Maskiner som ser: En implementation av aniktsigenkänning

Advanced Image Processing - TNM034

Wilhelm Björkström, Emil Wallberg & Theodor Östling

11 december 2023

Innehåll

Kapitel 1

Introduktion

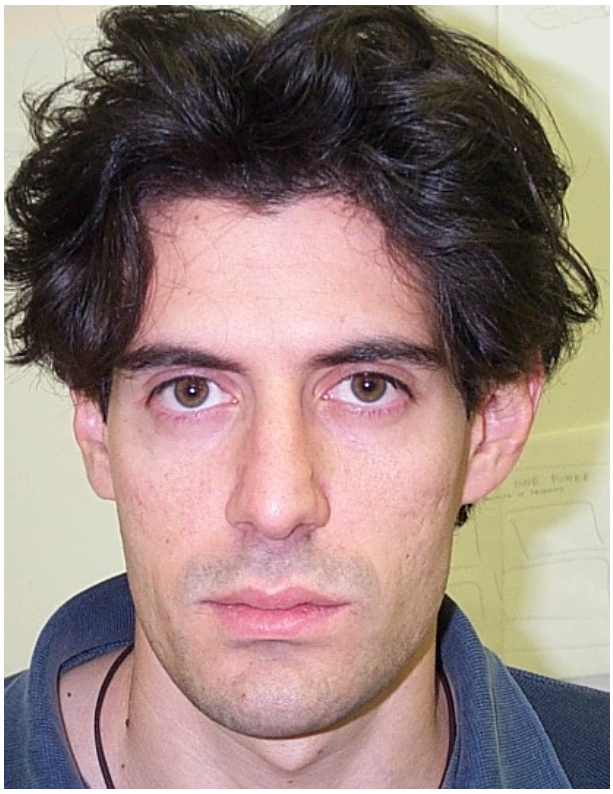
Det moderna samhället är en värld utformad av datorer och annan digital teknik. Vart än blicken hamnar runt omkring dig är det inte så svårt att hitta en teknisk innovation som den vardagliga människan använder men ej vet hur saken, till fullo, fungerar. Tekniken har med galopperande steg utvecklats och därför gjort det mycket svårt att veta hur saker och ting fungerar samt hur de faktiskt är uppbyggda när det gäller teknik. Ett bra exempel på något sådant är exempelvis hur ansiktsigenkänning fungerar på din smarttelefon eller dörr för att ta dig in till jobbet. Hur kan en enhet omvandla en bild på dig för att sedan fastställa om det är du eller någon annan person som tittat in i kameran?

I denna rapport kommer en ansiktsigenkännings metod presenteras och diskuteras. Metoden kommer inte bygga på maskininlärning som idag är en mycket vanlig metod för det mesta gällande igenkänning. I stället kommer en något äldre metod implementeras som bygger på en avancerad bildbehandling för ansiktsdetektion. Genomförandet kommer ske i det matematikorienterade programmeringsverktyget *MATLAB 2023a*.

Kapitel 2

Bakgrund

Som nämnt i föregående kapitel ska ett program för bildigenkänning implementeras i MATLAB. Programmet kommer inte grunda sig i maskininlärning utan istället arbeta med olika avancerade bildbehandlingprocesser för att försöka finna ett ansikte i en given bild. Sedan ska två koordinater för varje öga ges ut utifrån ansiktet. Dessa koordinater ska sedan bilden normaliseras utifrån för att sedan skickas in till en PCA och Fisherfsces algoritm. Det slutgiltiga resultatet blir sist ett ID-nummer som tillhör den person programmet ser ur bilden. De bilder som kommer användas finns i tre olika databaser DB0, DB1 och DB2 som alla kommer från *"Faces 1999"* utgiven av Caltech. DB0 består av 4 ansikten som inte ska kännas igen i systemet. Om ett ansikte inte känns igen ska ett ID-nummer på 0 returneras som innebär att ansiktet inte känns igen. DB1 innehåller en bild för samtliga personer som ska kännas igen som totalt är 16 stycken. Bilderna i DB1 är också med ansiktet i fokus och inte speciellt mycket bakgrund med i bilden. Ett exempel på en bild ur DB1 går att se i Figur 2.1 som har ID-nummer 14. DB2 innehåller en rad olika bilder som har mycket mer bakgrund med i bilden samt överexponerade eller andra potentiella svårigheter för igenkänningen med i bilden (totalt 38 bilder på de 16 olika personerna).



Figur 2.1: En obehandlad bild från DB1 på personen med ID-nummer 14.

Kapitel 3

Metod

Detta kapitel ämnar att redovisa de steg moment som genomförs för att få ett resultat till problemet. Till en början presenteras en metod för ögonigenkänning på följd av en bildnormalisering samt PCA-modell.

3.1 Ögonigenkänning

Det kanske mest kritiska momentet för en ögonigenkännings algoritm är att hitta själva ögonen i bilden. Men hur ser processen ut för att finna ögonen i en bild? Det är detta som detta stycke ämnar att redovisa.

3.1.1 Färgkorigering

Innan någon form av operation för att hitta ögon i bilden kan genomföras behöver bilden genomgå diverse färgkorrigeringar. Denna åtgärd syftar till att ge samma förutsättningar för samtliga maskeringsoperationer genom att standardisera färg och ljusstyrka, särskilt för hudfärg, i hela bilden. Med andra ord är färgkorrigeringen nödvändig för att säkerställa att exempelvis en överexponerad bild har liknande hudfärgvärden som en normalt exponerad bild. Detta steg är därmed avgörande för att uppnå en jämförbar och konsekvent grund inför ögonigenkänning.

En typ av färgkorrigeringar är en så kallad RGB-korrigering. Denna operation ämnar att korrigera RGB balansen i bilden separat för varje kanal vilket resulterar i en form av vitbalans. Detta görs genom att multiplicera varje kanal med dess normaliserade medelvärde som i Ekvation 3.1 där C representerar en färgkanal. Den slutgiltiga bilden blir därmed erhållen enligt Ekvation 3.2.

$$C_{cor} = C \cdot \frac{C_{avg}}{C_{max}} \tag{3.1}$$

.

$$I_{cor} = \begin{pmatrix} C_{corR} \\ C_{corG} \\ C_{corB} \end{pmatrix} \tag{3.2}$$

Utöver den tidigare nämnda ljusbalanseringsmetoden går det tillämpa en metod som bygger på specifika alpha och beta värden ur bilden [1]. Dessa värden definieras utifrån två olika kvoter mellan färgkanalerna. Detta görs enligt Ekvation 3.3 och 3.4. Efter beräkningen av alpha och beta värdet beräknas den färgkorrigerade bilden som i Ekvation 3.5. Notera att denna typ av färgkorrigering kan uppbåda problematik om stora mängder av samma färg skulle finnas i bakgrunden.

$$\alpha = \frac{G_{avg}}{R_{avg}} \tag{3.3}$$

$$\beta = \frac{G_{avg}}{B_{avg}} \tag{3.4}$$

$$I_{AWB} = \begin{pmatrix} C_R \cdot \alpha \\ C_G \\ C_B \cdot \beta \end{pmatrix} \tag{3.5}$$

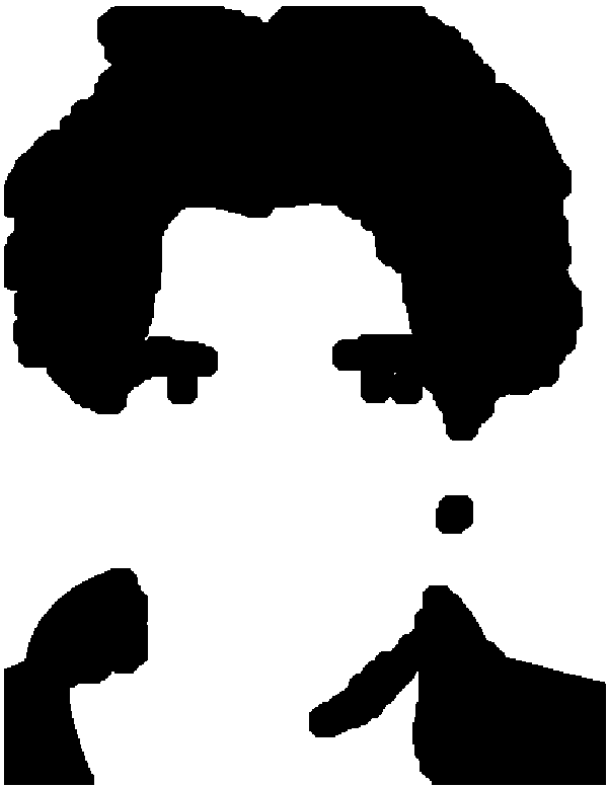
Det kan även vara intresant att utföra en kontrastutsträckning på bilden för att jämna ut mellan maximi och minimivärden i bilden. Metoden bygger på att det värdet som är störst i respektive färgkanal ska korrigeras till att vara det maximala färgvärdet och det minsta värdet i respektive färgkanal korigeras till att vara det minimala färgvärdet (som alltid är 0). Resterande pixlar skalas även om och ger på så sätt en större utsträgning för samtliga pixlar i bilden. Detta görs via Ekvation 3.6. Notera att funktionen enkelt kan ändras för att sträcka ut bilden mellan två önskade maximi och minimi världen istället för hela omfånget hos en kanal.

$$C_{stre} = \frac{C - C_{min}}{C_{max} - C_{min}} \tag{3.6}$$

3.1.2 Tröskelmask

Den första hudfärgsfiltreringen är en implementation av en grundläggande tröskelmaskering. Vid denna process tilldelas pixel i bilden antingen värdet 0 eller 1 beroende på om dess ljusintensiteten överstiger en viss tröskel. Denna tröskel bör definieras efter undersökningar där det optimala resultatet är en bild med enbart hudfärgsdelarna av bilden kvar. Det är dock viktigt att notera att en fullständigt perfekt filtrering via hudfärgsfiltreringen är i praktiken omöjligt, vilket i sin tur leder till att den resulterade trösklade bilden är en balans mellan optimal prestation och eliminering av icke-hudfärgsrelaterade element. För att tillämpa denna maskering på en RGB bild i MATLAB behöver först funktionen *rgb2gray* tillämpas då tröskningen är tänkt att utföras på en bild i gråskala (som ger

ljusintensiteten). Denna funktion returnerar en gråskalebild som är en viktad summa av de tre färgkanalerna. Resultatet av en tröskelmaskering går att se i Figur 3.1.



Figur 3.1: Den resulterande bilden vid tröskling av Figur 2.1 med en tröskel på 60%.

3.1.3 Sobelmask

Den andra masken som ska tillämpas på bilden för att underlätta ögonigenkänning är en så kallad Sobel-mask. Namnet på masken avslöjar att det är en kantdetekteringsoperator som tillämpas på bilden. Själva operationen är till synes enkel och tillämpas både lodrätt och horisontellt via en enkel faltning på bilden som går att se i Ekvation 3.9 [2]. De båda filterna representeras av Ekvatio 3.8 och 3.7.

$$H_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \tag{3.7}$$

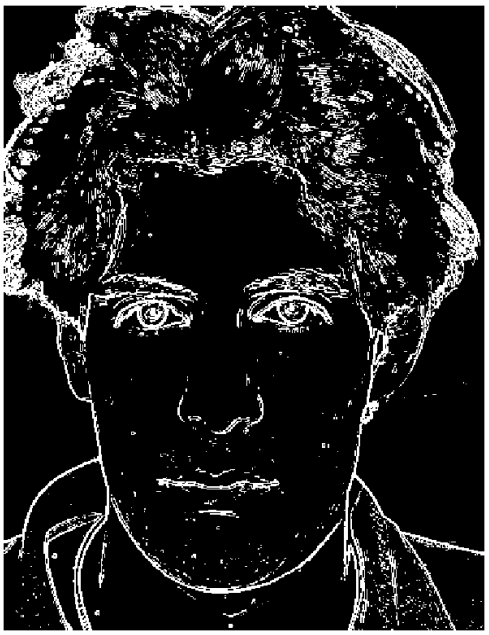
$$H_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \tag{3.8}$$

$$G_i = (I * H)(n, m) \tag{3.9}$$

Det går enkelt att se hur de båda filterna slår ut alla element utom kanter vid tillämpning på en bild och på så sätt ger en gradienten i x-riktningen och en i y-riktningen av bilden. Sist tillämpas Ekvation 3.10 för att ge den slutgiltiga sobelfiltrerade bilden.

$$I_{filt} = \sqrt{G_x^2 + G_y^2} \tag{3.10}$$

För att sedan får ut en mask av filtreringen sätts alla världen över ett tröskelvärde till 1 och resterande till 0. Tröskelvärdet bör utgå från olika tester där ett bra resultat definieras i en bild där enbart ansiktets konturer ingår. Det är dock i praktiken omöjligt att alltid få ett perfekt resultat med väldefinierade kanter vilket är viktigt att ha i åtanke vid tillämpning. Resultatet av sobel filtrering på Figur 2.1 går att se i Figur 3.2.

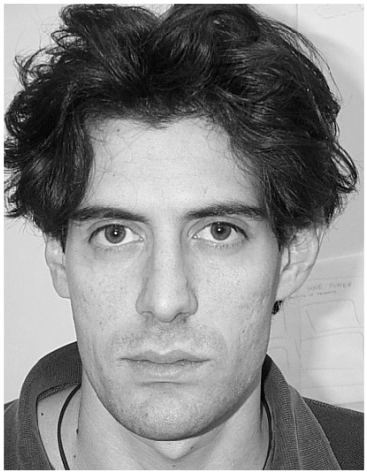


Figur 3.2: Den resulterande bilden vid sobelfiltrering av Figur 2.1 med en tröskel på 50%.

3.1.4 Hudfärgsmask

Den tredje masken som kan appliceras är en hudfärgsmask. Denna mask är den som har störst tyngd för huruvida ögonen kommer kunna hämtas ut ur bilden och behöver därför fungera väl. Det är nämligen denna mask som kommer att definiera vilket område som är ansiktet och därmed vilket område som ögonen befinner sig i. Med vetskapen om kraven på masken är det dags att gå över hur den faktiskt tas fram.

Till en början omvandlas bildens RGB kanaler till YC_gC_r färgskala. Det känns kanske ovant att se denna typ av färgdimension då YC_bC_r kan anses som något vanligare. I fallet för hudfärgsmasken är dock den gröna C-kanalen av större intresse än den blåa då grön färg är avsevärt bättre än blå vid detektion av hud [3]. Med omvandlingen klar är det helt enkelt dags att pröva fram mellan vilka olika värden för respektive kanal som hudfärg befinner sig inom.



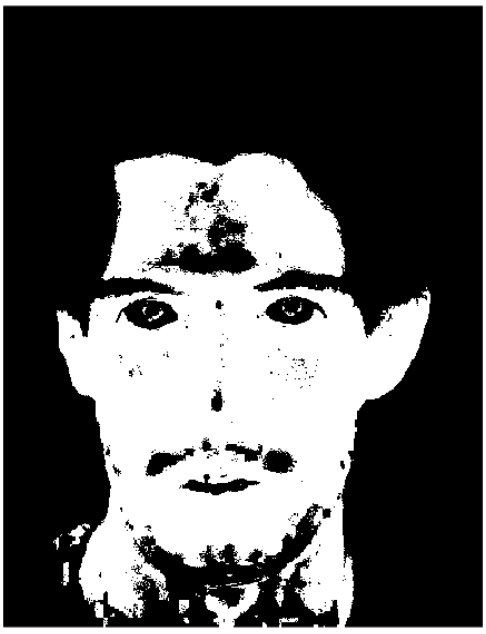
(a) En gråskalig representation av Y kanalen.

(b) En gråskalig representation av C_g kanalen.

(c) En gråskalig representation av C_r kanalen.

Figur 3.3: Figur 2.1 i gråskaligt YC_gC_r format istället för RGB format.

Det är även möjligt att blanda in hsv färgskalan för ytterligare prestation i att hitta hudfärg. Det blir då h-kanalen, vilket representerar "Hue", som blir den intressanta kanalen att studera världen för hudfärg [3]. Även här får prövning genomföras för att därmed ge ett lämpligt maximum- och minimumvärde för masken där H-kanalen tillämpas. Sedan kan de fyra olika maskerna kombineras till en enda hudfärgsmask som i Figur 3.4.



Figur 3.4: Resultatet efter att ha applicerat en hudfergsmaskering på Figur 2.1 med strikta gränser.

Vid tillämpning av hudfärgsmasken som ovan är det viktigt att ha definierat för sig om det är en exakt eller generell mask som krävs för att få med ögonen i masken. Vid ett enklare dataset där stora delar av bilderna är täckta av ansiktet med minimal delaktighet från bakgrunden kan det anses lämpligt att tillämpa en exakt mask. Om istället mer bakgrund är med i bilden och färgbalansen är sämre behövs det kanske mer generösa värden i hudfärgsmasken för att den ska ge världe vid tillämpning. Stor vikt bör därför läggas på hur datasetet ser ut innan värden för YC_gC_r och HSV maskerna definieras.

3.1.5 Morfologiska operationer

Ett sätt att förbättra en mask efter att den skapats är att tillämpa diverse morfologiska operationer. En sådan morfologisk bildbehandling är en morfologisk öppning som är tänkt att eliminera objekt utifrån ett givet strukturelementt. Öppningen genomförs genom att utföra erosion (krympning) följt av dilation (expansion) på en gråskallebild eller binärbild. Detta tillämpas i MATLAB via funktionen *imopen(I, SE)* där SE är ett strukturelement definierat via *strel(typ, storlek)*.

En annan morfologisk operation med till en öppning är en morfologisk stängning. Denna bildbehandling ämnar att fylla små hål och sammanfoga närliggande objekt i bilden. Detta görs genom att istället utföra dilation först för att sedan tillämpa erosion. För att göra detta i MATLAB används funktionen *imclose(I, SE)* där SE är ett strukturelement och bilden är en gråskallebild eller binärbild.

Utöver de två tidigare nämnda operationerna kan även erosion samt dilation utföras separat utan någon form av samband. Detta ger en större frihet i vilka strukturelement som ska användas och flera olika typer samt storlekar kan tillämpas i följd. Processen blir därmed att utifrån tillämpning testa sig fram ett bra resultat. Detta görs i MATLAB via *imclose(I,SE)* och *imopen(I,SE)* på samma sätt som tidigare morfologiska operationer i MATLAB.

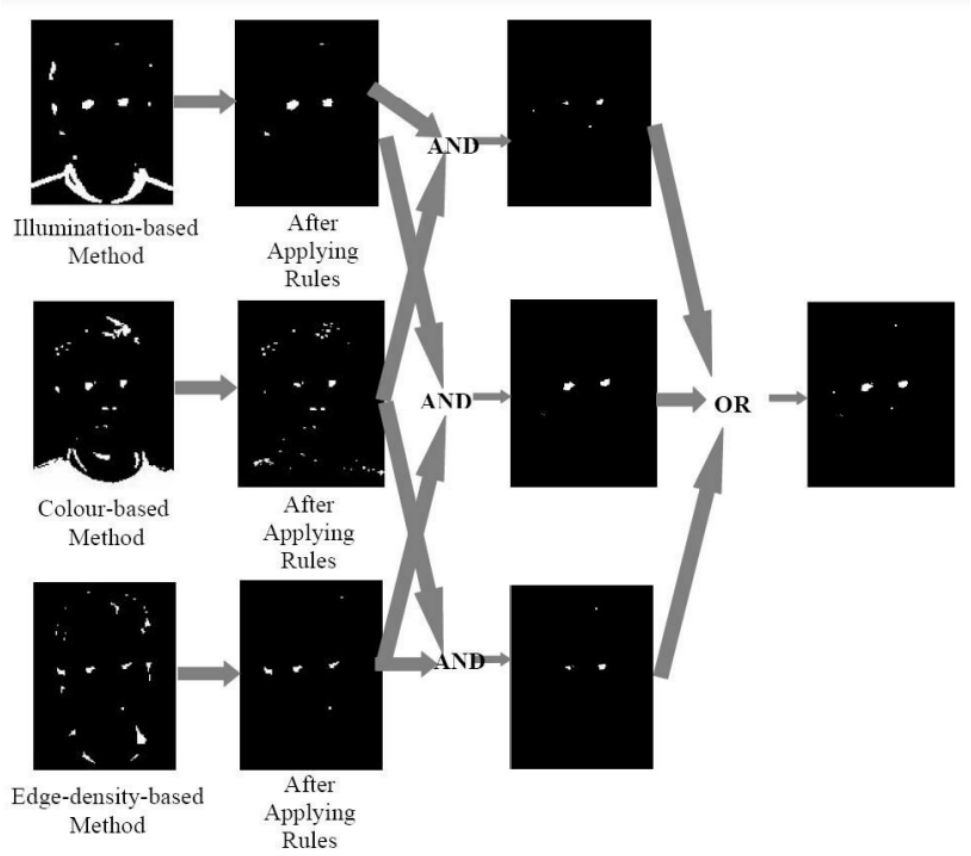
Sist kan det vara bra att filtrera bort mindre element ur bilden för att på så sätt ha kvar ett definierat antal av de största områdena i den maskerade bilden. I MATLAB görs detta via funktionen *bwareafilt(I, antal önskade områden)* som filtrerar bort så enbart antalet önskade områden finns kvar i bilden. Dessa områden representeras av de X antal största områden i bilden. Denna operation säkerställer att brus och andra störningar i masken elimineras och därmed ger en mer väldefinierad mask.

Ett exempel på hur en maskerad bild förändras utifrån en rad morfologiska operationer nämnda ovan går att se i Figur 3.5. Masken kanske ser ut att ha en sämre prestation efter operationen vilket delvis är korrekt. Den vinnande faktorn som klart kastar denna uppenbarelse i skugga är hur hela ansiktet nu kommer med vid tillämpning av masken samt att det finns en större marginal vid tillämpning för olika ansiktstyper. Detta gör därmed masken rimligare att tillämpa på flera olika ansikten och därmed inte överanpassad utifrån ett eller ett fåtal ansikten.

Figur 3.5: Resultatet efter tillämpningen av morfologiska operationer på Figur 3.4.

3.1.6 Slutgiltigmask

Med tre olika former av masker gjorda är det dags att ta fram den slutgiltiga masken för att maskera bort allt förutom ansiktet. Maskerna för sig själva är nämligen inte de starkaste (även om hudfärgsmasken klart skiner över de andra två) men vid en rad kombinationer kan en enda ordentlig mask erhållas [2]. Detta kan göras på olika sätt och ge varierande resultat. En aktuell metod redovisas i Figur 3.6 där en form av hybridmask tas fram av de tre maskerna. I stora drag fungerar metoden bra vid tillämpning och ger mycket bättre resultat än om var mask för sig tillämpas. Notera dock att om en stark hudfärgsmask har erhållits kan det vara aktuellt att blanda in den direkt i den slutgiltiga masken. Processen blir, igen, att en noggrann prövning utförs för att få fram en mask som ger önskvärda resultat i ögonigenkänning.



Figur 3.6: En ilustration över hur tre medelmåttiga masker kan kombineras för att ge en stark mask för ögonigenkänning (Muhammad Shaf et. al) [2].

3.1.7 Munigenkänning & fönstermask

Med samtliga maskering ur vägen är det rimligt att titta på en metod som kan rama in ögonen ytterligare i den maskerade bilden. Detta för att försöka filtrera bort hår och annat bakgrundsbrus som skulle kunna störa ögonigenkänning. En till synes enkel metod skulle kunna vara att hitta munnen i den maskerade bilden och utifrån den göra ett fönster som fångar in ögonen och filtrerar bort en stor del av den maskerade bilden. Det som gör metoden aktuell att tillämpa bygger på att munnen upptar ett mycket större område av den maskerade bilden och är därmed enklare att fånga upp än två par ögon.

Med motivering till hur en enkel ögonmask skulle kunna skapas är det dags att studera hur den faktiskt tas fram. Till en början behövs den maskerade bilden att utsättas för en rad operationer som frambringar munnen ur bilden. Det första steget är att konvertera bilden från RGB till YC_bC_r som är den optimala färgdimensionen att arbeta i för munigenkännig [4]. Denna konvertering ger möjligheten att isolerar och behandlar ljusintensiteten (Y) separat från färginformationen (C_b och C_r). Utifrån detta tillämpas Ekvation 3.11 där η beräknas som i Ekvation 3.12. Resultatet efter att ha tillämpat föregående ekvationer på Figur 2.1 går att se i Figur 3.7 i from av en munkartan.

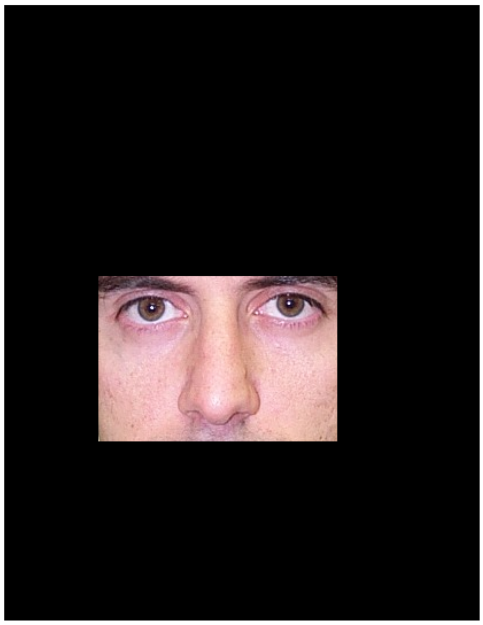
$$I_{MouthMap} = C_r \cdot (C_r^2 - \eta \cdot \frac{C_r}{C_b})^2 \tag{3.11}$$

$$\eta = 0,95\frac{\frac{1}{n} \cdot \sum_{(x,y)\in FG} C_r(x,y)^2}{\frac{1}{n} \cdot \sum_{(x,y)\in FG} \frac{C_r(x,y)}{C_b(x,y)}} \tag{3.12}$$

Figur 3.7: Den resulterande bilden vid tillämpning av munigenkänning av Figur 2.1.

Med tidigare operationer gjorda på den maskeradebilden är det bara helt enkelt att, med ett tröskelvärde, ta bort alla element förutom munnen. Tröskelvärdet bör i det här fallet vara en hög procent av det största värdet i munkartan. Därefter fås en trösklad bild vars vita element i bilden representerar munens placering. Den exakta positionen för munnen i bilden kan enkelt fås i MATLAB genom att, exempelvis, sätta ut en centroid i det vita området.

Med den exakta vetskapen om munnens position i bilden kan en fyrkantig mask utformas och appliceras på den markerade bilden. Masken bör börja något ovanför munnen och sluta godtyckligt så ögonen fångas in i masken. För att inte upplösningen på en bild ska påverka maskens storlek bör bredden och höjden formuleras utifrån den nuvarande bildens höjd och bredd. Därmed bör ett antal undersökningar genomföras som fastställer att masken fungerar för en godtycklig bild. Resultatet efter att ha format och tillämpat ögonmasken på en bild går att se i Figur 3.8 där orginalbilden är Figur 2.1.



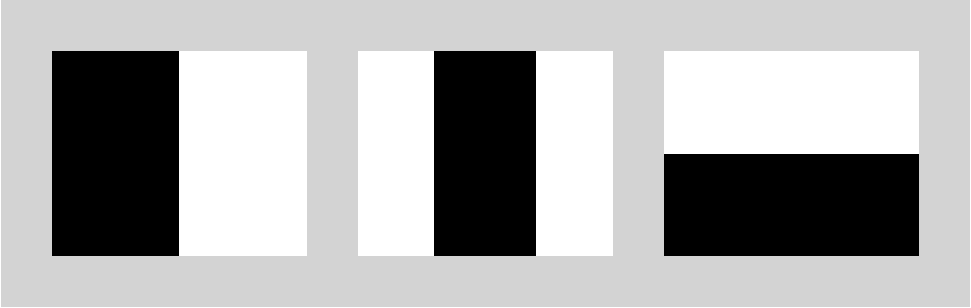
Figur 3.8: Resultatet av att skapa och tillämpa ögonmasken utifrån munnen på en person som i det här fallet är Figur 2.1.

3.1.8 Viola-Jones algoritmen

Den tidigare nämnda metoden är en väldigt enkel metod för att rama in ögonen ytterligare utifrån den maskerade bilden. Metoden förutsätter att det inte finns något annat i bilden som skulle kunna likna en mun vilken i sin tur kan leda till att ögon saknas helt i den slutgiltiga maskeringen. Detta är inget problem vid bilder där ansiktet tar upp största delen av bilden och det inte finns så mycket bakgrund. Tas dock dessa egenskaper bort kommer problemen tillbaka och det kan bli svårt att finna ögonens position. Det kan därför vara aktuellt att tillämpa en något mer avancerad modell för att skapa ögonmasken utifrån den maskerade bilden. En sådan modell är Viola-Jones algoritmen som är en maskininlärningsmodell optimerad för att känna igen egenskaper ur ett ansikte.

Själva algoritmen bygger på följande fem steg: Haar-like egenskaper, integral bild, adaboost, Attentional cascade och data-experiment [5]. Det första steget, Haar-like egenskaper, innebär att flera skalärprodukten utförs mellan bilden och olika mönster (Haar-like egenskaper). Ett mönster är i det här fallet ett rektangulärt fönster bestående av ett antal vita och svarta rektanglar i sig, likt i Figur 3.9. En egenskap i ansiktet vår sedan ett värde genom att ta skillnaden mellan summan av pixlarnas ljusintensitet i det vita och svarta områdena av fönstret. Olika egenskaper i ansiktet ger olika världen på summan och kan därmed användas för att detektera ansiktsegenskaper.

Det andra steget, integral bilden, fås sedan genom att skapa en bild, av samma storlek som originalbilden, där samtliga pixelvärden representerar summan för just den pixeln utifrån tidigare nämnda steg. Däremot existerar en bild vars värden kan användas för att



Figur 3.9: En visuell representation av tre olika Haar-like egenskaper som kan tillämpas i Viola-Jones algoritmen

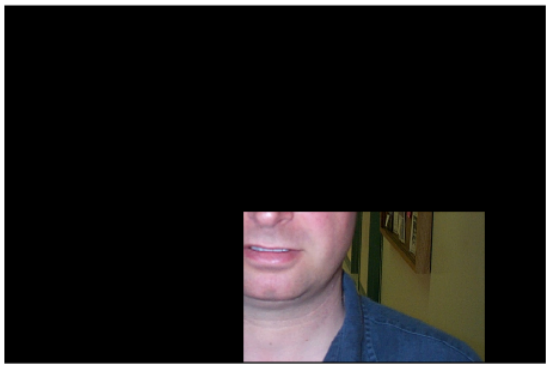
definiera olika ansiktsegenskaper.

Steg tre, adaboost, innebär att de värden som faktiskt anses viktiga för att hitta egenskaper blir viktade för att på så sätt framträda mer utöver de övriga värdena. Detta är en något avancerad process som sammanfattningsvis viktar en summa beroende på dess prestanda under själva maskininlärningprocessen. Detta görs genom att iterativt kombinera flera svaga egenskaper till en stark [6]. I det här fallet representerar en svag egenskap en enda Haar-like egenskap. Utifrån träningen kan därmed algoritmen välja vilka egenskaper som är viktigast för en korrekt igenkänning.

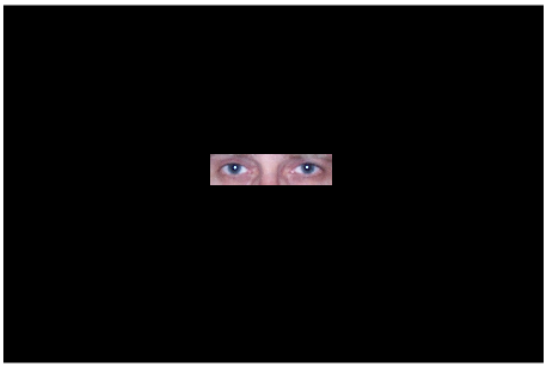
Sist kommer steg fyra som egentligen är ett inbakat moment i steg tre. Detta steg görs för att optimera algoritmen genom att effektivt sortera bort regioner som inte är en del av ett ansikte. Denna process innebär att stora delar av en bild inte behöver undersökas samt att databasen vid träning kan vara avsevärt mycket mindre än utan tillämpning av detta steg [5]. Själva processen består av ett antal matematiska filter formulerade för att filtrera bort onödiga områden ur bilden. Skulle ett område gå igenom alla filter antas området vara en viktig faktor till ansiktsigenkänningen.

Själva algoritmen tillämpas i MATLAB genom att skapa ett detektionsobjektet (*vision.CascadeObjectDetector*) specialiserad för inramning av ögon. Sedan matas den maskerade bilden in i detektionsobjektet som i sin tur returnerar en x och y koordinaten för det vänstra överst hörnet i följd av bredden och höjden på fönstret.

Notera att Viola-Jones algoritmen inte är så pass perfekt att den kan hitta ögon helt på egen hand. Vid bilder med dålig färgbalans och plottrig bakgrund kan algoritmen lätt hitta något i bakgrunden som efterliknar två bar ögon. Algoritmen bör därmed kombineras med den tidigare gjorda ansiktsmasken på liknande sätt som den första varianten av ögonmask. Ett visuellt exempel på när algoritmen ser fel utan maskering och färgkorrigering men rätt då metoderna tillämpas går att se i Figur 3.10.



(a) Resultat då färgkorigering och ansiktsmaskering ej tillämpas på en bild ur DB2.



(b) Resultat då färgkorigering och ansiktsmaskering tillämpas på en bild ur DB2.

Figur 3.10: En visuell representation över hur en bild ur DB2 inte får en korrekt ögonmask om den inte först genomgår färgkorigering och ansiktsmaskering.

3.1.9 Ögonmaskering

För att kunna separera ögonen ur den maskerade bilden som nu kombineras med en ögonmask (fönster mask eller Viola-Jones) krävs likande operationens som vid munigenkännigen. Därmed ändras den maskerade bilden från RGB rymd till YC_bC_r rymd. Därefter kan Ekvation 3.13 tillämpas för att få den så kallade C-delen av den slutgiltiga ögonkartan [4]. Sedan genomförs Ekvation 3.14 för att få L delen. Notera att $\oplus$ innebär en utvidgning och $\ominus$ en erosion med strukturelementet g_σ(x,y).

$$eyeMapC = \frac{1}{3}(C_b^2 + (1 - C_r)^2 + \frac{C_b}{C_r}) \tag{3.13}$$

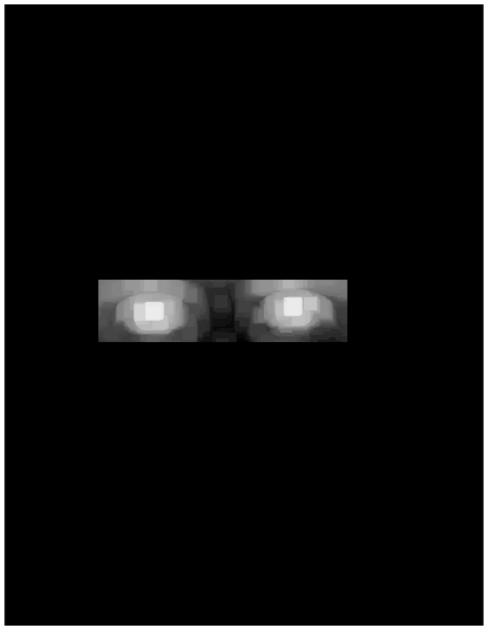
$$eyeMapL = \frac{Y(x,y) \oplus g_\sigma(x,y)}{(Y(x,y) \ominus g_\sigma(x,y)) + 1} \tag{3.14}$$

Med både L- och C-kartan uträknade för den maskerade bilden är det att följa Ekvation 3.15 för att få den fullständiga ögonkartan.

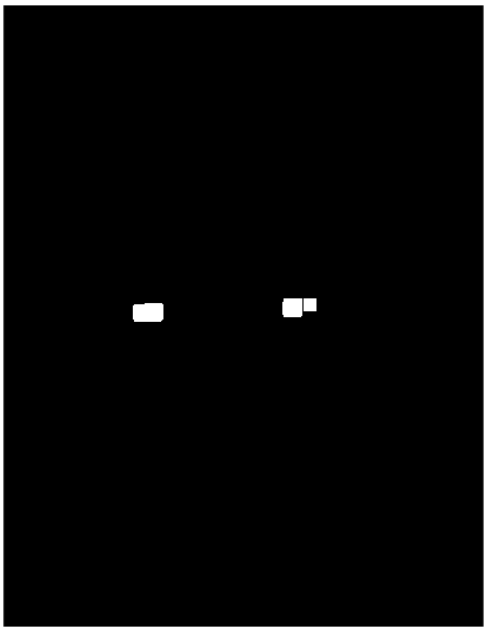
$$eyeMap = eyeMapL \textbf{ AND } eyeMapC \tag{3.15}$$

En visuell reprastion på den slutgiltiga ögonkartan för Figur 2.1 (ej en maskerad bild) går att se i Figur 3.11. För att sedan få en mask där enbart ögon kvarstår är det, likt som i munigenkännigen, att tröskla bort alla världen förutom det ljusaste som nu representeras av ögonen. En visuel representation efter tröskling på en tidigare maskerad bild går att se i Figur 3.12.

Figur 3.11: Den resulterande bilden vid tillämpning av ögon kartläggning av Figur 2.1.



(a) Ögonkartan efter att ViolaJones samt den slutgiltiga masken tillämpas på Figur 2.1.



(b) Den trösklade versionen utifrån Figur 3.12a.

Figur 3.12: Tillämpning av ögonkartan efter att ViolaJones och den slutgiltiga masken tillämpas på Figur 2.1 samt en tröskling på det.

3.1.10 Cirkulär Hough transform

Cirkulär Hough transform är en avancerad bildbehandlingsalgorithm som används för att upptäcka cirklar i en bild genom att omvandla pixelpunkter i bilden till en parametriserad cirkelform [7]. Algoritmen är något komplicerat men bygger sammanfattningsvis på ekvationen för en cirkel i ett koordinatsystem, se Ekvation 3.16 och inleds med en kantdetektion för cirklar. När en kant hittas överförs den till parameterform för att sedan undersökas med hjälp av Hough transformen (där toppar kommer tyda på cirklar i bilden).

$$(x-a)^2 + (y-b)^2 = r^2 \tag{3.16}$$

Omvanlingen till Hough-rummet kan ses i Ekvation 3.17, där *c1*, *c2* och *c3* representerar parametrar för cirkelns centrum och radie. Algoritmen löser 3.17 för varje punkt i bilden. För varje löst punkt ökar räknaren för de motsvarande parametrarna *c1*, *c2* och *c3* i Hough rummet.

$$(x-c_1)^2 + (y-c_2)^2 - c_3^2 = 0 \tag{3.17}$$

I MATLAB kan en cirkulär Hough transform som finner cirkulära objekt göras med *[koordinater i bilden, radie] = imfindcircles(I,maximala radie)*. Denna funktion kan därmed tillämpas i ett område runt de nuvarande utsatta centroiden för vardera öga för att på så sätt ge ännu bättre prestation för ögats mitt. Notera dock att denna metod inte är felfri och kommer med all säkerhet inte fungera på samtliga bilder utan snarare enbart ett fåtal.

3.2 Ansiktsnormalisering

Ansiktsnormaliseringen utgör en stor del av vår ansiktsigenkänningsalgoritm och spelar en avgörande roll för att kunna utföra PCA med ett lyckat resultat. Denna del innehåller flera steg, inklusive translation, rotation, skalning och beskärning av ansiktsbilderna för att standardisera och förbereda bilderna för sektion 3.3. För ansiktsnormaliseringen skapades en *.m-fil* som returnerade en normaliserad bild. Till hjälp användes koordinaterna för ögonen som invariabler, diskutrat i sektion 3.1. Translation, rotation, skalning och beskärning applicerades i den ordningen för att normalisera och standardisera ansiktsbilderna. Notera att ordningen för operationerna är viktig eftersom varje steg bygger på det föregående steget.

3.2.1 Translation

Först applicerades en translationsoperation på bilderna för att centrera dem inom bildramen. För detta användes ögonens mittcentrum och bilderns nuvarande mittcentrum för att beräkna en translationsvektor. Denna translations vektor användes sedan i funktionen

imtranslate(bild, translateVec). Efter translationen uppdaterades även ögonens positioner enligt translationsvektorn.

3.2.2 Rotation

Efter translationen roterades bilderna utifrån ögonens positioner. Tanken här är att få ögonen raka emot varandra utifrån ett horisontellt plan. Bilden roterades med hjälp utav funktionen *imrotate(bild, vinkel, metod för interpolation, logisk invariabel)*. Eftersom ögonens positioner är kända kan vinkeln enkelt räknas ut. Vidare så användes interpolations metoden *bicubic* för att hantera de tomma områdena som bildas när en bild roteras. *Bicubic* interpolation är en mer avancerad interpolationsmetod jämfört med andra enklare metoder som *bilinear interpolation*. För *logisk invariabel* användes *crop*, vilket innebär att att rotationen inte kommer beskära bilden vid rotation.

3.2.3 Skalning

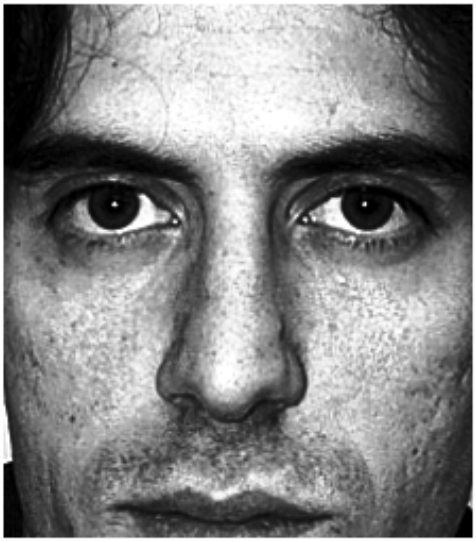
Efter rotationen skalades bilden utifrån ögonens centrum och ett önskat avstånd. Skalfaktorn beräknas som kvoten mellan det önskade avståndet och det faktiska avståndet. För denna operation användes *imresize(bild, skalFaktor)* som skalar om en *bild* enligt en *skalfaktor*. Efter denna operation skalades även ögonens position med samma skalfaktor för vidare operationer.

3.2.4 Beskärning

Efter skalningen så beskärdes bilden så att endast ansiktet var kvar. För detta användes ögonens centrumpunkt som mittpunkten för en beskärningsruta. För beskäringen användes fasta värden för en önskad storlek på beskärningsrutan. Beskärningsrutan sparades i *cropX = (x1, x2)* och *cropY = (y1, y2)* variabler som innehöll maximala och minimala koordinaterna för rutan. För att säkerställa att dessa koordinater inte överskred bildgränserna användes *min* och *max* i MATLAB. Slutligen så beskärdes bilden enligt följande: *beskärdBild = skaladBild(cropY(1):cropY(2), cropX(1):cropX(2), :);*.

3.2.5 Korrigering

Slutligen så kontrastrjusterades och färgkorrigerades bilden för extra förbättring. För detta användes dels MATLAB-funktionen *colorCorrection* för justering av färgbalansen, *AWB* för att anpassa färgtemperaturen i en bild så att den ser mer naturlig ut och inte lider av överdrivna färgtoner och *contrastStretchColor* som är en teknik som används för att öka kontrasten i en bild genom att sprida pixlarnas intensiteter över hela det möjliga intensitetsområdet. Detta hjälper till att göra skillnaderna mellan ljusa och mörka områden mer tydliga. En illustration av en normaliserad bild går att se i Figur 3.13.



Figur 3.13: En illustration av det normaliserade resultatet utifrån Figur 2.1.

3.3 Huvudkomponentanalys, PCA

PCA är en teknik inom dels statistik och maskininlärning som bland annat används för dimenstionsreduktion för att endast använda de viktigaste egenskaperna från en datamängd. För detta arbete används PCA för att minska den höga dimensionaliteten hos ansiktsbilderna för att skapa en kompakt representation av ansiktsrummet. Denna sektion beskriver hur PCA applicerades på detta arbete.

3.3.1 Dimensionsreduktion

Först utfördes dimenstionsreduktion genom PCA. För detta skapades en matris X som innehöll en kolumn för varje ansiktsbild. Varje ansiktsbild har då omvandlats till en vektor genom att platta ut bildens pixelvärden. För att standardisera denna data subtraherades sedan medelvärdet av alla ansiktsbilder på matrisen, vilket sparades i matrisen A. Därefter beräknades kovariansmatrisen $V = A^t A$ genom att multiplicera transponaten av den standardiserade datan med sig själv.

3.3.2 Egen-ansikten

Egen-ansikten representerar en kraftfull tillämpning av PCA inom ansiktsbildsbehandling. Egen-ansikten utgörs av egenvektorer från en träningsuppsättning av ansiktsbilder och samlas för att fånga upp variationen i de ursprungliga ansiktena. Med hjälp av detta kan varje ansikte i träningsuppsättningen representeras som en viktad linjär kombination av de grundläggande egen-ansiktena.

För att beräkna egen-ansikten beräknades egenvektorerna U med hjälp av kovariansmatrisen multiplicera matrisen V med matrisen A. Med hjälp av denna operation kan de mest signifikanta riktningarna eller egenskaperna hos ansiktsdatan identifieras.

Det hela börjar med att bilden transformeras om från en matris till en enda vektor x_i [8]. Sedan tas ett genomsnittligt ansikte ut utifrån samtliga vektorer som kallas för medleansiktet μ. Medelansiktet tas sedan bort från x_i som i Ekvation 3.18. Storleken för x_i blir därmed antalet kolumner multiplicerat med antalet rader för en bild som i det här samanhanget kallas n.

$$\phi_i = x_i - \mu \tag{3.18}$$

Nästa steg är sedan att beräkna kovariansmatrisen enligt Ekvation 3.19 där $A = [\phi_1, \phi_2, \ldots \phi_M]$. Här uppstår en viss problematik då C blir en nxn stor matris vilket i praktiken är helt omöjligt att jobba med.

$$C = AA^T \tag{3.19}$$

En lösning till detta dilemma går att se i Ekvation 3.20 som är av storleken MxM och returnerar därmed enbart M egenvektorer av storleken Mx1 vilket är avsevärt mycket enklare att hantera [9].

$$V = A^T A \tag{3.20}$$

Utifrån V kan sedan U beräknas som representerar de M största egenvektorerna från V för matrisen AA^T. Detta görs enligt Ekvation 3.21 och öppnar därmed upp för klassificering för en bild.

$$U = AV \tag{3.21}$$

För att få vikter till distansberäkningarna i PCA modelen används projektionen Ekvation 3.22 där Ω representerar en vektor med vikter för samtliga bilder i det kända datasetet. μ är medelansiktet och x_i är varje bild i vektorform som tidigare redovisat.

$$\Omega_i = U^T(x_i - \mu) \tag{3.22}$$

$$W_{pca} = [\Omega_1, \Omega_2, \ldots \Omega_M] \tag{3.23}$$

För att klassificera en bild $x_{unknown}$ används sedan Ekvation 3.22 på samma sätt som innan där x_i är $x_{unknown}$. Det euklidiska normaliserade avståndet mellan det okända Ω och de kända representerar likheten i bilderna. Den bild i databasen som ger den minsta distans är därmed programmets slutsats om vem individen är. Det är dock mycket viktigt att ha med ett tröskelvärde på avståndet för att inte klassificera fel ansikte. Efter ett tillräckligt stort avstånd för det minsta avståndet returneras därmed ID-numret 0 då inget ansikte i databasen är tillräckligt likt det okända.

3.4 Fisher

Fisherfaces-algoritmen är en metod som används för ansiktsigenkänning och kombinerar styrkorna hos Principal Component Analysis (PCA) och Fisher's Linear Discriminant (FLD) [10]. Dess syfte är att extrahera diskriminerande egenskaper från ansiktsbilder för att underlätta klassificering.

3.4.1 Datatillredning

Med en datamängd av ansiktsbilder representerade som $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]$, där varje $\mathbf{x}_i$ är en vektoriserad bild, börjar algoritmen genom att organisera dessa bilder i klasser betecknade som $C_1, C_2, \ldots, C_K$, där K representerar det totala antalet klasser. Varje klass C_i innehåller n_i bilder.

3.4.2 Beräkning av Medelansikte

För att kunna genomföra operationerna nödvändiga för klassificering av en bild behövs, som nämnt i sektion 3.3.2, ett medelansikte beräknas. Det genomsnittliga ansiktet $\bar{\mathbf{x}}$ för hela datamängden beräknas som:

$$\bar{\mathbf{x}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i \tag{3.24}$$

Nu räcker det dock inte enbart att beräkna ett medelansikte för samtliga bilder. Ett medel måste tas fram för varje person då det nu finns flera bilder för en person. För varje klass C_i beräknas därmed det genomsnittliga ansiktet $\bar{\mathbf{x}}_i$ som:

$$\bar{\mathbf{x}}_i = \frac{1}{n_i} \sum_{\mathbf{x}_j \in C_i} \mathbf{x}_j \tag{3.25}$$

Resultatet som fås av att beräkna medelansiktet utifrån DB2 och DB1 går att se i Figur 3.14 (medelansiktet av samtliga personer). Ett liknande ansikte går även att få om medelansiktet för en klass skulle illustreras.

Figur 3.14: Medelansiktet från DB1 och DB2.

3.4.3 Projektion på Egenansikten

Inom datorseende och särskilt inom ansiktsigenkänning har metoder för att minska dimensioner och representera data på ett effektivt sätt varit av betydande intresse [10]. Eftersom att korrelation tekniker både är kräver stora mängder processor kraft och lagrings utrymme. Används PCA (se, 3.3) för att reducera datakomplexiteten utan att förlora väsentlig information. Projektionen på egenansikten är därför en nyckelkomponent i att extrahera de viktigaste aspekterna av ansiktsvariationen som möjliggör effektiv ansiktsigenkänning.

3.4.4 FLD och *Fisher*-ansikten

I det här arbetet används en algoritmen för *Fisher*-ansikten fram tagen i [10]. Denna sektion beskriver hur tidigare teori för PCA samt egenansikter tillämpas för att kunna klassificera ett ansikte. Med samtliga steg i sektionerna ovan beräknas Mellanklassscattermatrisen $\mathbf{S}_b$ och Inomklassscattermatrisen $\mathbf{S}_w$ för FLD modellen.

Mellanklassscattermatrisen $\mathbf{S}_b$ ges av ekvationen nedan. Notera att medelansiktet samt medelansiktet för varje klass beräknas som i sektion 3.4.2.

$$\mathbf{S}_b = \sum_{i=1}^{K} n_i(\bar{\mathbf{x}}_i - \bar{\mathbf{x}})(\bar{\mathbf{x}}_i - \bar{\mathbf{x}})^T \tag{3.26}$$

Inomklassscattermatrisen $\mathbf{S}_w$ beräknas som summan av scattermatriserna för varje klass:

$$\mathbf{S}_w = \sum_{i=1}^{K} \sum_{\mathbf{x}_j \in C_i} (\mathbf{x}_j - \bar{\mathbf{x}}_i)(\mathbf{x}_j - \bar{\mathbf{x}}_i)^T \tag{3.27}$$

S_b och S_w används för att beräkna W_{fld} som ges av:

$$\mathbf{W}_{fld} = \arg\max_{\mathbf{W}} \left| \frac{\mathbf{W}^T \mathbf{W}_{pca}^T \mathbf{S}_B \mathbf{W}_{pca} \mathbf{W}}{\mathbf{W}^T \mathbf{W}_{pca}^T \mathbf{S}_W \mathbf{W}_{pca} \mathbf{W}} \right| \tag{3.28}$$

Där $\mathbf{W}_{pca}$ kommer från Ekvation 3.23. Vidare används $\mathbf{W}_{pca}$ och $\mathbf{W}_{fld}$ för att ta fram $\mathbf{W}_{opt}$ som maximerar förhållandet mellan determinanten för den scattermatris som representerar skillnader mellan klasser för de projicerade ansiktena. Detta är det som kallas *Fisher*-ansikten.

$$\mathbf{W}_{opt}^T = \mathbf{W}_{fld}^T \mathbf{W}_{pca}^T \tag{3.29}$$

Algoritmen projiserar sedan tränings bilderna $\mathbf{X}$ på $\mathbf{W}_{opt}$ för att skapa igenkännings modelen.

$$\mathbf{traindModel} = \mathbf{W}_{opt}^T \mathbf{X}_{unknown} \tag{3.30}$$

Modellen används sedan för att klassificera nya bilder. På samma sätt projiceras den oklassificerade bilden på $\mathbf{W}_{opt}$. Därefter beräknas det kvadratiska euklidiska avståndet mellan den projicerade bilden och klasserna i den förtränade modellen. Om avståndet inte överstiger en förutbestämd gräns klassas bilden som tillhörande den klass med det minsta avståndet.

Kapitel 4

Resultat

Resultatet av implementationen baserat på den teori framlagt i Kapitel (Viola-Jones används och inte mun metoden) 3 går att se i Tabell 4.1. Kolumnen med rubrik DB står för typ av databas, count för antalet bilder, ACC för totala träffsäkert, FR antalet icke accepterade bilder som bör accepteras, FRR den procentuella siffran på icke accepterade, FA antal felaktig accepterade och FAR procenten på antalet felaktig accepterade. De bilder som använts är från DB0, DB1 och DB2. Bilder från DB1 och DB2 har även körts med pålagda transformer (T) för att testa programmet i ytterligare svåra förhållanden. En transform av en bild betyder i det här fallet en slumpmässig förändring i rotation (mellan $\pm 5°$), translation, skalning (mellan $\pm 10\%$) samt förändring av tonvärden i bilden (mellan $\pm 30\%$). Utöver detta visar även Figur 4.1 en visuell representation vart centroiderna för ögonen hamnar på en modifierad bild från DB2.

Tabell 4.1: Resultatet efter att ha aplicerat programmet på DB0, DB1 och DB2 samt några variationer av DB1 och DB2.

DB	Count	ACC	FR	FRR	FA	FAR
DB0	4	100.00%	0	0.00%	0	0.00%
DB1	16	100.00%	0	0.00%	0	0.00%
DB1T	80	91.25%	6	7.50%	1	1.25%
DB2	38	100.00%	0	0.00%	0	0.00%
DB2T	190	66.84%	41	21.58%	22	11.58%

Figur 4.1: En Figur som illustrerar detektion av två ögon utifrån en bild från DB2 på personen med ID-nummer 14. Notera att bilden är manipulerad med skalning, translation, rotation samt tonvärden.

Kapitel 5

Diskussion

Utifrån resultatet från Kapitel 4 finns det en rad olika aspekter intressanta att diskutera. Det första området är frågan om huruvida en hög träffsäkerhet eller säkerhet i sitt svar av ID är att föredra. Många av de fall då en person som egentligen inte ska nekas blir nekad hade programmets avståndsberäkning rätt person som närmast. Anledningen till att personens ID inte returneras blir då att tröskeln för avstånd i likhet är för liten. Skulle dock denna tröskel ökas blir risken större att systemet läser en person felaktigt och även inte returnerar 0 för personer som ska returnera 0. Svaret på frågan bygger lite på vad systemet faktiskt ska tillämpas till. Om systemet skyddar något viktigt är det av högsta vikt att absolut inte släppa in fel människor och därmed används en låg tröskel (lägren än den som tillämpas nu om bilder som DB2T ska tillämpas). Detta gör då att personer som inte finns i databasen inte tar sig igenom systemet till priset av att en person som ska kunna komma in behöver ta om sin bild ett antal gånger (i vissa fall). Det går dock att argumentera för att om denna typ av säkerhet ska upprätthållas bör det finnas krav på att en bild för identifiering måste vara så "ren" som möjligt och därmed mer lik DB1 än DB1T, DB2 eller DB2T.

Ett annat område värt att diskutera är hur programmets igenkänning skulle kunna optimeras. Som det går att se nu får programmet svårt att identifiera vissa ansikten som manipuleras. Dessa problem är sannolikt ett resultat av att varje klass består av för få bilder med olika skillnader i sig. Med en större testdatabas skulle varje klass bli mer väldefinierad och på så sätt marginellt öka sannolikheten för en korrekt igenkänning. Det optimala hade även varit att ha en met precis metod att sätta ut centroiderna för ögonen då en liten skillnad kan potentiellt ge en stor skillnad i det normaliserade ansiktet.

Tillämpningen av Viola-Jones algoritm är också ett område intresant att studera. Denna algoritm är, som nämnt i Kapitel 3, inte tillräckligt stark att på egen hand hitta ett ansikte från en oredigerad bild. Metoden är dock mycket kraftfull på rätt typer av bilder och är en stor del i att finna ögon inför normaliseringen.Tillämpningen efterliknar även den enklare metoden som tillämpar en mask utifrån munnen. Vid de mer avancerade bilderna från DB2 blir det dock mycket svårt att alltid hitta munnen och därmed blir Viola-Jones det bästa alternativet. De båda sätten kräver också olika typer av ansiktsmasker då den enklare enbart behöver ha en mask som garanterar att munnen finns i bild medan Viola-Jones kräver ett helt ansikte. Därmed behöver hudfärgsmasken optimeras utifrån vilken

av dessa metoder som tillämpas för att ge rätt förutsättningar.

Metoden för ansiktsnormalisering och ögonigenkänning, som presenteras i Kapitel 3, utgör en omfattande och noggrant utformad process för att standardisera och förbereda ansiktsbilder för den efterföljande analysen PCA. Utan dessa steg hade inte PCA fungerat väl eftersom PCA kräver normalisation över sin data för lyckat resultat. Det hade därför varit intressant att i en framtida utveckling lägga ytterligare fokus på just normaliseringen av ansiktet för att potentiellt ge en bättre precision i det slutgiltiga resultatet.

När det kommer till ansiktsigenkänning blir metoder ofta utmanade när de tillämpas på mer komplicerade bilder som i DB2. Några faktorer som framförallt kan skapa problem är perspektiv, störningar i bakgrundsinformation, förändrade ljusförhållanden med mera. Trots dessa faktorer så lyckas dock programmet utföra en korrekt ögondetektion. För fortsatt arbete skulle andra faktorer kunna testas såsom otydliga ansiktsvinklar, andra perspektiv, variation i ålder eller smink för att på så sätt visa förbättringsområden i optimering.

Sist är det även av intresse att diskutera hur användandet av maskininlärning eller, mer exakt, deep learning skulle kunna vara en potentiell ersättare till stora delar av implementationen som gjorts i detta projekt. I nuvarande läge (sena 2023) är avancerade AI algoritmer baserade på deep learning extremt aktuellt och förväntas kunna bli en stor del av mänsklighetens framtid integrerat i allt möjligt. Det finns redan nu algoritmer som tillämpas för att genomföra ansiktsigenkänning som klart är mer effektiva än att tillämpa enbart bildigenkänning. Ett exempel på en metod som potentiellt skulle kunna ersätta den implementerade metoden är exempelvis *convolutional neural networks (CNN)* (avancerade faltningsfilter för att analysera bilder) i kombination med *Multi-Layer Perceptron (MLP)* algoritm för att analysera och beräkna likheter i en bild mot en databas. Utöver möjligheten att tillämpa CNN kombinerat med MLP lär det finnas en rad olika neuralt nätverk som efter optimering alla slår den implementerade metoden för ansiktsigenkänning. Dessa metoder är däremot mycket mer komplicerade än den implementerade metoden och kan kräva större databaser för att vara optimala samt fortfarande kräva någon form av bildbehandling. Ett problem som dessa nya metoder skulle kunna lösa är att Fisherfaces-metoden kräver stora variationer i kluster för att vara effektiva och får utifrån detta, som vi ser i resultatet, lätt problem när olika kluster börjar likna varandra. Neurala nätverk baserade på deep learning är därmed troligen vad som kommer användas i framtiden då teknik måste bli mer avancerad för att slå teknik från det förflutna. Med andra ord är den implementerade metoden i detta projekt troligen bestående av vissa delar del av det förflutna och neurala nätverk lär sannolikt ta över rollen för ansiktsigenkänning i kombination med bildbehandling.

Litteraturförteckning

[1] D. Nyström. Face detection in color images, November 2023.

[2] Muhammad Shafi and Paul Chung. A hybrid method for eyes detection in facial images. 2009.

[3] Jose M Chaves-González, Miguel A Vega-Rodríguez, Juan A Gómez-Pulido, and Juan M Sánchez-Pérez. Detecting skin in face recognition systems: A colour spaces study. *Digital signal processing*, 20(3):806–823, 2010.

[4] Rein-Lien Hsu, Mohamed Abdel-Mottaleb, and Anil K Jain. Face detection in color images. *IEEE transactions on pattern analysis and machine intelligence*, 24(5):696–706, 2002.

[5] Yi-Qing Wang. An analysis of the viola-jones face detection algorithm. *Image Processing On Line*, 4:128–148, 2014.

[6] Robert E Schapire. Explaining adaboost. In *Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik*, pages 37–52. Springer, 2013.

[7] Simon Just Kjeldgaard Pedersen. Circular hough transform. *Aalborg University, Vision, Graphics, and Interactive Systems*, 123(6):2–3, 2007.

[8] Matthew A Turk and Alex P Pentland. Face recognition using eigenfaces. In *Proceedings. 1991 IEEE computer society conference on computer vision and pattern recognition*, pages 586–587. IEEE Computer Society, 1991.

[9] D. Nyström. Dimensionality reduction for facial features, November 2023.

[10] Peter N. Belhumeur, Joao P Hespanha, and David J. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on pattern analysis and machine intelligence*, 19(7):711–720, 1997.