn og tatt perpendikulært i forhold til skiltet.

Bildene er i utgangspunktet ikke slik, de inneholder gjerne mye bakgrunn og i tillegg er noen skilt tatt på skrå (mange er tatt på skrå bevisst for å unngå refleks). Bildene som var tilgjengelig i forkant av dette prosjektet var rå originaler. Dermed har det vært nødvendig å gjøre en del bildebehandlig for å danne de forskjellige datasettene som brukes i denne oppgaven.

Jeg har benyttet meg av Rune Johansen's løsning i min implementering av warping-trinnet (trinn 2). Teorien her er en kort oppsummering og er hentet fra Rune's bacheloroppgave [3].



Figur 20: Markering av skilt i min implementering av warping-steget.

A.1 Warping av bilder, teori

Transformasjoner som translerer/skalerer/roterer deler av et bilde kalles koordinat-transformasjoner. Som navnet antyder er det ikke selve pikselverdiene i seg selv som behandles, men heller piksel-koordinatene. Fig[21] illustrerer overgangen fra kilde til destinasjon.

Bildet ut kalles destinasjonsbildet"og er bildet etter transformasjonen, bildet inn kalles kildebildet"og er bildet pikselverdiene leses fra.

Selve koordinat-transformen av orden n er definert som følger (graf og vektorisert form):

$$\begin{bmatrix} u & v \end{bmatrix} = \sum_{x=0}^{n} \sum_{y=0}^{x} \begin{bmatrix} a_{xy} & b_{xy} \end{bmatrix} \cdot m^{x-y} \cdot n^{y} \tag{51}$$

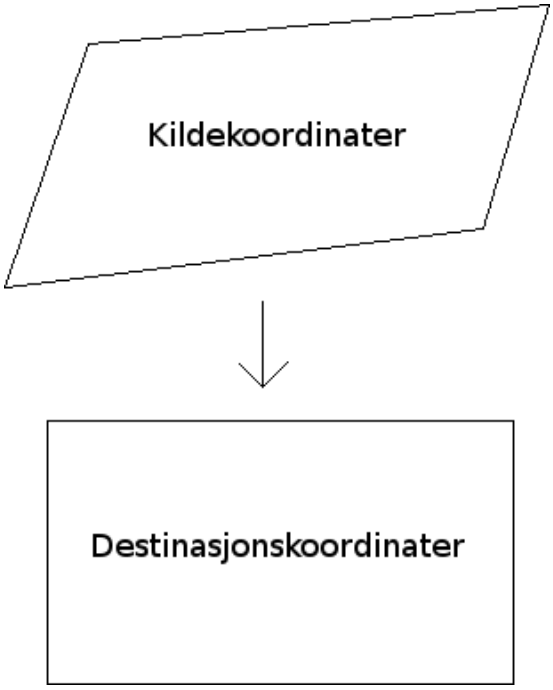
$$\begin{bmatrix} \underline{u} & \underline{v} \end{bmatrix} = \underset{\sim}{D} \begin{bmatrix} \underline{a} & \underline{b} \end{bmatrix} \tag{52}$$

Der $\begin{bmatrix} \underline{u} & \underline{v} \end{bmatrix}$ er kildekoordinater, $\underline{a}$ og $\underline{b}$ er koeffisientene til transformen, og $\underset{\sim}{D}$ er en matrise fylt inn med koordinater fra destinasjonsbildet:

$$\underset{\sim}{D} = \begin{bmatrix} 1 & \underline{m} & \underline{n} & \dots \end{bmatrix} \tag{53}$$

$\underline{m}$ og $\underline{n}$ i ligning[53] stammer fra ligning[51], for å finne en høyere ordens warp velg en større n og fyll inn flere elementer (ledd) i matrisen $\underset{\sim}{D}$.

Koeffisientene $\underline{a}$ og $\underline{b}$ finnes med minste kvadraters metode vha. kontrollpunkter mellom kilde- og destinasjons-bildet.



Figur 21: Transformasjon fra destinasjonskoordinater til kildekoordinater.

$$\begin{bmatrix} \underline{a} & \underline{b} \end{bmatrix} = \left(\underset{\sim}{D}^T \underset{\sim}{D}\right)^{-1} \underset{\sim}{D}^T \begin{bmatrix} \underline{u} & \underline{v} \end{bmatrix} \tag{54}$$

Dette er den pseudoinverse av $\underset{\sim}{D}$ multiplisert med korresponderende kildekoordinater.

Det trengs $(1+n) + \sum_{i=1}^{n} i$, antall kontrollpunkter for en warp av orden n.

Interpolering For å kunne lese av koordinater som ikke er heltall må vi interpolere mellom nabo-pikslene. Vanlige metoder for interpolering er:

- Nærmeste nabo.
- Bilineær.
- Bikubisk. [7]

Nærmeste nabo gir et veldig pikselert"og kantete resultat. *Bilineær* er litt bedre, men kan ofte gi et litt uskarpt resultat. *Bikubisk* er den beste av disse metodene, resultatet er en god del skarpere enn det bilineær interpolering gir.

Implementering Denne typen transformasjoner implementeres ved å iterere over destinasjonsbildet, transformere hvert av destinasjonskoordinatene til kildekoordinater, og så benytte interpolering for å lese av pikselverdiene ved de transformerte koordinatene fra kildebildet.

Rune's implementasjon tar utgangspunkt i at et ferdig warpet bilde er tilgjengelig og viser hvor korresponderende punkter skal markeres i kildebildet.

Jeg valgte å lage min egen implementasjon basert på Rune sin, modifisert til å basere warpen kun på hjørnekoordinatene til skiltet. Fig[20] viser GUI'en til applikasjonen jeg laget.

A.2 Warping og kutting, resultater

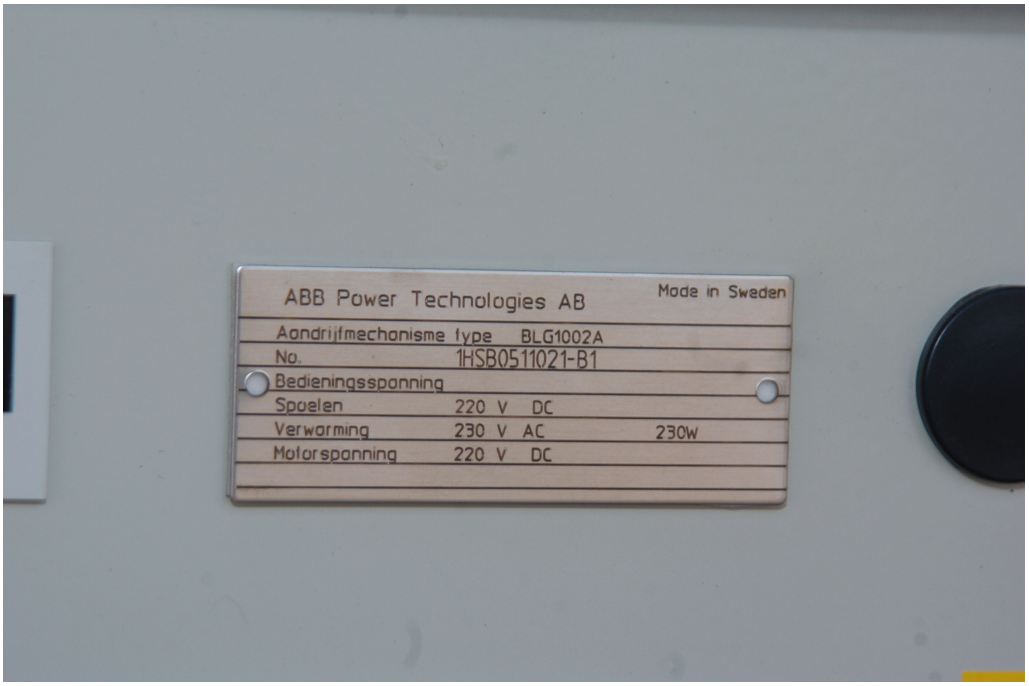
I implementasjonen av preprosesseringen har jeg ganske enkelt valgt en 2. ordens warp med kun hjørnene som kontrollpunkter, de andre kontrollpunktene som trengs beregnes lineært

med utgangspunkt i hjørnene (en 2. ordens warp krever minst 6 kontrollpunkter). Kutting utføres implisitt ved at et utgangsrektangel mappes kun til inngangskoordinater tilhørende skiltet i bildet.

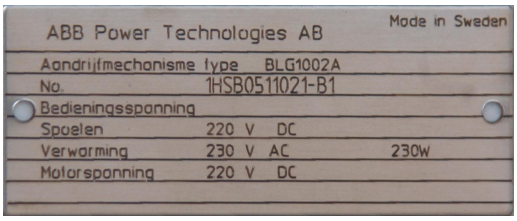
Prosessen kan deles inn i følgende trinn:

1. *Registrer hjørnekoordinater manuelt ved å klikke på bildet.*
2. *Lag utgangsrektangel.*
3. *Estimer flere kontrollpunkter ut fra hjørnekoordinatene.*
4. *Finn koeffisientene* a *og* b*.*
5. *Transformer alle kildekoordinatene fra utgangsrektangelet til inngangskoordinater.*
6. *Bruk interpolering til å beregne pikselverdier fra inngangsbildet.*

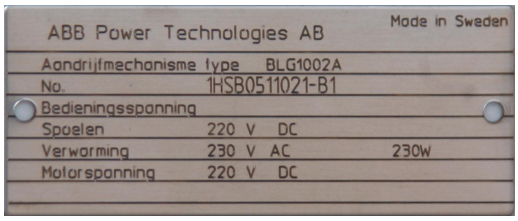
Jeg har brukt 4x4 kubisk interpolering, men bilineær interpolering gir også et tilstrekkelig godt resultat. Fig[22] viser 4 resultater, en for hver interpoleringsmetode. Som en kan se er det svært liten forskjell mellom dem. Bikubisk interpolering gir en større filstørrelse på resultatet enn de andre, dette indikerer at bildet inneholder mer informasjon.



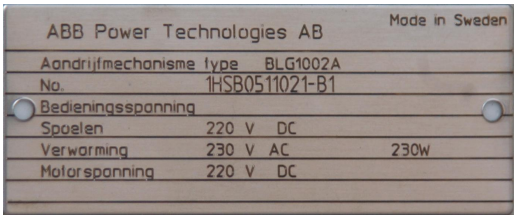
(a) Rå-bilde rett fra kamera.



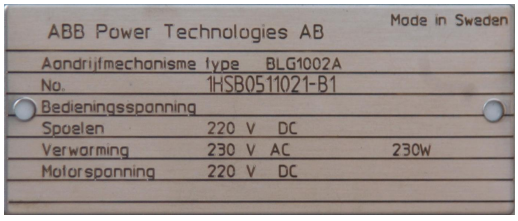
(b) Nærmeste nabo (2x2).



(c) Bilineær (2x2).



(d) Bikubisk (4x4).



(e) Bikubisk (8x8).

Figur 22: Resultater fra warping med forskjellige interpoleringsmetoder. Det er vanskelig å se forskjell mellom bilineær og bikubisk, sistnevnte har en anelse bedre kontraster. Bildet er warpet fra bildet på fig[20]

B Datasett og klasser

Her beskrives datasettene og klassene som er brukt i eksperimentene. Bildene til datasettene ble gitt av Verico AS.

Det er 4 forskjellige datasett:

- Alle - alle bildene samlet.
- Trening - bildene brukt til å trene klassifisereren.
- Testing - testsett nr 1.
- Verifisering - testsett nr 2.

B.1 Klassedefinisjoner

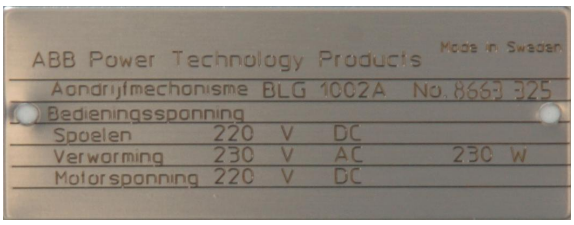
Definisjon av alle de forskjellige klassene som er brukt i prosjektet. Tab[3] lister antallet bilder i hvert av datasettene. Fig[23] - fig[32] viser et bilde fra hver klasse (rekkefølgen er laget for å utnytte sidene best mulig, underteksten viser klassetilhørighet for hvert av bildene).

Det er 655 bilder totalt.

Datasettene er konstruert for stresstesting av egenskaper. Noen klasser er nesten helt like andre, forskjellene ligger i plasseringen av datafeltene.

Klasse	Treningsdata	Testdata	Verifiseringsdata
0	10	12	13
1	10	14	35
2	4	5	5
3	4	5	5
5	4	4	4
6	10	6	5
7	10	14	21
8	10	22	39
9	10	14	11
10	10	14	13
11	10	26	44
12	10	70	65
13	6	5	6
14	10	28	29
16	10	15	18

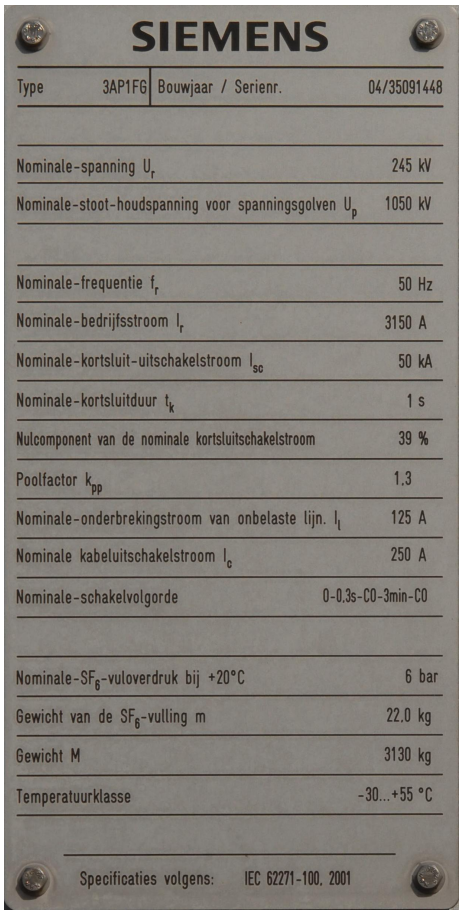
Tabell 3: Antall bilder i hvert datasett for hver klasse.



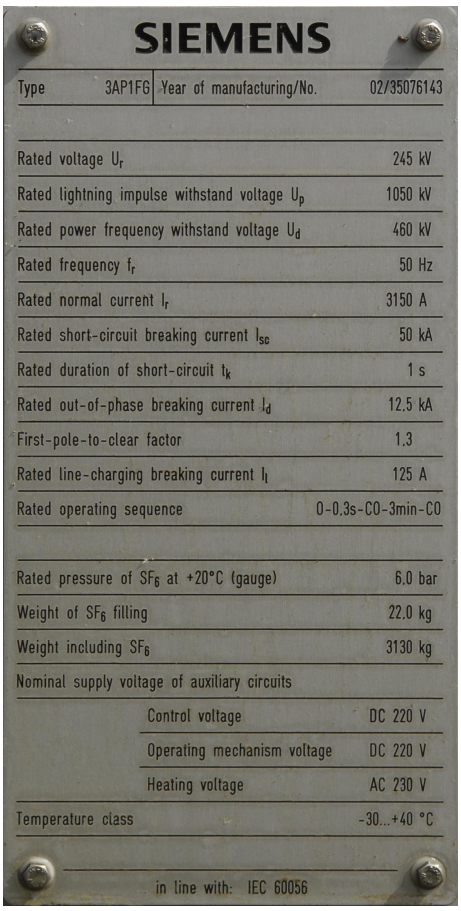
Figur 23: Klasse 2.

Figur 24: Klasse 3.

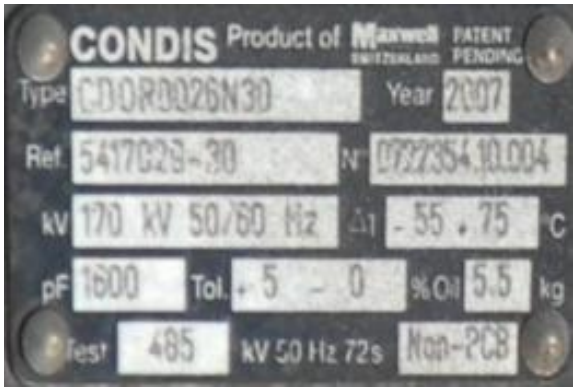
Figur 25: Klasse 9.



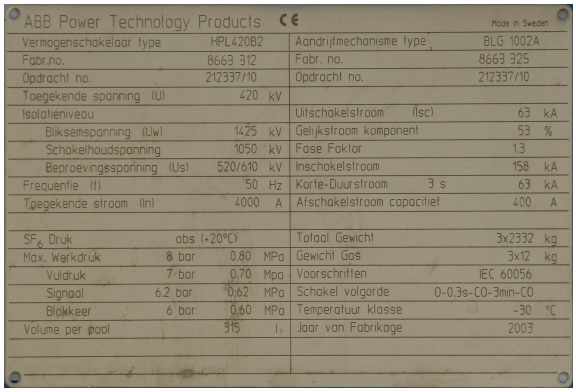
Figur 26: Klasse 0.



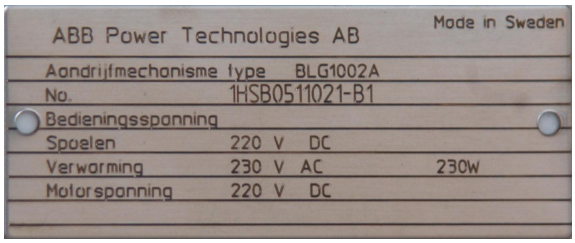
Figur 27: Klasse 13.



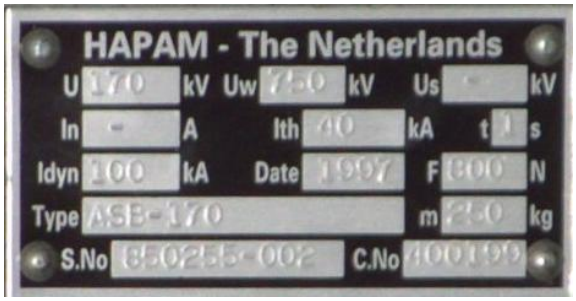
Figur 28: Klasse 5.



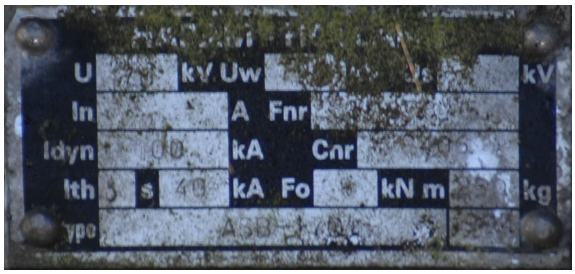
Figur 29: Klasse 6.



Figur 30: Klasse 7.

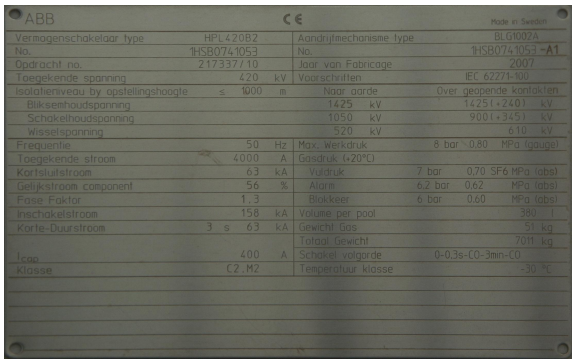


Figur 31: Klasse 8.

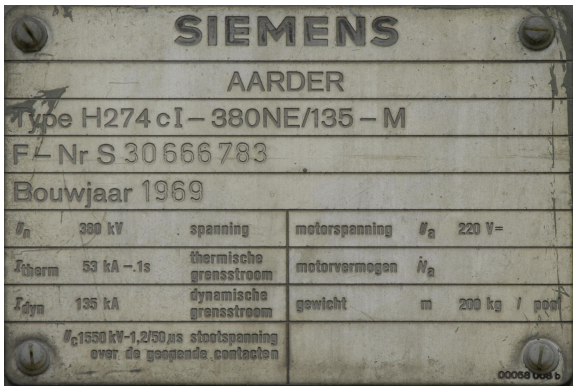


Figur 32: Klasse 16.

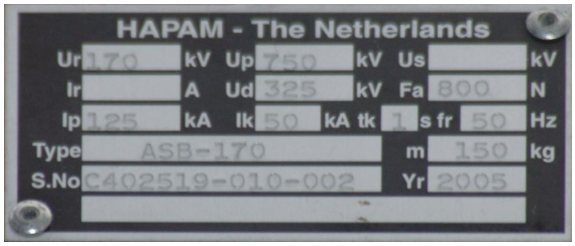
Figur 33: Klasse 10.



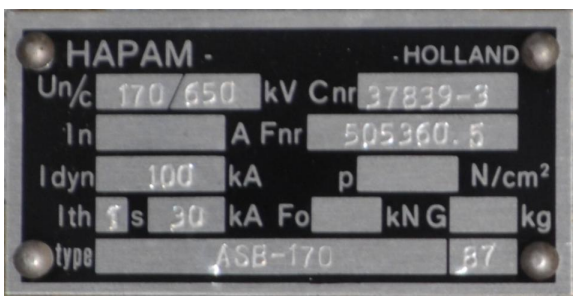
Figur 34: Klasse 1.



Figur 35: Klasse 12.



Figur 36: Klasse 11.



Figur 37: Klasse 14.

C Kodelisting

De viktigste kodefilene fra implementeringen listet og forklart.

C.1 Klassifiserer

Koden for selve klassifiserersystemet.

Filnavn	Argumenter	Returnerer	Beskrivelse
`k_init`	folder features featureargs name	klassestrukturer	Setter opp en generell struktur for en eller flere klasser, fig[**??**]. Fyller inn generelle data: filstier til treningsdata til medlemmet `files`.
`k_crossvalid_init`	folder features featureargs N name	klassestrukturer	Som `k_init.m`, men kaller `u_rand_datasett` i stedet for `u_datasett_training`. N er antall tilfeldige treningsdata per klasse.
`k_normal_init.m`	class_in	class_out	Fyller inn klassespesifike datafelter i klassestrukturen, fig[**??**]. Medlemmet `parms` holder på klassespesifik informasjon (P_w, $\underline{\mu}$, og $\underset{\sim}{\Sigma}$).
`k_parzen_init.m`	class_in	class_out	Fyller inn klassespesifike datafelter i klassestrukturen, fig[**??**]. Medlemmet `parms` holder på klassespesifik informasjon ($\underline{\mu}$, h_n, og V_n).
`k_knearest_init.m`	class_in	class_out	Fyller inn klassespesifike datafelter i klassestrukturen, fig[**??**]. Medlemmet `parms` holder på klassespesifik informasjon ($\underline{\mu}$, og k_n).
`k_normal_train.m`	class_in classifier_in	class_out	Trener klassifisereren med MLE. P_w estimeres som $1/\mathsf{numel}(\mathsf{classifier\_in})$. Parametrene som trenes ligger i medlemmet `class_in.parms`.
`k_parzen_train.m`	class_in classifier_in	class_out	Trener klassifisereren. Beregner $\underline{\mu}$ via MLE, $h_n = h_1/\sqrt{n}$, $V_n = h_n^d$. Parametrene som trenes ligger i medlemmet `class_in.parms`.
`k_knearest_train.m`	class_in classifier_in	class_out	Trener klassifisereren. Beregner $\underline{\mu}$ via MLE, $k_n = \lfloor\sqrt{n}\rceil$. Parametrene som trenes ligger i medlemmet `class_in.parms`.
`k_train.m`	class_in classifier_in	class_out	Trener klassifisereren basert på datamedlemmet `class_in.type`.
`k_klassifiser.m`	classes_in x	i	Klassifiserer egenskapsvektoren `x`. `classes_in` er en liste av klasser, og `i` er indeksen til den tilordnede klassen.

Diskriminantfunksjoner - til de 3 typene klassifiserere. Strukturen `class.discriminant` holder en funksjonspeker til en av disse (settes i funksjonene `k_*_init.m`).

Filnavn	Argumenter	Returnerer	Beskrivelse
`d_normal.m`	class_in x classifier_in	gi	Beregner diskriminanten til en klasse for egenskapsvektoren `x`, her en normalfordeling.
`d_parzen.m`	class_in x classifier_in	gi	Beregner diskriminanten til en klasse for egenskapsvektoren `x`, her et parzen-estimat. For hver av treningsdataene beregnes en lokalt definert vindusfunksjon.
`d_knearest.m`	class_in x classifier_in	gi	Beregner diskriminanten til en klasse for egenskapsvektoren `x`, her et knearest-estimat. A posteriori sannsynlighet beregnes fra `classifier_in` via $gi = k_n/k$.

C.2 Analyse

Toppnivåfiler som brukes til å kjøre eksperimenter og analyserer resultatene fra klassifiseringene.

Filnavn	Argumenter	Returnerer	Beskrivelse
`a_create_datastructure.m`	klasse sett	d	Setter opp en datastruktur som holder styr på filstier, sann klassetilhørighet, og klassifisert klassetilhørighet. Opererer på kun ett datasett om gangen (trening, testing, eller verifisering).
`a_classify_datastructure.m`	classes datastructure_in	datastructure_out	Bruker `u_feature_calc.m` til å beregne egenskapsvektorer fra bildene listet i `datastructure_in.files`. Klassifiserer så egenskapsvektorene og lagrer resultatene i `datastructure_out`.
`a_analyse_total.m`	datastructure	p	Estimerer total ytelse ved å telle opp antall korrekte klassifiseringer og beregne correct/numel(files).
`a_analyse_per_class.m`	datastructure	p	Samme som over, men teller opp for hver klasse separat. Resultatene lagres i en datastruktur `p` som en liste `p.pcnt`.

C.3 Datasett

Filer som implementerer en datakilde fra fig[38].

Filnavn	Argumenter	Returnerer	Beskrivelse
`u_datasett_training.m`	klasse	datasett_training	Går gjennom mappestrukturen fra fig[11] og lager en liste av alle filstier til bildene av klassen gitt som argument. Gir kun bilder fra testingsettet.
`u_datasett_testing.m`	klasse	datasett_testing	Går gjennom mappestrukturen fra fig[11] og lager en liste av alle filstier til bildene av klassen gitt som argument. Gir kun bilder fra treningssettet.
`u_datasett_verif.m`	klasse	datasett_verif	Går gjennom mappestrukturen fra fig[11] og lager en liste av alle filstier til bildene av klassen gitt som argument. Gir kun bilder fra verifiseringssettet.
`u_datasett.m`	klasse sett	datasett_ut	Går gjennom mappestrukturen fra fig[11] og lager en liste av alle filstier til bildene av klassen gitt som argument. Gir kun bilder fra datasettet `sett`.
`u_rand_datasett.m`	klasse sett N	datasett_ut	Samme som `u_datasett.m`, men setter sammen datasettet fra N tilfeldige bildeindekser.

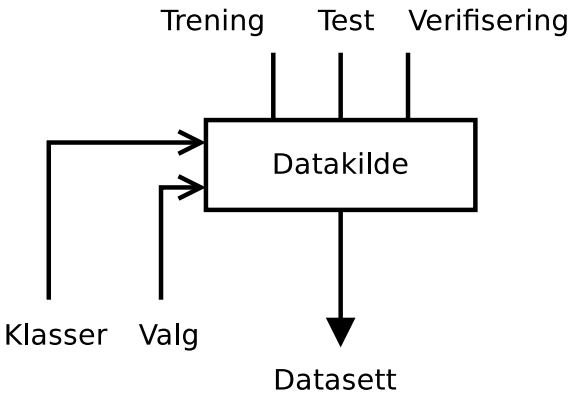
C.4 Egenskaper og diverse

Egenskaper og diverse andre vesentlige funksjoner.

Filnavn	Argumenter	Returnerer	Beskrivelse
`e_*.m`	img args	e n	Egenskapsfunksjoner. Beregn `n` stk. egenskaper fra bildedata `img`. Argumentet `args` er valgfritt og kan defineres per egenskap.
`e_grid.m`	img args	e n	Lokal egenskap. Deler bildet `img` opp i [M,N] blokker vha. MATLAB's `blockproc`. [M,N] gis via argumentet `args.M` og `args.N`. Hvilken egenskap som beregnes per blokk defineres inni funksjonen (pga. en begrensning i `blockproc`).
`u_feature_calc.m`	feature_handles feature_args files	x	Beregner en egenskapsvektor for hver av bildene i `files`. Funksjonen leser filene og kaller hver egenskapsfunksjon listet i `feature_handles`. Egenskapsvektorene fra hver egenskapsfunksjon settes sammen til en total vektor `x`.

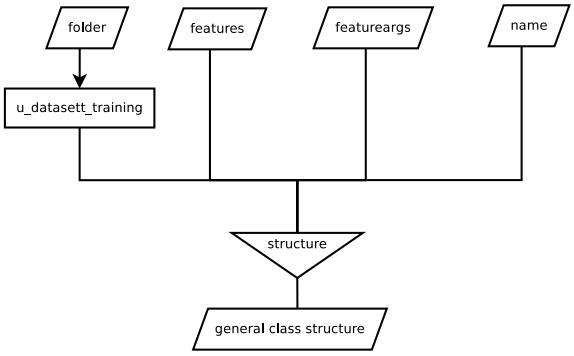
C.5 Blokkdiagram

Blokkdiagram for de viktigste delene av implementeringen er vist i fig[38] - fig[43]. Diagrammene er ment å gi en oversikt og er derfor forenklet.

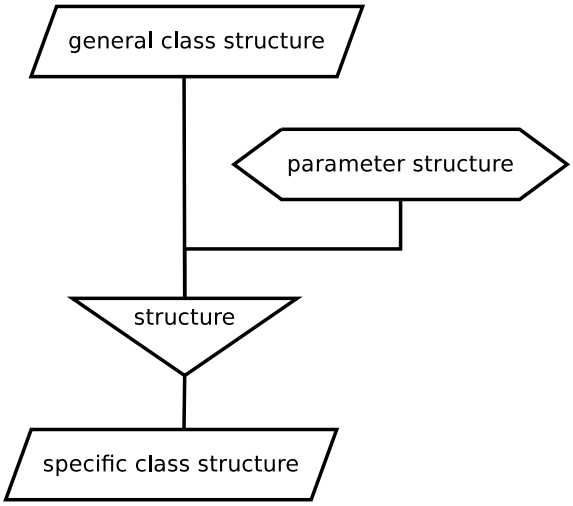


Figur 38: Datakilde. Setter sammen en liste av filer fra de valgte klassene og det valgte datasettet.

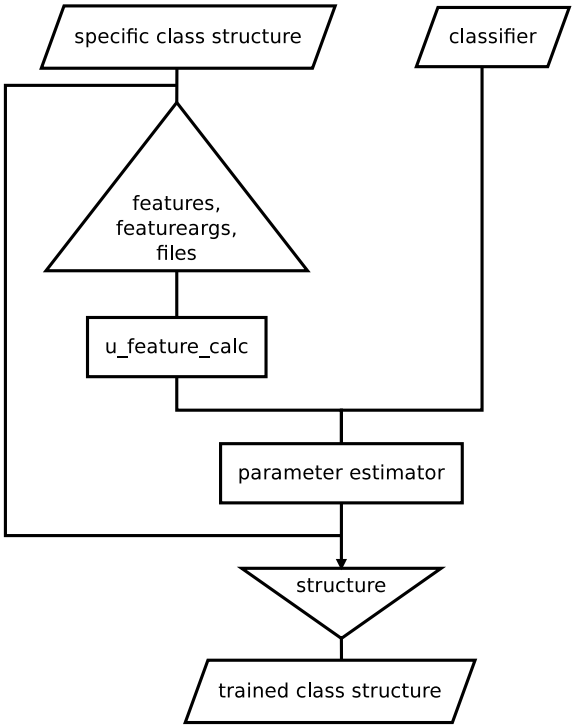
Figur 39: Klassifiseringssystemet er implementert som en liste av klasseobjekter.



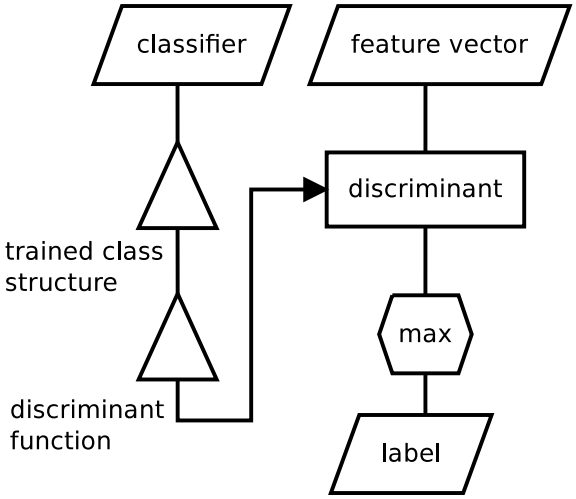
Figur 40: Initialisering av generelle klasser. Input `folder` er en liste av mapper korresponderende til klassene.



Figur 41: Initialisering av type-spesifike detaljer.



Figur 42: Trening av klassifisererene. Med pyramideblokken menes det å plukke ut datamedlemmene som er listet. Treningsdata er listen `files`.



Figur 43: Klassifiseringsprosessen. Her plukkes diskriminantfunksjonen ut fra hver `classifier`-objekt.