EXPRESSIEHERKENNING

ALBERT GERRITSEN (0318310)

Inhoudsopgave

1. Introductie

Interpersoonlijke communicatie bestaat uit meer dan alleen woorden. Tijdens het communiceren gebruiken we lichaamstaal, stemverheffing, de context van het gesprek en onze bestaande kennis om de betekenis van het gesprokene vast te stellen. Soms vullen deze kanalen elkaar aan en soms zijn ze redundant en stellen ze ons in staat fouten op te vangen.

Deze vorm van communicatie is robuust en het communiceren via meerdere kanalen doen we zonder veel moeite. Daarom staat de communicatie tussen mensen model voor nieuwe vormen van mens machine interactie (HCI) [JS05]. Het ideaal is dat interactie met computers net zo natuurlijk gaat als interactie tussen mensen. Hiervoor moet tijdens deze interactie gebruik gemaakt worden van de vele kanalen die beschikbaar zijn.

Een belangrijk kanaal is de gezichtsuitdrukking van onze gesprekspartner. In een willekeurig blikveld herkennen we het relevante gezicht en stellen we de emotionele toestand of reactie van onze gesprekspartner vast, en we doen dit alles zonder veel moeite. Wat voor mensen gebeurt in een oogopslag, blijkt opvallend moeilijk in een algoritme te vangen te zijn.

Deze kwestie houdt dan ook vele onderzoekers bezig die het probleem op een verschillende manier benaderen. Sommige onderzoekers proberen het herkennen zoals dat in het menselijk brein plaatsvind nauwkeurig te beschrijven [HDR93, HHG00]. Andere onderzoekers werken aan een herkenningsprocedure die niet persé gelijkenis vertoont met de processen in het menselijk brein. Wij zullen ons richten op deze tak van onderzoek. Een review van Maja Pantic geeft een goed overzicht van het veld en deelt het herkenningsproces op in drie onderdelen [PR00]:

- *Lokalisatie* van het gezicht
- Verandering van *representatie*
  (bijvoorbeeld: FACS, eigen-vectoren, wavelets en facial features)
- *Classificatie*
  (bijvoorbeeld: SVM, Bayesiaanse netwerken en neurale netwerken)

Voor deze scriptie gaan we er vanuit dat het gezicht al gelocaliseerd is, hiervoor zullen we een bestaand programma gebruiken. Voor de keuze voor een representatie en een classificatie methode hebben we ons laten inspireren door [BLL+04]. Als representatie kiezen we voor een bank van Gabor filters. Deze representatie maakt het gemakkelijker specifieke lijnen in een plaatje te herkennen. Voor de classificatie wordt gebruik gemaakt van het adaboost algoritme. Dit algoritme kiest op iedere ronde een nieuwe feature om de fout van de vorige ronde te compenseren en combineert deze features tot een classifier. Dus mijn onderzoeksvraag luidt[1]:

> *Kan d.m.v. Gabor filters met adaboosting de expressie van een gezicht herkend worden?*

In het volgende hoofdstuk werken we de onderzoeksvraag verder uit. Daarna, in hoofdstuk 3, wordt wat achtergrond informatie gegeven over de gebruikte technieken. In hoofdstuk 4 wordt de opzet van het experiment uiteengezet. Daarna worden de resultaten behandeld en geanalyseerd in hoofdstuk 5. Ten slotte trekken we onze conclusies in het laatste hoofdstuk.

[1] Oorspronkelijk was deze scriptie een gezamenlijk werk met Ruben Muijrers, die hetzelfde doel voor ogen had maar met een andere representatie en classificatie procedure (resp. PCA en SVM). De oorspronkelijke onderzoeksvraag vergeleek de verschillende technieken met elkaar. Uiteindelijk is toch voor gekozen om twee losse scripties te maken. De voorbewerkte dataset is hetzelfde en de experimentele opzet is vergelijkbaar met [Mui09]. Verder zijn secties 2.2, 4.1 overgenomen uit [Mui09]. Aan het einde van deze scriptie zal ik alsnog de resultaten met elkaar te vergelijken.

2. Probleemdefinitie

De onderzoeksvraag zoals geformuleerd in de introductie is onvermijdelijk een beetje vaag. Bij het lezen ervan rijzen vragen als: *wanneer is de herkennings procedure goed genoeg?*, *welke emoties wil je kunnen onderscheiden?*, *bestaat er zoiets als een universele verzameling van emoties?* en *welke aannamen doe je over de data?*. Dit hoofdstuk zal de onderzoeksvraag nader uitwerken door deze vragen te beantwoorden.

Gezichtsexpressies hebben betekenis en spelen een belangrijke rol in de communicatie tussen mensen. Vanuit de psychologie zijn gezichtsexpressies ook al vele jaren onderwerp van studie. Alleen is de vraag, vanuit psychologisch perspectief, iets ingewikkelder. Psychologie beperkt zich niet tot het simpelweg herkennen van expressies maar beschouwt ook de mogelijke invloeden, gevolgen en oorzaken van expressies.

De vraag of expressies (en de bijbehorende emoties) universeel zijn is een ingewikkelde. Veel onderzoek is gedaan met verschillende resultaten. Het artikel van James A. Russel [Rus94] geeft een grondige review van voorgaand onderzoek en plaatst het in context door verschillende uitgangspunten aan te stippen in de onderzoeken. Russel geeft de volgende vier hypothesen:

(1) *Universality of facial movements* (bepaalde spierbewegingen van het gezicht zijn aanwezig in alle mensen)
(2) *Expressiveness of facial movements* (bepaalde spierbewegingen zijn een manifestatie van dezelfde emotie in alle mensen)
(3) *Universality of attribution* (observeerders overal ter wereld kennen dezelfde betekenis toe aan deze spierbewegingen)
(4) *Correctness of attribution* (observeerders kennen de juiste emotionele betekenis toe aan deze spierbewegingen)

Iedere mogelijke toepassing van geautomatiseerde expressieherkenning is, op één of andere manier, ter bevordering van de communicatie tussen mens en machine. Omdat de laatste twee hypothesen bevragen hoe *mensen* expressies interpreteren kunnen we concluderen dat deze hypothesen niet van belang zijn voor geautomatiseerde herkenning, de *machine* is hier immers de observeerder. We willen een correcte afschatting maken van de emotionele toestand van de gebruiker. Wat een menselijke waarnemer uit dezelfde expressie op zou maken is irrelevant. We zullen de bevindingen van Russel gebruiken om erachter te komen welke nauwkeurigheid redelijkerwijs verwacht kan worden en welke emoties als universeel beschouwd worden.

2.1. **Nauwkeurigheid.** Het herkennen van expressies door mensen gebeurt niet foutloos. Fouten kunnen optreden door verkeerde interpretatie of een afwijkende expressie. Aan een afwijkende expressie is weinig te doen, deze fout zullen we ook terugzien in onze resultaten. Theoretisch gezien zou verkeerde interpretatie door een programma niet hoeven gebeuren. Maar we vermoeden dat het niet realistisch is beter te presteren dan een gemiddeld mens. In de resultaten van de review van Russel zien we dat de nauwkeurigheid waarmee expressies herkend worden tussen de 77% en de 96% ligt.

2.2. **Klassen.** Als we emoties willen herkennen zullen we moeten besluiten *welke* emoties we precies van elkaar willen kunnen onderscheiden, ofwel: van welke klassen gaan we uit? Veel artikelen vermelden niet waarom voor een bepaalde verzameling klassen gekozen wordt (bijvoorbeeld: [SGM06, DD06]). Opvallend is dat in vrijwel alle gevallen een subset van de volgende, vrij intuïtieve, expressies gebruikt word: blij (happiness), verrast (surprise), angst (fear), woede (anger) en verdriet (sadness).

Recognition Scores From Eight Studies With Literate Subjects

Culture	*n*	Facial expression "Happy"	"Surprise"	"Sadness"	"Fear"	"Disgust"	"Anger"
Western cultures							
American[a]	99	97	91	73	88	82	69
Brazilian[a]	40	97	82	82	77	86	82
American[b]	89	96.8	90.5	74.0	76.0	83.2	89.2
English[b]	62	96.2	81.0	74.5	67.0	84.5	81.5
German[b]	158	98.2	85.5	67.2	84.0	73.0	83.2
Swedish[b]	41	96.5	81.0	71.5	88.8	88.0	82.2
French[b]	67	94.5	84.2	70.5	83.5	78.5	91.5
Swiss[b]	36	97.0	85.5	70.0	67.5	78.2	91.8
Greek[b]	50	93.5	80.2	54.5	67.8	87.5	80.0
Chilean[c]	119	90.2	88.3	90.9	78	85	76
Argentine[c]	168	94.0	93	87.6	68	79.3	71.6
Estonian[d]	70	88.0	82.5	84.7	60.2	89.0	77.7
American[e]	53	96.7	85.9	72.6	69.8	71.7	64.6
American[g]	40	100	92.5	87.5	67.5	92.5	90.0
Estonian[h]	85	90	94	86	91	71	67
German[h]	67	93	87	83	86	61	71
Greek[h]	61	93	91	80	74	77	77
Italian[h]	40	97	92	81	82	89	72
Scottish[h]	42	98	88	86	86	79	84
American[h]	30	95	92	92	84	86	81
Median		96.4	87.5	80.5	77.5	82.6	81.2
M		95.1	87.4	86.5	77.3	81.1	79.1
Non-Western cultures							
Japanese[a]	29	87	87	74	71	82	63
Japanese[b]	60	93.8	79.2	66.8	58.2	55.8	56.8
African[b]	29	68.0	49.0	32.2	49.0	55.0	50.8
Kirghizian[d]	80	89.2	71.3	89.2	51.3	86.0	47.2
Malaysian[e]	30	95.8	69.8	66.4	45.6	59.2	49.8
Ethiopian[f]	100	86.8	50.5	52.0	58.8	54.8	37.3
Malaysian[g]	31	100	95	100	66.5	97.5	86
Chinese[h]	29	92	91	91	84	65	73
Japanese[h]	98	90	94	87	65	60	67
Sumatran[h]	36	69	78	91	70	70	70
Turkish[h]	64	87	90	76	76	74	79
Median		89.2	79.2	76.0	65.0	65.0	63.0
M		87.1	77.7	75.1	63.2	69.0	61.8

Note. Izard's (1971) term for "sadness" was *distress*, but it was defined as synonymous with *sadness*.
[a] Ekman, Sorenson, and Friesen (1969). [b] Izard (1971). [c] Ekman (1972). [d] Niit & Valsiner (1977). [e] Boucher and Carlson (1980); figures given are unweighted average across two stimulus sets. [f] Ducci, Arcuri, W/Georgis, and Sineshaw (1982). [g] McAndrew (1986). [h] Ekman et al. (1987).

Volgens Russel is er een set van universele expressies die cultuuronafhankelijk zijn. De volledige set bestaat ongeveer uit de volgende expressies: blij (happiness), verrast (surprise), angst (fear), woede (anger), minachting (contempt), afkeer (disgust) en verdriet (sadness). Dit verklaart waarom veel artikelen voor de in de eerste paragraaf genoemde set kiezen, deze is namelijk een subset van de universele set.

Uit hetzelfde artikel van Russell bleek helaas dat niet al deze 7 expressies even goed te herkennen zijn door mensen. Zo werd 'verdriet' vaak als 'woede' opgevat en lijkt 'minachting' voor veel mensen op 'afkeer'. 'Minachting' werd door Engelssprekende het slechtst herkend van de bovenstaande zeven. In een test waar een vrij gekozen label aan de expressie 'minachting' gekoppeld mocht worden kozen slechts 3 van de 160 onderzochten voor het label 'minachting'. De meningen waren nogal verdeeld over welk label nou bij de expressie hoorde. 'Afkeer' werd het meest genoemd met 16 van de 160. De andere expressies worden over het algemeen door westerlingen vrij accuraat herkend met gemiddeld herkenningsscores van rond de 80 procent.

Wij willen voor ons onderzoek de universele set gebruiken, maar omdat het herkennen van expressies voor computers nog steeds moeilijker is dan voor mensen hebben we besloten om 'minachting' in te wisselen voor 'neutraal'. 'Minachting' is slecht herkenbaar en omdat niet iedereen altijd boos is of lacht, is neutraal geen overbodige expressie.

2.3. **Tijdsduur van herkenning.** Naast de meer fundamentele vragen zijn er natuurlijk ook nog praktische criteria waaraan we onze oplossing willen toetsen zoals: *is dit wel snel genoeg?*. We willen graag een algoritme schrijven dat *real-time* de classificatie kan doen. De algoritmen die we zullen gebruiken hebben een lange looptijd tijdens het trainen maar kunnen daarna snel classificeren. Voor sommige toepassingen kan het vereist zijn real-time de classificatie te doen, dus we willen onszelf in dit opzicht niet beperken.

Frequency of Free-Response Labels for Expressions Reported to Be of Anger and Contempt

"Anger" expression		"Contempt" expression	
Label	Freq	Label	Freq
Frustration	49	*Disgust*	16
Anger	41	*Bored*	10
Mad	18	*Disappointment*	9
Constipated	4	*Puzzled*	6
Upset	3	*Confusion*	6
Confusion	2	*Frustration*	5
Making a decision	2	*Indifference*	5
Perturbed	2	*Smug*	5
Perplexed	2	*Contempt*	3
Irritable	2	*Perplexed*	3
Doubt	2	*Pissed off*	3
Pissed off	2	*Cynical*	3
Scorn	2	*Disgruntled*	3
Idiosyncratic	27	*Sarcastic*	3
		Anger	3
		Stupid	2
		Depression	2
		Indecisive	2
		Impatient	2
		Dissatisfaction	2
		Pain	2
		Troubled	2
		Arrogant	2
		Disbelief	2
		Perturbed	2
		Amused	2
		Idiosyncratic	46
		Other	9

Note. $n = 160$ for each type of expression.

Recognition Scores and Stimulus Presentation

	Design		
Facial expression[a]	Preview + within S	Between S	χ^2 $(1, N = 224)$
"Happy"	99.1	91.1	7.74**
"Surprise"	92.8	87.5	1.81
"Disgust"	79.5	66.1	5.07*
"Sad"	71.4	59.8	3.34
"Fear"	69.6	50.9	8.22**
"Anger"	60.7	40.2	9.45**
"Contempt"	1.8	0	
M	67.8	56.5	

Note. S = subject. Each recognition score is based on 112 observations. χ^2 was used to assess the effect of design, separately for each facial expression. The dependent measure was a free label. Data are described by Russell (1993). For "contempt," expected frequency was too low to calculate a chi-squared.

[a] Label is that provided by Matsumoto and Ekman (1988).

* $p < .05$. ** $p < .01$.

2.4. **Toepasbaarheid.** De toepasbaarheid van dit soort algoritmen is afhankelijk van veel factoren. We willen ons concentreren op het herkennen van expressies en de randvoorwaarden buiten beschouwing laten (aangezien het probleem al ingewikkeld genoeg is). Mogelijke randvoorwaarden die een rol spelen in een real-life situatie:

- Foto's genomen van verschillende standpunten
- Foto's die op een verschillende manier belicht zijn
- Foto's waar niet het hele gezicht op afgebeeld staat
- Foto's van verschillende kwaliteit

In onderzoek naar het menselijk brein zijn er diverse aanwijzingen dat het omgaan met deze randvoorwaarden los gebeurt van het herkennen van de eigenlijke expressies [HDR93, HHG00]. Ook zijn er diverse methoden in de Kunstmatige Intelligentie die het mogelijk maken om te gaan met deze randvoorwaarden [CET01]. Het is dus gerechtvaardigd dit deel los te zien van het eigenlijke herkennen van een expressie, we nemen aan dat het gezicht al voor ons gelocaliseerd is en dat de foto van voren genomen is.

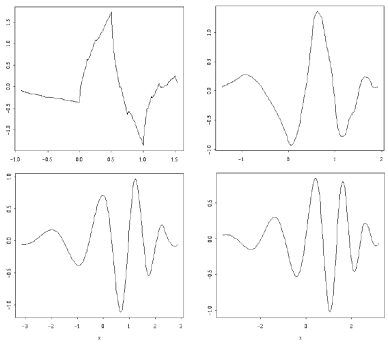
3. Achtergrond

Een plaatje wordt in een computer gerepresenteerd als een matrix van waarden. Een dergelijke representatie is erg ongeschikt voor classificatie; de data is hoog dimensionaal en het is moeilijk om patronen in zo'n matrix te herkennen. Om deze reden wordt bij het verwerken van visuele data meestal voor een alternatieve representatie gekozen die dit probleem niet heeft. De data in deze representatie wordt vervolgens aan een algoritme gevoerd dat de classificatie op zich neemt. Dit hoofdstuk behandelt hoe deze fasen in zijn werk gaan.

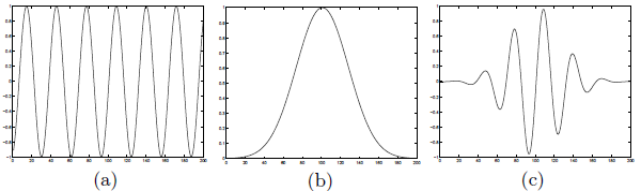
3.1. **Gabor wavelets en filters.** Als men een geluidssignaal digitaal wil beschrijven, moet het signaal eerst gesampeld worden. Op vaste intervallen wordt de amplitude gemeten om zo een discrete vorm van de golf te verkrijgen. Door deze gesampelde punten trekken we, m.b.v. fourieranalyse, een golf zodat we weten uit welke basisgolven het geluidssignaal is opgebouwd.

Deze techniek is ook bruikbaar om visuele informatie compact op te slaan. Plaatjes bevatten namelijk ook veel golf-achtige informatie (bijvoorbeeld regelmatige verlopen in kleur en samenhang met omliggende gebieden). Het probleem is alleen dat het meestal geen regelmatige golven zijn (er zijn wel golfvormen aanwezig maar ze keren meestal niet terug). Ook komen er regelmatig scherpe overgangen voor in plaatjes welke moeilijk te beschrijven zijn met een fourierreeks.

De hiervoor genoemde problemen worden opgelost door een signaal te beschrijven als een serie van wavelets. Een wavelet is een golffunctie welke *compactly supported* is (op een beperkt interval neemt deze waarden aan, buiten dit interval heeft hij de waarde nul). Voorbeelden van wavelets zijn: de Debauchies wavelets (zie figuur 1) en de Gabor wavelet (zie figuur 2).



Figuur 1. Debauchies wavelets.



Figuur 2. Gabor wavelet.

Gaborwavelet. Een veelgebruikte wavelet is de Gabor wavelet. Deze wavelet is gedefinieerd als de vermenigvuldiging van een harmonische functie (sinus) met een Gaussische functie (zie figuur 2). De Gabor wavelet zoals wij hem zullen gebruiken is complex en 2-dimensionaal, een algemene definitie is te vinden in [Mov]. Wij zullen een vereenvoudigde definitie hanteren zoals gebruikt in [BLL+04] en [LVB+93]):

$$\begin{aligned} g(\mathbf{x},\mathbf{k},\sigma) &= s(\mathbf{x},\mathbf{k},\sigma)\cdot w(\mathbf{x},\mathbf{k},\sigma) \\ s(\mathbf{x},\mathbf{k},\sigma) &= exp(i(\mathbf{k}\bullet\mathbf{x})) - exp(-\tfrac{\sigma^2}{2}) \\ w(\mathbf{x},\mathbf{k},\sigma) &= \tfrac{\mathbf{k}^2}{\sigma^2} exp(-\tfrac{\mathbf{k}^2\cdot\mathbf{x}^2}{2\sigma^2}) \end{aligned}$$

Hierbij is $s(\mathbf{x},\mathbf{k},\sigma)$ de complexe sinusoïde[2] en is $w(\mathbf{x},\mathbf{k},\sigma)$ de twee dimensionale Gaussische functie. Deze functies zijn geparameteriseerd met drie variabelen:

x: Het punt uit het domein waarvan we de waarde willen weten

k: Deze vector bepaalt de golflengte en de richting van de complexe sinusoïde. De richting van de golf is gelijk aan de richting van **k**. Voor de golflengte geldt hoe groter $|\mathbf{k}|$, hoe korter de golflengte; iets preciezer: $\lambda = \frac{2\pi}{|\mathbf{k}|}$

σ**:** Deze reeële waarde bepaald hoeveel golflengtes de wavelet groot is, oftewel de breedte van de Gaussische envelop. Bij een kleine σ zullen er minder golfbeweging zichtbaar zijn in de wavelet omdat de Gaussische functie de rest van de sinusoïde uitdooft.

Om de wavelet te kunnen visualiseren gebruiken we Euler's identiteit[3] om het harmonische deel uit te kunnen drukken in een imaginair deel en een reeël deel. Dit levert de plots op in figuur 3.

De mogelijke richtingen van van **k** worden bepaald door punten op de eenheidscirkel. De ν verschillende richtingen worden gelijkmatig verdeeld over één helft van de cirkel (de gespiegelde richtingen zijn niet nodig). Waarden voor **k** vinden we met de volgende formules:

$$\begin{aligned} \mathbf{k} &= \tfrac{2\pi}{\lambda}\langle cos(\alpha), sin(\alpha)\rangle \\ \alpha &\in \{\tfrac{0\pi}{\nu}, \tfrac{1\pi}{\nu}, \cdots, \tfrac{(\nu-1)\pi}{\nu}\} \\ \lambda &\in \text{Golflengtes} \end{aligned}$$

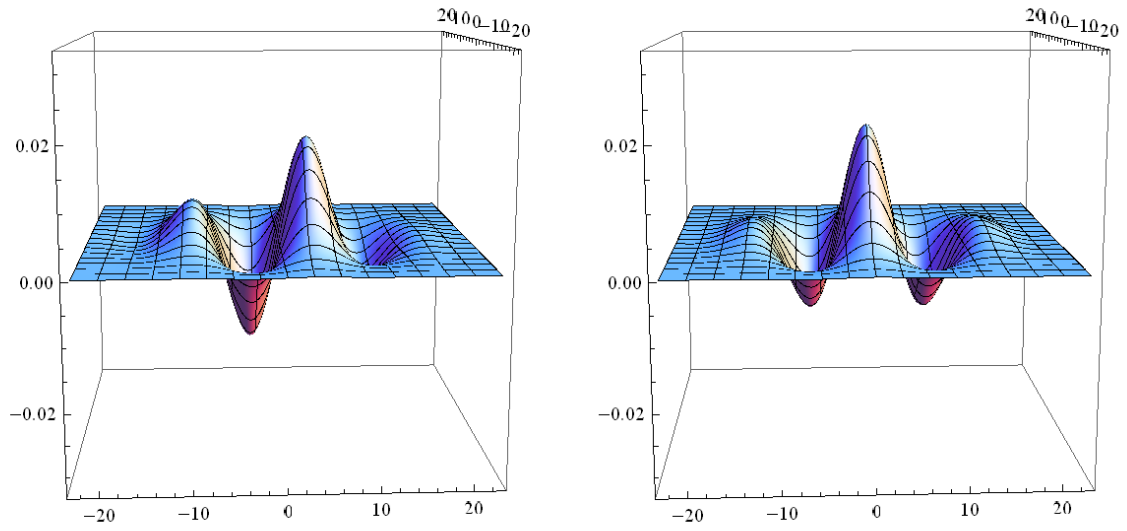
Gaborwavelet als filter. Het berekenen van een waveletdecompositie voor deze wavelets is een tijdrovende kwestie (dit moet gebeuren met biorthogonale functies, zie [Yao93] voor meer informatie). Daarom wordt een andere aanpak gebruikt; we gebruiken de wavelet als een filter operatie.

Als maat voor de response van de filter op een bepaald punt uit I (het domein van het plaatje) gebruiken we de convolutie:

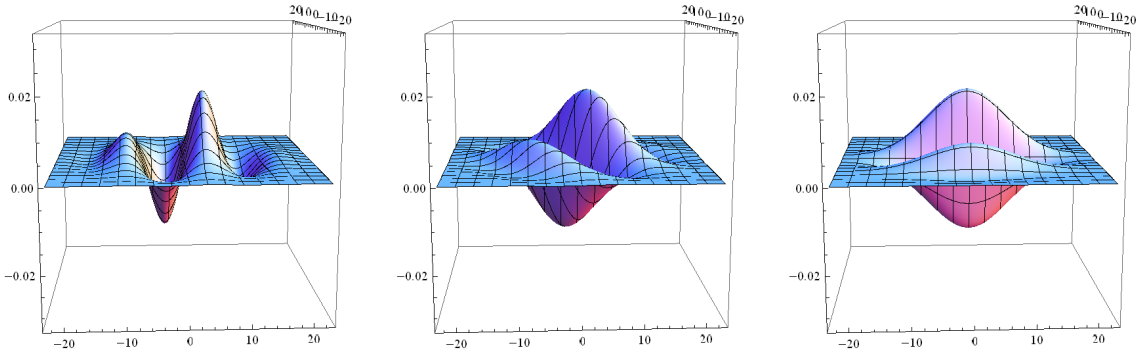
$$(I * g)(\mathbf{x}) = \int_{-\infty}^{\infty} I(\mathbf{x}-\tau)\cdot g(\tau,\mathbf{k},\sigma)d^2\tau$$

[2] De extra term aan de rechterkant is om voor de zogenaamde DC-waarde van de filter te compenseren, hiermee wordt vermeden dat de respons van de filter afhankelijk is van de intensiteit van de afbeelding

[3] $e^{i\theta} = cos(\theta) + isin(\theta)$



FIGUUR 3. Links het imaginaire deel, rechts het reëele deel van de wavelet.

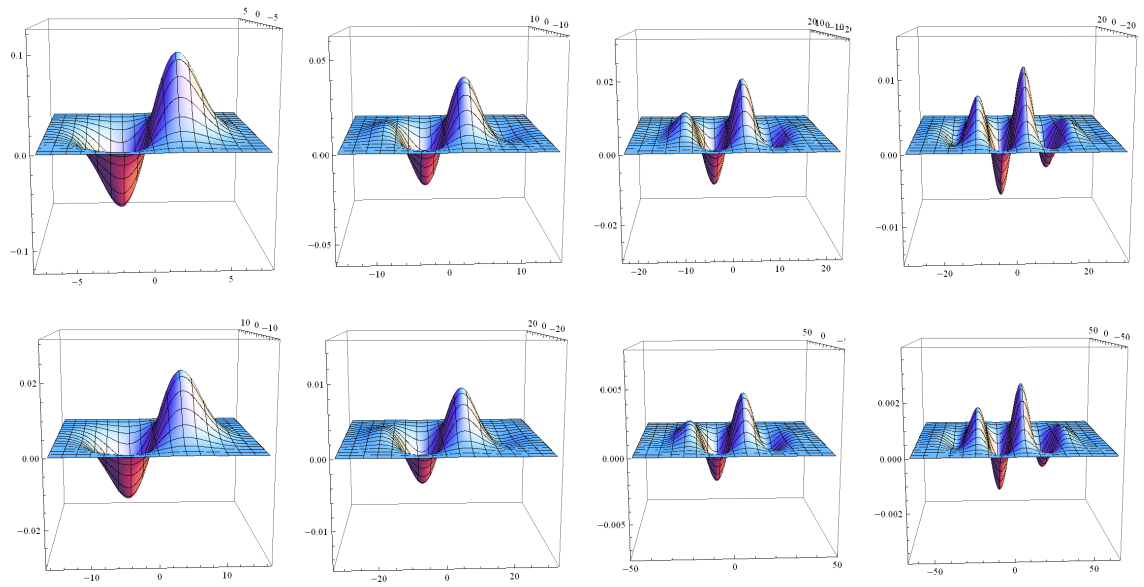


FIGUUR 4. Voorbeeld met verschillende hoeken v.l.n.r. $\alpha = 0$, $\alpha = \frac{\pi}{4}$, $\alpha = \frac{\pi}{2}$.

We hoeven voor onze toepassing alleen de discrete variant te beschouwen:

$$(I * g)(\mathbf{x}) = \sum_{\tau \in Domein(g)} I(\mathbf{x} - \tau) \cdot g(\tau, \mathbf{k}, \sigma)$$

Deze functie geeft voor een punt $\mathbf{x}$ aan hoe goed de wavelet op het plaatje past met dat punt als oorsprong van de wavelet. Het berekenen van deze operatie kost per punt $O(n)$ operaties, met n het aantal pixels. Het berekenen van de totale filter response $(I * g)$ kost $O(n^2)$ omdat we voor $\mathbf{x}$ n verschillende waarden kunnen kiezen. Dit kunnen we reduceren door gebruik te maken van het feit dat het berekenen van de convolutie een lineaire operatie is in het frequentiedomein (we hoeven dan alleen de coëficienten puntsgewijs te



FIGUUR 5. Voorbeeld op de bovenste rij is $\lambda = 15$ en op de onderste rij geldt $\lambda = 32$ verder geldt v.l.n.r. $\sigma = 1$, $\sigma = 2$, $\sigma = 3$, $\sigma = 4$ (hierin is te zien hoe veranderingen van σ geen invloed heeft op de golflengte maar wel op de volledige grootte van de wavelet).

vermenigvuldigen). Dan wordt het berekenen van de FFT en de iFFT de dominante factor in de complexiteitsanalyse waardoor de convolutie operatie nog slechts $O(n \cdot log2(n))$ kost.

Gaborfilter als feature. In de computer vision speelt het detecteren van randen een grote rol. De Gaborfilter reageert sterk als de richting van de golf loodrecht staat op de richting van de rand. De bovenstaande convolutie geeft een complexe respons. Zoals opgemerkt in [LVB+93] oscilleren dan het reëele en complexe deel van de wavelet met een karakteristieke frequentie. We kiezen ervoor de absolute waarde te nemen van deze complexe respons. Op deze manier krijgen we een reëele waarde ($|(I * g)(\mathbf{x})|$) welke aangeeft of op een bepaald punt $\mathbf{x}$ een rand is of niet, en dit is een handige representatie om mee aan de slag te gaan voor een herkenningsprocedure. Deze representatie wordt ook wel de *Gabor magnitude representatie* genoemt.

Samenvatting. Een Gabor wavelet g is gedefinieerd door een richting α en een golflengte λ. Gegeven een plaatje I met n pixels kunnen we de Gabor magnitude representatie $|(I * g)|$ berekenen in $O(n \cdot log2(n))$ operaties. Deze representatie geeft voor ieder punt in I aan hoe goed de wavelet past op dat deel van het plaatje. Stel we willen dit berekenen voor ν richtingen en |Golflengte| golflengtes dan resulteert dit in $n \cdot \nu \cdot$ |Golflengte| features.

3.2. **Adaboost.** Adaboost is een algoritme om classifiers te trainen en te combineren, adaboost gebruikt dus andere classifiers en combineert ze tot een nieuwe. We behandelen hier het adaboosting algoritme voor binaire classificatie zoals beschreven in [FS95].

Neem een dataset $S = (x_1, y_1), \ldots, (x_N, y_N)$ uit $X \times Y$, met $Y = \{0, 1\}$ als klasse labels. Nu zijn we op zoek naar een classifier die voor een willekeurige x de bijbehorende klasse y vaststelt. Stel we hebben een algoritme *WeakLearn* dat ons, gegeven een aantal voorbeelden uit S en een distributie $p : S \to \mathbb{R}$, een hypothese $h : X \to [0..1]$ geeft welke de fout $E_{(x,y)\sim p}(|h(x) - y|)$ minimaliseert. Dan kan adaboost in een serie van rondes een gecombineerde hypothese samenstellen. Op iedere ronde wordt een nieuwe hypothese gegenereerd met *WeakLearn*. Vervolgens wordt de distributie p aangepast aan de fout die gemaakt wordt zodat de volgende ronde een hypothese gegenereerd wordt die voor deze fout kan compenseren. De uiteindelijke classifier is een gewogen som van de gegeneerde hypothesen.

Het algoritme. Het adaboosten is een for-loop over t welke doorgaat totdat een bepaalde stopconditie bereikt is. Deze stopconditie kan bijvoorbeeld afhankelijk zijn van het aantal iteraties dat reeds geschied is of waarden die de gecombineerde hypothese geeft op de dataset. In de body van de loop wordt een vector van gewichten w bijgehouden welke per element uit de dataset aangeeft hoe belangrijk dat element is. De vector p is slechts een genormaliseerde versie van w.

De eerste stap in de for-loop is het zorgen dat p een nette distributie is door de gewichten uit w te nemen en deze te normaliseren. Daarna wordt de distributie p aan *WeakLearn* gegeven en levert deze een hypothese h_t welke het beste werkt volgens de meegegeven distributie. Met deze hypothese berekenen we de fout ε_t:

$$\varepsilon_t = \sum_{i=1}^{N} p_i \cdot |h_t(x_i) - y_i|$$

Van deze fout kan de weegfactor β_t berekend worden:

$$\beta_t = \frac{\varepsilon_t}{1.0 - \varepsilon_t}$$

Deze weegfactor wordt later gebruikt in de gecombineerde classifier om h_t te wegen met $\ln(\frac{1}{\beta_t})$. Verder wordt deze weegfactor ook nu gebruikt om de gewichten aan te passen:

$$w_i = w_i \cdot \beta_t^{1.0-|h_t(x_i)-y_i|}$$

Voorbeelden uit de dataset die fout geclassificeerd werden door h_t worden zo verhoogd en de correct geclassificeerde voorbeelden worden minder belangrijk gemaakt. Vervolgens worden de weegfactor β_t en de bijbehorende hypothese h_t samen opgeslagen in de gecombineerde hypothese h_f. Het volledige algoritme ziet er als volgt uit:

```
1  def Adaboost(X = {(x_1, y_1), ..., (x_N, y_N)}, WeakLearn, StoppingCondition):
2    h_f = []
3    for (i = 1; i ≤ N; i++):
4      w_i = 1.0
5
6    for (t = 1; StoppingCondition(h_f, X) == False; t++):
7
8      for (i = 1; i ≤ N; i++):
9        p_i = w_i / (Σ_{i=1}^{N} w_i)
10
```

```
11      h_t = WeakLearn(X, p)
12
13      ε_t = Σ_{i=1}^{N} p_i · |h_t(x_i) − y_i|
14
15      β_t = ε_t / (1.0 − ε_t)
16
17      for (i = 1; i ≤ N; i++):
18          w_i = w_i · β_t^{1.0−|h(x_i)−y_i|}
19
20      h_f += (h_t, β_t)
21
22  return h_f
```

De classificatie. Stel het adaboost algoritme heeft T iteraties gedraaid voordat de stop conditie het proces gestopt heeft. Dan hebben we een gecombineerde hypothese h_f gevonden waarmee we als volgt kunnen classificeren:

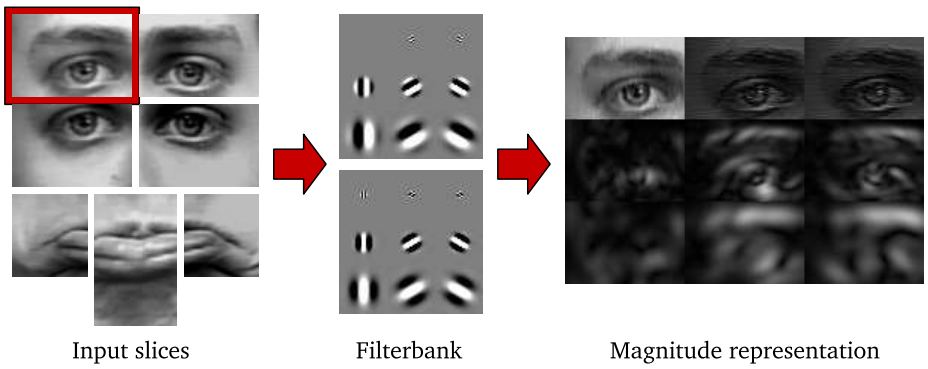
```
1 def Classify(h_f =< (h_1, β_1), ..., (h_T, β_T) >, x):
2
3   output    = Σ_{t=1}^{T} ln(1/β_t) · h_t(x)
4
5   threshold = 1/2 Σ_{t=1}^{T} ln(1/β_t)
6
7   return output ≥ threshold
```

4. Methode

De experimentele opzet is gebaseerd op de opzet van één van de experimenten uit [BLL+04]. De belangrijkste aanpassing is dat het gezicht in zeven stukken opgedeeld wordt voordat de Gabor magnitude representatie berekent word. De verschillende stadia van het trainen zijn:

(1) Preprocessing (lokalisatie, opdelen in stukken)
(2) Berekenen Gabor magnitude representatie
(3) Trainen adaboost

Het berekenen van de Gabor magnitude representatie gebeurt door middel van een filter bank. Deze filterbank bestaat uit Gabor filters met een verschillende golflengte λ en verschillende oriëntatie α. Voor ieder van deze filters wordt de convolutie genomen met één stuk s uit het gezicht. Het resultaat van deze operatie geeft aan hoe goed de betreffende filter past op iedere mogelijke locatie (x, y) van de input. We beschouwen iedere filter output als een aparte hypthese, een dergelijke hypothese is dus gedefinieerd door het tuple $(\lambda, \alpha, s, x, y)$.



Figuur 6. Berekenen Gabor magnitude representatie

Voor iedere emotie gebruiken we adaboost om een gewogen combinatie van deze filter responses te bepalen welke aangeeft of de input wel of niet tot die bepaalde klasse behoort. Op iedere ronde t levert *WeakLearn* de hypothese $h_t = (\lambda, \alpha, s, x, y)$ welke, rekeninghoudend met distributie p, de kleinste fout maakt op de dataset. Een gewogen combinatie van deze filter responses is de uiteindelijke classifier.

We zullen de threshold uit het adaboost classificatie algoritme van de output van de gecombineerde hypothese aftrekken zodat de classificatie geen binaire maar een reële waarde oplevert. We krijgen dus een reële waarde voor iedere emotie. De gecombineerde hypothese met de hoogste waarde kiezen we als uitkomst van ons classificatie proces.

4.1. Datasets.

We hebben op diverse plaatsen datasets gevonden welke bruikbaar zijn om ons classificatie algoritme op te testen. Enkele voorbeelden:

FERET database: Een database opgezet door het NIST in de Verenigde Staten om onderzoek te ondersteunen naar automatische identificatie van mensen. Een deel van de files is ook geschikt om expressie herkenningsalgoritmen mee te testen. We hebben deze dataset aangevraagd maar nooit mogen ontvangen. Meer informatie is te vinden op: http://www.itl.nist.gov/iad/humanid/feret/

RU BSI dataset: Een dataset opgesteld op de Radboud Universiteit door het Behavioural Science Institute. Proefpersonen zijn geïnstrueerd om bepaalde Facial Action Units te gebruiken en dit is vastgelegd. De beschrijving van de dataset klinkt veelbelovend, echter kwamen we te laat achter het bestaan van deze dataset. Zie: http://www.ru.nl/wetenschapsagenda/huidige_editie/vm/item_721253/beeld_duizenden/

MMIFaceDB: Dataset van Maja Pantic van het Imperial College London. We hebben toegang gekregen tot deze dataset en de data deels voorbewerkt. Het probleem met deze set is echter de kleine hoeveelheid verschillende sessies (175 samples). Dus we zijn eerst verder gaan zoeken naar een grotere dataset. Voor meer informatie: http://www.mmifacedb.com/

Cohn-Kanade dataset: Deze dataset is ook opgezet als benchmark voor herkenningsalgoritmen. De plaatjes zijn geannoteerd met de Facial Action Units welke de proefpersonen na instructie vertonen. In deze dataset zitten 487 samples. We hebben ervoor gekozen met deze dataset verder te werken en de MMIFaceDB even te laten voor wat het is. Voor meer informatie: http://vasc.ri.cmu.edu/idb/html/face/facial_expression/index.html

We zullen dus de Cohn-Kanade dataset gebruiken voor onze experimenten.

4.2. **Preprocessing.** Het voorbereiden van de dataset is gedaan in samenwerking met Ruben Muijrers [Mui09]. Voor die experimenten is PCA gebruikt als tussen-representatie. PCA stelt een aantal principale componenten vast op basis van de covariantie matrix. Deze componenten worden gekozen op basis van de variantie in de data en zijn misschien niet de meest geschikte features om emoties mee te classificeren. We besloten de gezichten op te delen in verschillende stukken in de hoop zo principale componenten te krijgen die meer gecorreleerd zijn met bepaalde emoties. Voor de keuze *welke* onderdelen van het gezicht dit moeten worden, hebben we ons laten inspireren door [DC99]. We zijn tot de volgende overlappende stukken gekomen:

- Linkeroog boven (80*55 pixels)
- Linkeroog onder (80*55 pixels)
- Rechteroog boven (80*55 pixels)
- Rechteroog onder (80*55 pixels)
- Linkermondhoek (50*55 pixels)
- Middelste deel mond (55*80 pixels)
- Rechtermondhoek (50*55 pixels)

We hebben een aantal toolkits gevonden om deze onderdelen te detecteren[4]:

MPTLab: Volgens [SLB+06] zou deze bibliotheek 90% van de gezichten moeten herkennen en nauwelijks false positives moeten hebben. In onze experimenten presteert hij echter aanzienlijk slechter. Bovendien detecteert de software alleen de bounding box van het gezicht. De grootte van de bounding box die opgeleverd wordt is niet in constante verhouding met de grootte van het gezicht. Omdat wij relatief t.o.v. deze bounding box de verschillende onderdelen van het gezicht moeten bepalen, is deze bibliotheek voor ons niet goed bruikbaar. Voor meer informatie over MPTLab zie: http://mplab.ucsd.edu/grants/project1/free-software/mptwebsite/API/index.html

OpenCV's HaarDetectObjects: OpenCV is een algemene computer vision library. Oorspronkelijk ontwikkeld door Intel, maar nu beschikbaar als open source software.

[4]Met dank aan Xuelong Li voor het wijzen op het bestaan van OpenCV en Stasm

Deze bibliotheek bevat een object detector, gebaseerd op [VJ01], die gebruikt maakt van een cascade van haar-like features. De parameters voor gezichtsdetectie worden ook met de bibliotheek meegeleverd. Deze procedure levert ook alleen de bounding box op. We hebben niet getest hoe deze werkt omdat Stasm beter aan onze eisen voldeed. Voor meer informatie over OpenCV zie: http://opencv.willowgarage.com/wiki/

Stasm: Stasm gebruikt eerst technieken uit [VJ01] of [BRK98] om de locatie van het gezicht vast te stellen. Vervolgens gebruikt het Active Shape Models [Mil07] om de posities van allerlei facial landmarks vast te stellen. Deze facial landmarks kunnen wij weer gebruiken om de juiste onderdelen van het gezicht uit te snijden. De resultaten van Stasm zijn erg betrouwbaar en daarom hebben we besloten met deze software verder te gaan voor het preprocessen. Meer informatie over Stasm is te vinden op: http://www.milbo.users.sonic.net/stasm/

We schalen de gezichten zodanig dat ze allemaal dezelfde afmetingen hebben. Vervolgens snijden we op basis van de posities van de facial landmarks de benodigde onderdelen van het juiste formaat uit. Verder compenseren we voor het verschil in lichtintensiteit door de waarden te transleren zodat het gemiddelde op de helft van de maximale waarde komt te liggen.

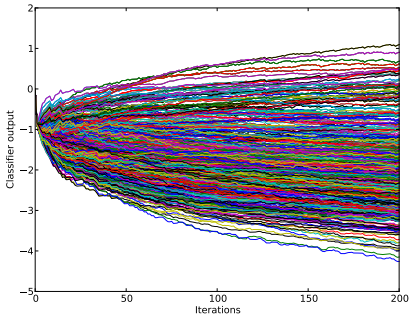
We gebruiken 10-fold crossvalidatie om te meten hoe de algoritmen het doen op ongeziene data. Om te voorkomen dat er een bias ontwikkeld wordt voor een bepaalde klasse moet iedere klasse even vaak voorkomen in iedere training set. We garanderen dit door bepaalde voorbeelden dubbel toe te voegen (uiteraard alleen binnen één fold zodat de validatie set echt ongezien is).

4.3. **Gabor.** Voor de Gabor filterbank moeten we een aantal parameters kiezen: het aantal oriëntaties ν, de verzameling golflengtes Golflengtes en de beedte van de Gausssische envelop σ. Omdat ons experiment gebaseerd is op [BLL+04] zullen we ons hierdoor laten inspireren voor een keuze voor deze parameters. Zij kiezen voor 8 oriëntaties en golflengtes van 2 tot 32 pixels in 1/2 octave stappen. Er wordt alleen geen waarde gegeven voor σ, dus heb ik daarvoor de waarde 2.0 gekozen. Het systematisch proberen van verschillende waarden voor deze parameter laten we voor toekomstig onderzoek. Dus:

$$\begin{aligned} \text{Golflengtes} &= \{2, 3, 4, 5, 7, 10, 15, 22, 32\} \\ \nu &= 8 \\ \sigma &= 2.0 \end{aligned}$$

4.4. **Adaboost.** Voor Adaboost is de enige parameter die we vast moeten stellen welke stop-conditie we willen gebruiken. In [BLL+04] kiest men hypotheses totdat de outputwaarden van de gecombineerde hypothese voor positieve en negatieve trainingsvoorbeelden volledig van elkaar gescheiden zijn. In figuur 1 van dat artikel plotten ze hoe deze outputwaarden zich ontwikkelen tijdens het boosting proces. Daarop zien we dat de twee klassen inderdaad keurig uiteen lopen. Dezelfde grafiek (figuur 7) laat voor mijn opzet echter een ander verloop zien. De totale distributie loopt wel uiteen maar er ontstaat geen gat tussen de negatieve en positieve voorbeelden. Om deze reden hebben we besloten om een vast aantal Adaboost iteraties T te doen en verschillende waarden voor T te gebruiken om zo het optimum vast te stellen.

Hoewel het nergens in de tekst expliciet vermeld staat, suggereert Figuur 2 in [BLL+04] dat de filter outputs ge-threshold worden voordat er geadaboost wordt. We hebben dit voor de zekerheid ook geprobeerd maar zonder succes. We zien wel dat dan de distributie niet meer tijdens de eerste ronden naar -1 neigt en dat er snel een 100% score op de trainingsdata gehaald wordt, maar een slechtere score op ongeziene voorbeelden. De distributie vertoont ook geen scheiding tussen de positieve en negatieve voorbeelden.



FIGUUR 7. Outputwaarden op de dataset tijdens het adaboosten

5. Resultaten

Het trainen en testen van een classifier met $T = 200$ duurt ongeveer 1 dag op een AMD Opteron 8356 (gebruikmakend van slechts één core) en verbruikt ongeveer 6 GB aan geheugen. We hebben voor T alle waarden tussen 30 en 200 geprobeerd met 10 als stapgrootte.

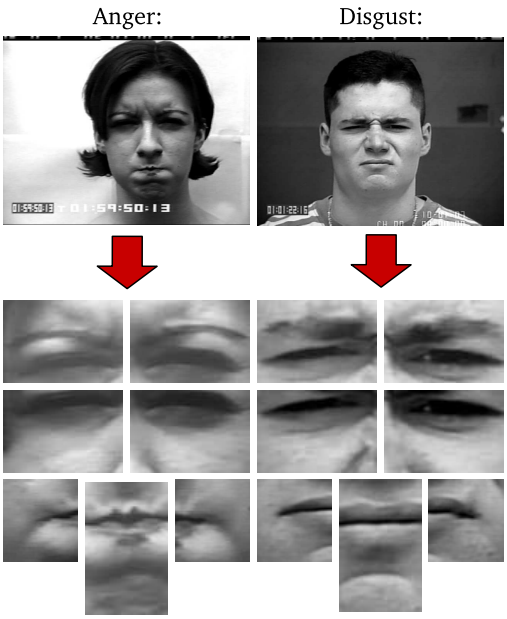
De beste resultaten worden behaald met $T = 90$ waarbij we een score halen van 69% en we de volgende confusie matrix verkrijgen op ongeziene data:

	SUR	ANG	DIS	FEA	HAP	NEU	SAD
SUR	90	0	0	6	0	2	3
ANG	1	53	15	5	1	4	8
DIS	1	19	65	5	1	8	1
FEA	0	0	8	58	8	2	2
HAP	1	0	0	15	86	1	1
NEU	3	10	8	4	2	68	20
SAD	4	18	4	7	2	13	65
	90%	53%	65%	58%	86%	69%	65%

```
Overall performance: 69.4842406877%
```

Hiermee heb ik een vergelijkbaar resultaat als [Mui09], die $\sim 71\%$ haalt op dezelfde voorbewerkte data. Maar het valt behoorlijk tegen wanneer je het resultaat afzet tegen de $\sim 90\%$ die [BLL+04] haalt met vergelijkbare technieken op dezelfde dataset!

Wat ook opvallend is zijn de overeenkomsten tussen de confusie matrices uit [Mui09] en de mijne. Al deze classifiers hebben moeite om het onderscheid te maken tussen de volgende klassen: (ANG, DIG), (FEA, HAP), (NEU, SAD) en (SAD, ANG). Wanneer we een aantal verkeerd geclassificeerde voorbeelden bekijken zien we het volgende:



Links een typisch voorbeeld van iemand uit de ANG klasse welke correct geclassificeerd wordt. Rechts hebben we een voorbeeld dat eigenlijk tot de DIS klasse behoort maar geclassificeerd wordt als zijnde ANG. Eronder staan de bijbehorende onderdelen zoals die in de preprocessing fase uitgesneden worden. Hier zien we de gelijkenis tussen de twee emoties en dat met het uitsnijden bepaalde relevante stukken van het gezicht verloren gaan, zoals de neus en de wangen. Het algoritme heeft geen mogelijkheid om bijvoorbeeld het optrekken van de neus te detecteren als iemand iets vies vindt, of de bolling van de wangen te observeren als iemand vrolijk is.

Een voor de hand liggende manier om uit te sluiten of de preprocessing het probleem is, is het experiment aan te passen en te herhalen. Het is niet ingewikkeld de code aan te passen zodat het gehele gezicht gebruikt wordt als invoer. De preprocessing zal echter wel opnieuw gedaan moeten worden. Verder onderzoek zou vast moeten stellen of dit daadwerkelijk is wat de performance van het algoritme tegenhoudt.

6. Conclusie

Door eerst de Gabor magnitude representatie te bepalen en de individuele filter outputs als features te gebruiken in het adaboosting algoritme, hebben we een classifier verkregen die met 69% nauwkeurigheid de emotie van de input vaststelt. Dit is vergelijkbaar met het resultaat uit [Mui09]. In de literatuur worden er echter significant betere resultaten gehaald met een vergelijkbare opzet en op dezelfde dataset. Inspectie van de verkeerd geclassificeerde voorbeelden suggereert dat er tijdens het voorbewerken van de data cruciale informatie verloren gaat. Dit zou verklaren waarom de resultaten hetzelfde zijn als in [Mui09], waar dezelfde preprocessing gebruikt is, maar slechter zijn als in [BLL+04], waar vergelijkbare technieken gebruikt worden. Een nieuw experiment met aangepaste voorbewerking kan hierover uitsluitsel bieden.

Meer in het algemeen was het waarschijnlijk een goed idee geweest om zorgvuldiger de regio's te kiezen waarin we de gezichten opdeelden. De technieken die in deze scriptie beschreven staan vereisen niet dat het gezicht opgedeeld word in regio's en kunnen zelfs gebruikt worden om de meest relevante regio's te bepalen. Adaboost kiest die features die het meest relevant zijn voor de classificatie. Door deze weer terug te herleiden naar de juiste posities in het beeld kan je vaststellen welke punten uit het beeld gebruikt worden tijdens de classificatie. De regio's met de meeste punten zullen op één of andere manier in de opdeling van het gezicht terug moeten komen.

Ondanks deze kanttekeningen kunnen we concluderen dat onze classificatie voldoet aan de vooraf gestelde criteria uit hoofdstuk 2. De uiteindelijke classificatie is met een nauwkeurigheid van 69% duidelijk beter dan willekeurig gokken. Verder geldt voor bepaalde klassen, zoals SUP en HAP, zelfs dat de nauwkeurigheid van mensen benaderd wordt.

Referenties

[BLL+04] Marian Stewart Bartlett, Gwen Littlewort, Claudia Lainscsek, Ian R. Fasel, and Javier R. Movellan. Machine learning methods for fully automatic recognition of facial expressions and facial actions. In *SMC (1)*, pages 592–597, 2004.

[BRK98] S. Baluja, HA Rowley, and T. Kanade. Neural network-based face detection. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 20(1):23–38, 1998.

[CET01] T.F. Cootes, G.J. Edwards, and C.J. Taylor. Active appearance models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(6):681–685, June 2001.

[DC99] M.N. Dailey and G.W. Cottrell. PCA = Gabor for expression recognition. *UCSD CSE TR CS-629*, 1999.

[DD06] F. Dornaika and F. Davoine. Facial expression recognition using auto-regressive models. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 2, pages 520–523, 20-24 Aug. 2006.

[FS95] Y. Freund and R. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory*, pages 23–37. Springer, 1995.

[HDR93] G. W. Humphreys, N. Donnelly, and M. J. Riddoch. Expression is computed separately from facial identity, and it is computed separately for moving and static faces: neuropsychological evidence. *Neuropsychologia*, 31(2):173–181, Feb 1993.

[HHG00] Haxby, Hoffman, and Gobbini. The distributed human neural system for face perception. *Trends Cogn Sci*, 4(6):223–233, Jun 2000.

[JS05] A. Jaimes and N. Sebe. Multimodal human computer interaction: A survey. *Computer vision in human-computer interaction*, pages 1–15, 2005.

[LVB+93] Martin Lades, Jan C. Vorbrüggen, Joachim M. Buhmann, Jörg Lange, Christoph von der Malsburg, Rolf P. Würtz, and Wolfgang Konen. Distortion invariant object recognition in the dynamic link architecture. *IEEE Trans. Computers*, 42(3):300–311, 1993.

[Mil07] S. Milborrow. *Locating Facial Features with Active Shape Models*. Master's thesis. University of Cape Town (Department of Image Processing), 2007. http://www.milbo.users.sonic.net/stasm.

[Mov] Javier R. Movellan. Tutorial on gabor filters.

[Mui09] Ruben Muijrers. *Expressieherkenning met PCA en SVM*. Bachelor thesis. Radboud University Nijmegen, 2009.

[PR00] M. Pantic and L.J.M. Rothkrantz. Automatic analysis of facial expressions: The state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1424–1445, 2000.

[Rus94] James A. Russell. Is there universal recognition of emotion from facial expression? a review of the cross-cultural studies. *Psychological Bulletin*, 115(1):102–141, 1994.

[SGM06] Caifeng Shan, Shaogang Gong, and Peter W. McOwan. Dynamic facial expression recognition using. a bayesian temporal manifold model. In *BMVC*, Edinburgh, September 2006.

[SLB+06] JM Susskind, G. Littlewort, MS Bartlett, J. Movellan, and AK Anderson. Human and computer recognition of facial expressions of emotion. *Neuropsychologia*, 2006.

[VJ01] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features, 2001.

[Yao93] J. Yao. Gabor transform: theory and computations. In A. F. Laine, editor, *Proc. SPIE Vol. 2034, p. 137-148, Wavelet Applications in Signal and Image Processing, Andrew F. Laine; Ed.*, volume 2034 of *Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference*, pages 137–148, November 1993.

Bijlage A. Implementatie

Voor de implementatie heb ik gebruik gemaakt van de programmeertaal Python en de OpenCV bibliotheek. De OpenCV bibliotheek is geïmplementeerd in C en geeft ons binnen een Python script snelle matrix bewerkingen.

A.1. **Gabor.** De kernel van een filter wordt door de volgende functies naar een matrix geschreven:

```
def Gaussian   (x, k, delta):
  return (k**2 / delta**2) *
         exp(-(k**2 * x**2) / (2.0 * delta**2))

def ImagKernel (x, k, delta):
  return Gaussian(x, k, delta) *  sin(x * k)

def RealKernel (x, k, delta):
  return Gaussian(x, k, delta) * (cos(x * k) - exp(-delta**2/2))

def PlotKernel(mat, kernel, *args):
  cv.Zero(mat)
  x_offset = mat.width  / 2
  y_offset = mat.height / 2
  for x in range(0, mat.width):
    for y in range(0, mat.height):
      mat[y, x] = kernel(Vector(x-x_offset,y-y_offset), *args)
  return
```

De code die de bovenstaande kernel gebruikt om de filter operatie uit te voeren is om performance redenen een beetje onoverzichtelijk. Maar een versimpelde versie ziet er als volgt uit:

```
def Filter(input, k, delta, dest):
  result_imag = cv.CloneImage(input)
  result_real = cv.CloneImage(input)

  size        = int(ceil( delta * get_wavelength(k) ))
  filter_imag = cv.CreateMat(size, size, cv.CV_32FC1)
  filter_real = cv.CreateMat(size, size, cv.CV_32FC1)
  PlotKernel(filter_imag, ImagKernel, k, delta)
  PlotKernel(filter_real, RealKernel, k, delta)

  cv.Filter2D(input, result_imag, filter_real, (-1,-1))
  cv.Filter2D(input, result_real, filter_real, (-1,-1))
  cv.Pow(result_imag, result_imag, 2)
  cv.Pow(result_real, result_real, 2)
  cv.Add(result_imag, result_real, dest)
  cv.Pow(dest, dest, 0.5)
  return dest
```

A.2. **Adaboost.** We gebruiken een simpele datastructuur om: de data, het label en de p en w distributies bij elkaar op te slaan:

```
class AdaboostExample():
  def __init__(self, data, label, w):
    self.data  = data
    self.label = label
    self.w     = w
    self.p     = w
```

De uiteindelijke implementatie in Python is dan een vrij rechttoe rechtaan vertaling van wat er in 3.2 beschreven staat. Deze implementatie hoeft niet erg snel te zijn omdat de bottleneck toch zal zitten in het uitvoeren van WeakLearn (in de code hier beneden is dat gethypothesis).

```
def AdaboostTrain(examples,
                  gethypothesis,
                  usehypothesis,
                  label,
                  stopping_condition):

  def h(t, x): return usehypothesis(x.data, t)
  def y(x):    return 1.0 if x.label == label else 0.0
  combined_hypothesis = []

  while not stopping_condition(combined_hypothesis,
                               label,
                               examples):
    # 1. Setup example.p's:
    w_total = sum([example.w for example in examples])
    for example in examples:
      example.p = example.w / w_total

    # 2. Set new_ht by calling "WeakLearn"
    new_ht = gethypothesis(examples, label)

    # 3. Calculate the error of this new hypothesis
    error = 0.0
    for example in examples:
      error += example.p * abs(h(new_ht, example) - y(example))

    # 4. Set new_beta
    new_beta = error / (1.0 - error)

    # 5. Update weightvector
    for example in examples:
      example.w *= new_beta **
                   (1.0 - abs(h(new_ht, example) - y(example)))

    combined_hypothesis.append((new_ht, new_beta))

  return combined_hypothesis
```

Omdat er een binaire classifier gemaakt wordt voor iedere emotie moeten we de outputs van deze classifiers vergelijken om de uiteindelijke classificatie te doen. We laten het classificatie proces een reële waarde opleveren zodat we de klasse kunnen kiezen welke de beste score oplevert:

```
def AdaboostClassify(combined_hypothesis,
                     usehypothesis,
                     data):
  threshold = 0.0
  value     = 0.0
  for hypothesis, beta in combined_hypothesis:
    lb = log(1.0 / beta)
    threshold += lb
    value     += lb * usehypothesis(data, hypothesis)
  return value - 0.5 * threshold
```

A.3. **Experiment.** Er zijn wat simpele datastructuren zoals *FaceSample* en *CrossValidationDS* voor het bewaren van de data. Deze zijn niet interessant genoeg om te hier uitgebreid te bespreken dus we zullen ons beperken tot hetgeen strikt noodzakelijk is om de rest te begrijpen. De Gabor magnitude representatie wordt helemaal in het begin van het programma berekend en opgeslagen in het *self.slices_gabor* veld van *FaceSample*. *self.slices_gabor[0]* verwijst naar de Gabor magnitude representatie van het eerste onderdeel van het gezicht. Deze magnitude representatie is een matrix met alle filter responses voor alle golflengten en oriëntaties, dus in de code wordt een hypothese geïdentificeerd met tuple $(slice, pos)$.

De Adaboost code is geschikt voor allerlei mogelijke soorten hypotheses dus we gebruiken de methodes *gethypothesis* en *usehypothesis* om daarover te abstraheren. Voor onze specifieke hypothesen is de functie *usehypothesis* als volgt:

```
def usehypothesis(data, hyp):
  slice, pos = hyp
  return data[slice][pos]
```

Zoals eerder vermeld is *gethypothesis* de bottleneck voor de snelheid van het programma. Deze functie zal de gewogen fout voor iedere mogelijke hypothese moeten berekenen. Door deze berekening om te schrijven in termen van gewogen optellingen van matrices in OpenCV is het dragelijk:

```
def gethypothesis(examples, label):
  min_val   = 1.0
  min_slice = None
  min_x     = None
  min_y     = None
  for slice in SLICES:
    errors = cv.CloneImage(examples[0].data[slice])
    cv.Zero(errors)
    for example in examples:
      if example.label == label:
        cv.AddWeighted(errors,                1.0,
                       example.data[slice], -1.0 * example.p,
                       1.0 * example.p,
                       errors)
      else:
        cv.AddWeighted(errors,                1.0,
```

```
17                          example.data[slice],  example.p,
18                          0.0,
19                          errors)
20      result = cv.MinMaxLoc(errors)
21      if min_val > result[0]:
22        min_val   = result[0]
23        min_slice = slice
24        min_x     = result[2][0]
25        min_y     = result[2][1]
26    return (min_slice, (min_y, min_x))
```

Dan zijn we nu klaar om onze uiteindelijke trainings procedure te definiëren. Deze creëert de datastructuur voor adaboosting en traint vervolgens een binaire classifier voor iedere emotie:

```
1  def stoppingcond(combined_hypothesis, label, examples):
2    return len(combined_hypothesis) >= AB_ITERS
3
4  def Train(dataset, filterbank):
5    classifiers = {}
6    for emotion in EMOTIONS:
7      examples = [AdaboostExample(x.slices_gabor, x.emotion, 1.0)
8                  for x in dataset.itervalues()]
9      classifiers[emotion] = AdaboostTrain(examples,
10                                          gethypothesis,
11                                          usehypothesis,
12                                          emotion,
13                                          stoppingcond)
14   return classifiers
```

En het resultaat hiervan kan gebruikt worden om te classificeren:

```
1  def Classify(facedata, classifiers):
2    max_emo = None
3    max_val = -10.0
4    for emotion in EMOTIONS:
5      result = AdaboostClassify(classifiers[emotion],
6                                usehypothesis,
7                                facedata.slices_gabor)
8      if result > max_val:
9        max_val = result
10       max_emo = emotion
11   return max_emo
```