Bachelorscriptie

Aan de hand van hersenactiviteit reconstrueren wat iemand ziet

Naam: Marco Henrix
Studentnr: 0513210
Studie: Bachelor Informatica - Radboud Universiteit Nijmegen
Begeleider: Dr. Marcel van Gerven
Datum: 31 Januari 2010

Inhoudsopgave

1 Introductie

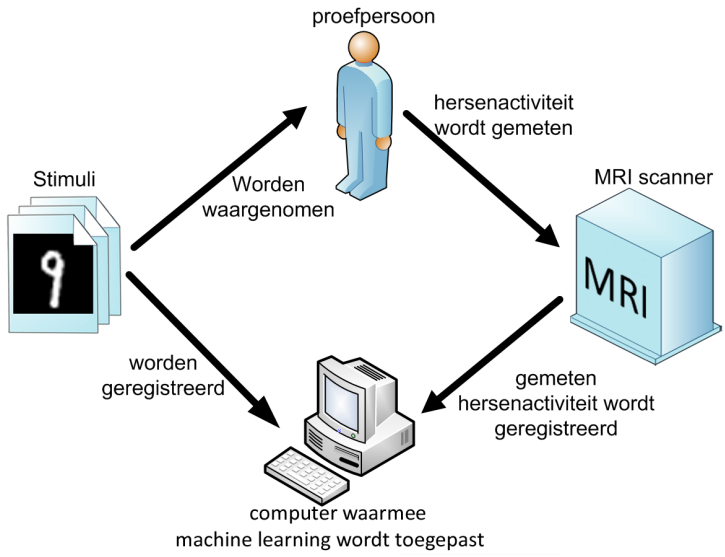
Ondanks dat er veel onderzoek wordt gedaan naar de werking van het menselijke brein lijkt het achterhalen welke informatie er exact wordt verwerkt in iemands hersenen nauwelijks haalbaar. Echter laten recente studies zien dat het wel degelijk mogelijk is om informatie uit de visuele cortex van de hersenen te decoderen. Bij deze vorm van 'brain decoding' blijkt het mogelijk te zijn om redelijk succesvol te voorspellen wat iemand ziet aan de hand van zijn of haar hersenactiviteit die gemeten wordt met behulp van een MRI scanner. Hiervoor wordt gebruik gemaakt van machine learning. Er worden meerdere stimuli getoond aan een proefpersoon en daarbij wordt de bijbehorende hersenactiviteit gemeten. Tijdens het leerproces (zie Figuur 1) zijn zowel de stimuli als de bijbehorende hersenactiviteit bekend bij het machine learning algoritme. Tijdens deze fase wordt een model geconstrueerd dat beschrijft welke eigenschappen van de gemeten hersenactiviteit bepalend zijn voor verschillen tussen de stimuli. Tijdens het reconstructieproces (zie Figuur 2) wordt een voor het algoritme onbekende stimulus getoond waarna met behulp van het geconstrueerde model voorspeld wordt welke stimulus is waargenomen aan de hand van de bijbehorende gemeten hersenactiviteit. Vervolgens kan geëvalueerd worden in welke mate de voorspelde stimulus overeenkomt met de daadwerkelijk door de proefpersoon waargenomen stimulus.

1.1 Gerelateerd werk

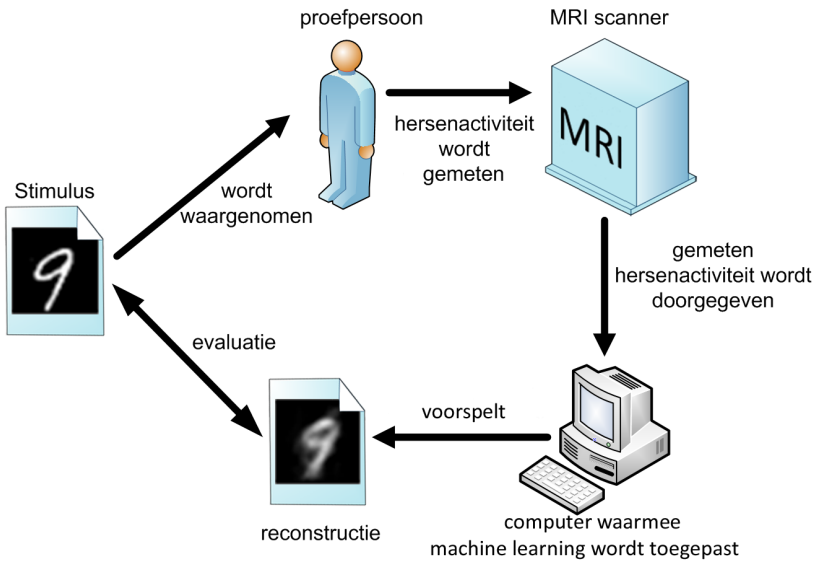
In het artikel "I can see what you see" van Kay en Gallant [7] staat een overzicht van studies waarin op soortgelijke wijze brain decoding wordt toegepast.

Eén van de eerste studies waarin deze toepassing van machine learning succesvol wordt toegepast is gepubliceerd in het artikel "Learning to decode cognitive states from brain images" van Mitchell et al. [9]. Daarbij worden classificatietechnieken toegepast om te bepalen welke soort stimulus werd waargenomen door de proefpersoon. Er werd onderscheid gemaakt in stimuli tussen afbeeldingen en tekst, tussen ambigue en niet-ambigue zinnen en tussen verschillende catagorieën woorden. Daarvoor is gebruik gemaakt van gaussian naive Bayes, k-nearest neighbour en lineair support vector machines als classifiers.

Eén van de eerste studies waarin aangetoond is dat machine learning



Figuur 1: *Schematische weergave van het leerproces van de toegepaste machine learning.*



Figuur 2: *Schematische weergave van het reconstructieproces van de toegepaste machine learning.*

niet enkel kan worden toegepast om te bepalen welke soort stimulus is waargenomen, maar ook om te reconstrueren hoe die stimulus eruitzag wordt beschreven in het artikel "Visual image reconstruction from human brain activity using a combination of multiscale local image decoders" van Miyawaki et al. [10]. In deze studie worden de stimuli, afbeeldingen bestaande uit 10×10 pixels gerepresenteerd met behulp van meerdere overlappende regio's van verschillende groottes. Met behulp van lineaire combinaties van de gemeten hersenactiviteit op verschillende plekken in de hersenen wordt de hoeveelheid contrast per regio van de reconstructieafbeelding voorspeld. De complete reconstructieafbeelding wordt vervolgens gevormd door de verschillende contrasten per regio (van verschillende groottes) samen te voegen.

Een voorbeeld van een studie waarbij geavanceerde reconstructietechnieken zijn toegepast wordt beschreven in het artikel "Bayesian reconstruction of natural images from human brain activity" van Naselaris et al. [11]. Daarbij wordt gebruik gemaakt van een Bayesiaans framework om zowel de ruimtelijke structuur als de semantische categorie van natuurlijke afbeeldingen te reconstrueren.

1.2 Probleemstelling

De onderzoeksvraag die in dit onderzoek wordt behandeld luidt als volgt: Is het mogelijk om een bestaand, relatief eenvoudig machine learning algoritme te gebruiken om aan de hand van iemands hersenactiviteit te reconstrueren wat diegene ziet.

Dit heeft zowel betrekking op het gebied van neurowetenschap waarbij onderzoek wordt gedaan naar de werking van het zenuwstelsel als op het gebied van machine learning, een vakgebied binnen de informatica en kunstmatige intelligentie waarbij algoritmes worden ontwikkeld om computers automatisch te laten leren op basis van data. Aangezien dit onderzoek wordt gedaan als invulling voor de bachelorscriptie informatica zal het zich dan ook voornamelijk richten op het gebied van machine learning.

2 Methode

2.1 Dataset

Bij dit onderzoek wordt gebruik gemaakt van een bestaande dataset die tot stand is gekomen bij het Donders Institute for Brain, Cognition and Behaviour. Daarbij is de hersenactiviteit van één proefpersoon gemeten terwijl deze verschillende afbeeldingen te zien kreeg van ofwel het cijfer zes, ofwel het cijfer negen. De stimuli werden 12.5 seconden getoond met pauzes van 12.5 seconden ertussen. De afbeeldingen zijn getoond met behulp van Psychophysics Toolbox Version 3 [4].

In totaal zijn er 111 stimuli afbeeldingen getoond die allen afkomstig zijn van de MNIST database of handwritten digits [1]. Dit zijn afbeeldingen van 28×28 pixels bestaande uit 8-bits grijstinten. In Figuur 3 zijn enkele voorbeelden getoond.

Als een bepaald gebied in de hersenen actief wordt gaat er meer zuurstofrijk bloed naartoe stromen, waarbij magnetische effecten optreden. Die magnetische effecten kunnen worden gedetecteerd met behulp van de functionele MRI scan. Bij de gebruikte dataset is dat gedaan door middel van een Siemens Trio 3 T MRI scanner. Daarbij zijn driedimensionale afbeeldingen gecreëerd die bestaan uit driedimensionale pixels (voxels) met afmetingen van $2.0 \times 2.0 \times 1.7$ mm. Vervolgens is er met behulp van het SPM5 framework [3] preprocessing toegepast op de data, voordat de dataset beschikbaar is gesteld voor dit onderzoek. Daarbij zijn onder andere bewegingen die zijn ontstaan tijdens de meting gecorrigeerd en zijn resultaten van tien tot vijftien seconden durende metingen samengevoegd. Om het aantal voxels te reduceren zijn tijdens deze preprocessing enkel de 1000 voxels geselecteerd die het meest significante verschil tonen tussen het moment dat een stimulus wordt weergegeven en de rusttoestand, waarin geen stimulus wordt weergegeven.

2.2 Preprocessing

Om tot een goede reconstructie te komen is het noodzakelijk om bewerkingen op de dataset uit te voeren voordat het classificatie- of regressie-algoritme wordt toegepast. De technieken die worden toegepast voordat de classificatie of regressie plaatsvindt staan hieronder vermeld.

Figuur 3: *Enkele voorbeelden van de 111 gebruikte stimuli, die bestaan uit verschillende handgeschreven zessen en negens.*

2.2.1 Zwart-wit conversie

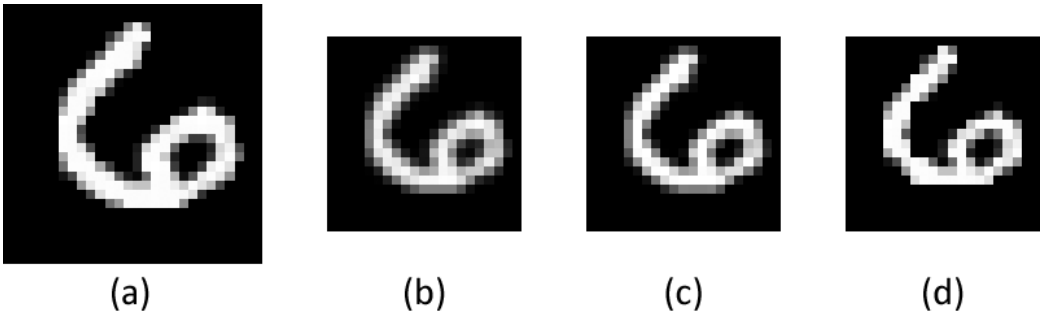
De aan de proefpersonen aangeboden stimuli zijn in de dataset gerepresenteerd als vectoren van 28×28 pixels die elk een waarde in het bereik 0-255 hebben. Echter als er gebruik wordt gemaakt van een classificatie-algoritme is het niet handig om gebruik te maken van 256 verschillende klassen. Daarom worden de stimuli omgezet naar zwart-wit afbeeldingen. Een pixel p behoort tot de klasse zwart als $p \leq \tau$ en behoort tot de klasse wit als $p > \tau$. Waarbij τ een constante thresholdwaarde in het bereik 0-255 is.

2.2.2 Afbeeldingsgrootte aanpassen

De stimuli zijn gerepresenteerd in de dataset als afbeeldingen van 28×28 pixels, echter is het onbekend hoe afbeeldingen exact worden gerepresenteerd in de hersenen. Daarom is niet duidelijk of dit ook de ideale grootte is om afbeeldingen te reconstrueren, of dat er bij andere afbeeldingsgroottes betere reconstructieresultaten te behalen zijn.

Daarnaast heeft de afbeeldingsgrootte ook invloed op de snelheid van het algoritme. Als bijvoorbeeld een afbeelding wordt gereconstrueerd van 14×14 pixels, oftewel 196 pixels in totaal, dan gaat dat ongeveer 4 keer zo snel dan het reconstrueren van een afbeelding van 28×28, oftewel 784 pixels in totaal.

Er is gekozen om gebruik te maken van bilinear interpolatie. Deze methode is standaard beschikbaar in de Image Processing Toolbox van MatLab en levert goede resultaten. De andere beschikbare methoden, bicubic in-



Figuur 4: *Verschillende interpolatiemethoden vergeleken. De originele afbeelding (a) heeft een afbeeldingsgrootte van* 28×28 *pixels. Deze afbeelding is getransformeerd naar een afbeelding met een afbeeldingsgrootte van* 21×21 *pixels met behulp van de volgende interpolatiemethoden: (b) bicubic interpolatie. (c) bilineair interpolatie. (d) nearest-neighbor interpolatie.*

terpolatie en nearest-neighbor interpolatie waren minder geschikt. Bicubic interpolatie gaf als resultaat een afbeelding met een groot aantal pixels met een waarde buiten het bereik van 0-255. Nearest-neighbor interpolatie gaf afbeeldingen van lage kwaliteit als resultaat (zie Figuur 4).

2.3 Classificatie

Om na te gaan of het mogelijk is om reconstructie uit te voeren is het verstandig om eerst na te gaan of het überhaupt mogelijk is om de data te classificeren. Dat wil zeggen dat er eerst nagegaan wordt of het mogelijk is om te bepalen of een proefpersoon ofwel een zes ofwel een negen heeft waargenomen. Classificatie is een aanzienlijk eenvoudiger probleem dan het reconstructuren van de stimuli, maar als het niet mogelijk is de data grotendeels correct te classificeren is de kans klein dat er een succesvolle reconstructie mogelijk is.

2.4 Reconstructietechnieken

2.4.1 Single-pixel reconstructie

Als eerste poging om een reconstructie te maken wordt gebruik gemaakt van single-pixel reconstructie. Dat is een van de eenvoudigste manieren om tot een reconstructie te komen. Daarbij wordt er voor elke individuele pixel uit de te reconstrueren afbeelding een aparte classifier aangemaakt en getraind

aan de hand van de trainingsdata. In het geval van 28×28 pixels, wat de originele grootte is van afbeeldingen uit de dataset, worden er dus 784 classifiers aangemaakt en getraind. Daarom kan deze methode rekenintensief zijn.

2.4.2 Principal component analysis

Principal component analysis (PCA) is een techniek die diverse toepassingen heeft, bijvoorbeeld bij gezichtsherkenning en bij beeldcompressie. Het is een veelgebruikte techniek om patronen in hoogdimensionale data te vinden. Hierbij wordt geprobeerd een nieuwe set dimensies te vinden zodat verschillen en overeenkomsten in de data duidelijker worden gemaakt. Specifiek gezegd moet de eerste dimensie zoveel mogelijk spreiding in de data beschrijven, de tweede dimensie moet orthogonaal zijn ten opzichte van de eerste dimensie en zoveel mogelijk van de overgebleven spreiding beschrijven, et cetera.

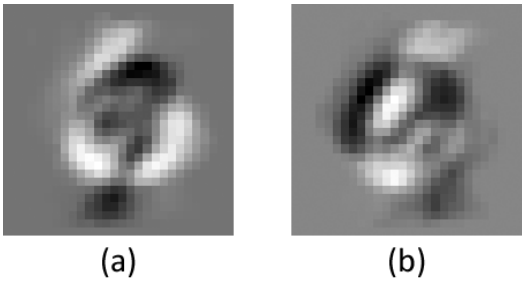
De principale componenten van de data kunnen als volgt worden berekend. Allereerst moet voor elke dimensie de gemiddelde waarde van alle datapunten nul worden. In dit geval betekent dat voor elke pixel de gemiddelde waarde van deze pixel voor alle trainingsvoorbeelden nul moet worden. Dit wordt gedaan door voor elke pixel in elk trainingsvoorbeeld de gemiddelde waarde van die pixel af te trekken.

Vervolgens kan van bovenstaand resultaat de covariantiematrix uitgerekenend worden. De covariantiematrix $\mathbf{C}$ geeft de samenhang in de data aan. Covariantie wordt altijd gemeten tussen twee dimensies, in dit geval tussen twee pixels. Daarom bevat matrix $\mathbf{C}$ de covariantie van alle mogelijke combinaties van twee pixels. Deze matrix $\mathbf{C}$ is $s \times s$ groot is waarbij s het totaal aantal dimensies, in dit geval het totaal aantal pixels, is.

Vervolgens kunnen van de covariantiematrix de eigenvectoren en eigenwaarden worden uitgerekend. Een eigenvector $\mathbf{u}$ van matrix $\mathbf{c}$ levert bij vermenigvuldiging met $\mathbf{c}$ een veelvoud van $\mathbf{c}$ op:

$$\mathbf{cu} = \lambda \mathbf{c}$$

Hierbij is λ de bijbehorende eigenwaarde. Helaas is het niet altijd mogelijk om op een efficiënte en eenvoudige wijze deze eigenvectoren exact te kunnen bepalen. Desondanks zijn er wel bruikbare algoritmen die ook in staat zijn om het antwoord te benaderen als er geen exacte oplossing gevonden kan worden. Een vereiste is dat alle eigenvectoren een lengte van 1 hebben, oftewel dat het zogenaamde unit-eigenvectoren zijn. Dit is eenvoudig te realiseren



Figuur 5: *De eerste twee gevonden principale componenten na het toepassen van PCA op alle stimuli afbeeldingen uit de dataset en vervolgens terug transformeren. Volledig zwart geeft de laagste vastgestelde waarde aan, volledig wit geeft de hoogste vastgestelde waarde aan. (a) de eerste principale component. (b) de tweede principale component.*

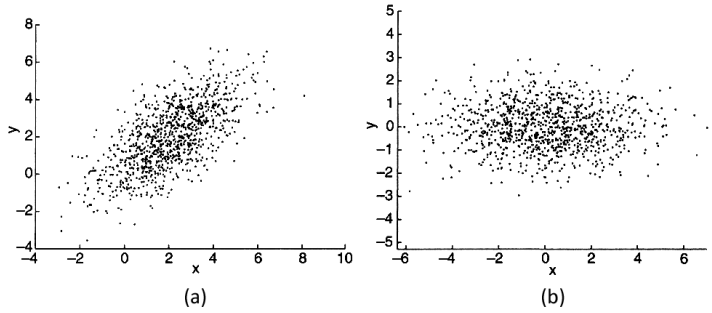
aangezien een eigenvector altijd een eigenvector blijft als deze met een scalair ongelijk aan nul wordt vermenigvuldigd.
Om de principale componenten te bepalen worden tenslotte de eigenvectoren geordend op grootte van de bijbehorende eigenwaarde. De eigenvector die hoort bij de grootste eigenwaarde is de eerste principale component, de eigenvector die hoort bij de op één na grootste eigenwaarde is de tweede principale component, et cetera.

Aangezien de principale componenten geordend zijn op de duidelijkheid van de in de data gevonden patronen kan de hoeveelheid data eenvoudig worden gereduceerd door enkel de eerste i principale componenten te gebruiken, die dus het meest belangrijk zijn en de rest te negeren. In Figuur 5 zijn de eerste twee principale componenten van alle 111 stimuli uit de dataset weergegeven. Daarin zijn duidelijk herkenbare patronen van een zes en van een negen te zien

De data kan zeer eenvoudig worden omgezet als de principale componenten bekend zijn. Dit kan gedaan worden door de datamatrix O, dat is de originele datamatrix waarbij voor elke afbeelding de gemiddelde afbeelding (zie Figuur 9) ervan af wordt getrokken, te vermenigvuldigen met de matrix $\mathbf{P}$ die de te gebruiken principale componenten bevat:

$$\mathbf{V} = \mathbf{O} \cdot \mathbf{P}$$

Hier is $\mathbf{V}$ de getransformeerde datamatrix en $\mathbf{P}$ is gelijk aan $(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_i)^T$ waarbij $\mathbf{p}_k$ de k^e principale component aangeeft. In Figuur 6 is te zien wat het effect is van het toepassen van PCA.



Figuur 6: *Voorbeeld van een set van 1000 tweedimensionale datapunten voor en na het toepassen van PCA. (a) de oorspronkelijke datapunten. (b) de datapunten na transformatie.*

Om de originele data terug te krijgen uit de getransformeerde data $\mathbf{V}$ moet de getransformeerde data vermenigvuldigd worden met de getransponeerde van matrix $\mathbf{P}$:

$$\mathbf{V} = \mathbf{O} \cdot \mathbf{P}^T$$

Daarna moet de gemiddelde afbeelding er weer bij worden opgeteld.

Om PCA te gebruiken voor het reconstrueren van afbeeldingen moeten ten eerste alle trainingsafbeeldingen worden getransformeerd met behulp van PCA. Vervolgens kan er voor elke gebruikte principale component een regressor worden aangemaakt. Daarmee kunnen voor de te reconstrueren afbeelding de gewichtsfactoren per principale component worden geschat. Dit resultaat kan tenslotte terug getransformeerd worden naar de uiteindelijke gereconstrueerde afbeelding.

2.5 Regressors en Classifiers

2.5.1 Lineaire regressie

Regressie is een machine learning techniek waarbij het in tegenstelling tot classificatie niet gaat om het voorspellen van een klasse, maar om het voorspellen van een continue waarde.

Bij lineaire regressie wordt er een lineaire relatie bepaald tussen de te voorspellen continue waarde y en de variabelen $\mathbf{x}$ waarop de voorspelling

gebaseerd is. Dit wordt gedaan aan de hand van de volgende functie:

$$f(\mathbf{x}) = w_0 + \sum_k w_k x_k$$

Om deze functie te kunnen gebruiken dienen echter wel de gewichten $\mathbf{w}$ bepaald te worden tijdens het trainen van de regressor. Een methode waarmee dat gedaan kan worden is de kleinste-kwadratenmethode. Hierbij wordt de som van de gekwadrateerde error $E = \sum_i (y_i - f(x_i))^2$ geprobeerd minimaal te maken. Hier is x_i het i^e trainingsvoorbeeld en y_i de te voorspellen variabele van het i^e trainingsvoorbeeld. Dit wordt gedaan door voor elke gewichtsfactor w_k de partiële afgeleide te nemen en deze gelijk te stellen aan nul:

$$\frac{\partial E}{\partial w_m} = -2 \sum_{i=1}^{n} (y_i - f(x)) = 0$$

Waarbij n het totaal aantal trainingsvoorbeelden is. Deze vergelijkingen kunnen worden samengevat in de zogenaamde normaalvergelijking:

$$\begin{pmatrix} n & \sum_i x_i \\ \sum_i x_i & \sum_i (x_i)^2 \end{pmatrix} \mathbf{w}^T = \begin{pmatrix} \sum_i y_i \\ \sum_i x_i y_i \end{pmatrix}$$

Het is gebruikelijk om deze normaalvergelijking compacter te noteren. Neem $\mathbf{X} = \mathbf{1}\mathbf{x}^T$, waarbij $\mathbf{1} = (1, 1, 1, \ldots)^T$.
Aangezien geldt dat:

$$\begin{pmatrix} n & \sum_i x_i \\ \sum_i x_i & \sum_i (x_i)^2 \end{pmatrix} = \begin{pmatrix} \mathbf{1}^T\mathbf{1} & \mathbf{1}^T\mathbf{x} \\ \mathbf{x}^T\mathbf{1} & \mathbf{x}^T\mathbf{x} \end{pmatrix} = \mathbf{X}^T\mathbf{X}$$

en er geldt dat:

$$\begin{pmatrix} \sum_i y_i \\ \sum_i x_i y_i \end{pmatrix} = \begin{pmatrix} \mathbf{1}^T\mathbf{y} \\ \mathbf{x}^T\mathbf{y} \end{pmatrix} = \mathbf{X}^T\mathbf{y}$$

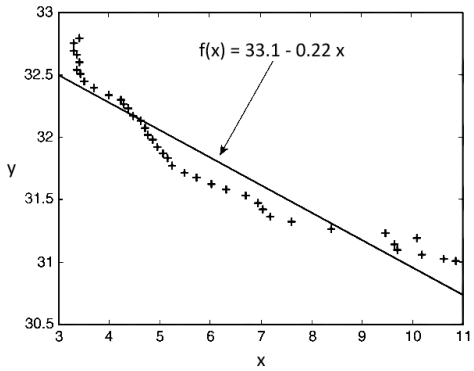
kan de normaalvergelijking worden genoteerd als:

$$\mathbf{X}^T\mathbf{X}\mathbf{w}^T = \mathbf{X}^T\mathbf{y}$$

Aan de hand van deze normaalvergelijking kan eenvoudig de gewichtsvector $\mathbf{w}$ uitgerekend worden:

$$\mathbf{w}^T = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

In Figuur 7 is een eenvoudig voorbeeld te zien van het toepassen van enkelvoudige lineaire regressie, waarbij $\mathbf{x}$ slechts uit één element bestaat.



Figuur 7: *Voorbeeld van enkelvoudige lineaire regressie. Aan de hand van de datapunten wordt met behulp van lineaire regressie de functie f geconstrueerd die de waarde y voorspelt op basis van de waarde van de variabele $\boldsymbol{x}$.*

2.5.2 Ridge regressie

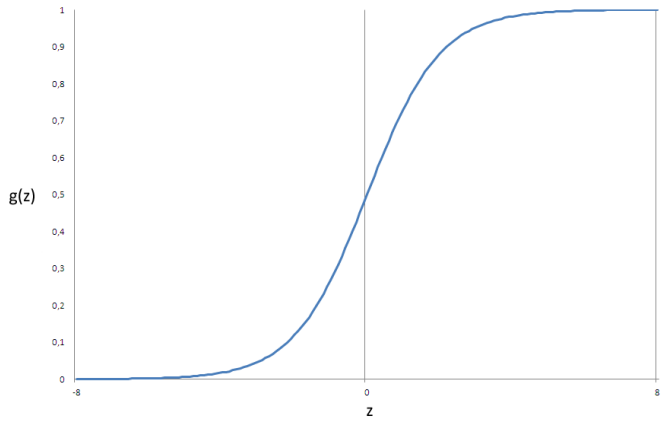
Een nadeel van lineaire regressie is dat het kan gaan overfitten op de trainingsdata. Dat houdt in dat de regressor zo getraind wordt dat de prestatie op de specifieke trainingsdata zo goed mogelijk is, maar dat het ten koste gaat van de algehele prestatie van de classifier, omdat de onderliggende patronen minder goed beschreven worden. Ridge regressie is een aangepaste vorm van lineaire regressie die overfitting probeert te voorkomen. Er wordt een extra term $\alpha\mathbf{I}$ toegevoegd aan de normaalvergelijking die kan zorgen dat de gewichten $\mathbf{w}$ kleiner worden gemaakt:

$$(\mathbf{X}^T\mathbf{X} + \alpha\mathbf{I})\mathbf{w}^T = \mathbf{X}^T\mathbf{y}$$

Hierbij is $\mathbf{I}$ de identiteitsmatrix ter groote van $n \times n$, die daarmee even groot is als $\mathbf{X}^T\mathbf{X}$ en α een scalar. Als voor $\alpha = 0$ wordt gekozen doet ridge regressie precies hetzelfde als lineaire regressie. Echter als α toeneemt zullen de gewichten w steeds kleiner worden. In de gebruikte implementatie heeft α standaard een waarde van 100.

2.5.3 Logistische regressie

Logistische regressie is een classifier gebaseerd op lineaire regressie die goed lijkt te presteren voor single-pixel reconstructie. In tegenstelling tot het eveneens goed presterende support vector machine algoritme is het bij logistische regressie wel mogelijk om bij single-pixel classificatie de kans op een bepaalde



Figuur 8: *Logistische functie* $g(z) = \frac{1}{1+e^{-z}}$.

klasse (in dit geval wit of zwart) te bepalen, waardoor het ook mogelijk is om daarbij een reconstructie te maken in grijswaarden door de waarde van een te reconstrueren pixel gelijk te stellen aan de kans dat de desbetreffende pixel wit is. De prestaties van de reconstructie gebaseerd op kansen kunnen vervolgens worden vergeleken met de binaire reconstructie (waarbij een pixel ofwel compleet wit ofwel compleet zwart is).

Bij logistische regressie wordt de kans dat een pixel wit is voorspeld aan de hand van de volgende formule:

$$P(y = 1|\mathbf{x}) = g(\mathbf{w}^T\mathbf{x})$$

Waarbij $g(z) = \frac{1}{1+e^{-z}}$. Door de toepassing van deze logistische functie valt de uitkomst van de formule altijd in het bereik 0-1, waardoor de formule geschikt is om kansen te bepalen (zie Figuur 8). Hier is y de te reconstrueren pixelwaarde; $\mathbf{x}$ is de vector die de gemeten hersenactiviteit representeert die hoort bij de te reconstrueren afbeelding; $\mathbf{w}$ is een vector die voor elke feature van de gemeten hersenactiviteit $\mathbf{x}$ een bijbehorend gewicht bevat.

Echter voordat bovenstaande formule kan worden toegepast moeten de gewichten uit vector $\mathbf{w}$ bepaald worden tijdens het trainen van de classifier. Dit wordt gedaan door de logaritmische waarschijnlijkheid L van de gewichten te maximaliseren. $L(\mathbf{w})$ is daarbij gedefinieerd als:

$$L(\mathbf{w}) = \sum_{i=1}^{n} \log g(y_i z_i)$$

Waarbij $z_i = \sum_k w_k x_{ik}$. Hier is x_{ik} de k^e feature van het i^e trainingsvoorbeeld en y_i is het label dat bij het i^e trainingsvoorbeeld hoort. Het totaal

aantal features dat hoort bij de gemeten hersenactiviteit $\mathbf{x}$ wordt aangegeven met l en het totaal aantal trainingsvoorbeelden met n.

Helaas is het niet mogelijk om altijd op een exacte manier het maximum te kunnen bepalen. Een techniek die wel altijd kan worden toegepast is gradient descent. Daarbij wordt geprobeerd iteratief de gewichten aan te passen om zo de prestaties van logistische regressie op de trainingsdata te optimaliseren. Om dat te doen worden als eerste de gewichtsfactoren willekeurig gekozen. Vervolgens kan per gewicht k de richtingscoëfficient worden bepaald aan de hand van de volgende formule:

$$\frac{\partial L}{\partial w_k} = \sum_{i=1}^{n} y_i x_{ik} g(-y_i z_i)$$

De nieuwe gewichtsfactoren worden bepaald door bij de voorgaande gewichtsfactoren een fractie (ϵ) van de richtingscoëfficient op te tellen:

$$w_k^{(t+1)} = w_k^{(t)} + \epsilon \sum_{i=1}^{n} y_i x_{ik} g(-y_i z_i^{(t)})$$

Elke keer als t toeneemt en dus bovenstaande formule herhaald wordt, dan neemt ook L toe, echter zal de toename per stap steeds kleiner worden naarmate de afstand tot het maximum kleiner wordt. Er wordt iteratief doorgegaan totdat de toename per stap onder een bepaald niveau komt.

2.6 Evaluatie

2.6.1 Afstandsmaat

In tegenstelling tot classificatie is het bij reconstructie niet triviaal om te bepalen in welke mate het resultaat correct is. Daarom moet er eerst een afstandsmaat worden gedefinieerd om te kunnen bepalen hoeveel een gereconstrueerde afbeelding verschilt van het origineel.

Er is gekozen om gebruik te maken van de city block distance, die ook wel Manhattan, taxicab of L_1 norm wordt genoemd. Deze afstandsmaat is geschikt omdat deze eenvoudig te implementeren is, eenvoudig onafhankelijk valt te maken van de resolutie van de afbeelding. Daarnaast geeft het een goede indruk in welke mate pixels correct worden geclassificeerd.

De city block distance bepaalt de afstand tussen twee vectoren $\mathbf{p}$ en $\mathbf{q}$ van gelijke grootte. Dat gebeurt door de som te nemen van alle afwijkingen per pixel:

$$d(\mathbf{p}, \mathbf{q}) = ||\mathbf{p} - \mathbf{q}||_1$$

Waarbij $||x||_1 = \sum_i |x_i|$.

Een belangrijk probleem met bovenstaande formule is echter dat deze afhankelijk is van het aantal pixels dat wordt vergeleken. Dat kan eenvoudig worden opgelost door een genormaliseerde versie te maken waarbij gedeeld wordt door het aantal pixels:

$$\bar{d}(\mathbf{p}, \mathbf{q}) = \frac{||\mathbf{p} - \mathbf{q}||_1}{s}$$

Waarbij s gelijk is aan de vectorgrootte van vector $\mathbf{p}$ en vector $\mathbf{q}$, in dit geval het totaal aantal te vergelijken pixels. Wanneer verderop in dit verslag gesproken wordt over de city block distance wordt deze genormaliseerde variant bedoeld.

2.6.2 Cross-validation

Er wordt gebruik gemaakt van cross-validation. Dit wordt gedaan door middel van een zogenaamde leave-one-out methode, van de beschikbare 111 metingen worden er 110 gebruikt voor de training van de classifier(s) en één om de prestaties van de classifier(s) te testen. Dit wordt 111 keer uitgevoerd waarbij er steeds een andere meting wordt gebruikt om te testen. Tenslotte wordt het gemiddelde genomen van de 111 testen.

2.7 Implementatie

Voor de implementatie is gebruik gemaakt van MatLab. Hiermee is het op eenvoudige wijze mogelijk om grafische bewerkingen uit te voeren op afbeeldingen, om lineaire algebra toe te passen en om machine learning algoritmes te implementeren en te gebruiken. De machine learning algoritmes die bij dit onderzoek zijn gebruikt zijn afkomstig uit de NMLT MatLab toolbox [2].

3 Resultaten

3.1 Classificatie

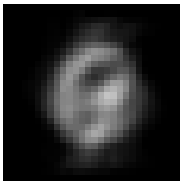
Bij de classificatie, oftewel het bepalen of er een zes of een negen is waargenomen wordt evenals bij de reconstructie ook gebruik gemaakt van leave-one-out cross-validation. Wanneer er gebruik wordt gemaakt van logistische regressie als classifier worden 85.6% van alle 111 stimuli correct geclassificeerd. Er is ook getest met andere classifiers die beschikbaar zijn in de NMLT MatLab toolbox [2], namelijk support vector machines, linear discriminant analysis en gaussian naive Bayes, maar die leveren een minder goed resultaat op, respectievelijk 84.7%, 69.4% en 68.5% van alle stimuli worden daarbij correct geclassificeerd. Desondanks is het percentage dat correct voorspeld wordt aanzienlijk hoger dan wanneer er geen gebruik zou zijn gemaakt van de gemeten hersenactiviteit. Aangezien 53 van de 111 stimuli een zes representeren en 58 van de 111 stimuli een negen representeren zouden hoogstens 52.3% van de stimuli correct te voorspellen zijn als alle stimuli als negen worden geclassificeerd.

3.2 Reconstructie

3.2.1 Baseline

Er is gebleken dat de in de dataset beschikbare stimuli grotendeels uit donkere pixels bestaan. Ondanks dat voor alle afbeeldingen geldt dat de donkerste waarde vrijwel gelijk aan nul is en de lichtste waarde vrijwel 255 is ligt de gemiddelde waarde van alle pixels in alle afbeeldingen rond de 33. Dat betekent dat als een reconstructie volledig bestaat uit witte pixels, dat de genormaliseerde city block distance gelijk is aan 0.8707. Maar het betekent ook dat als een reconstructie volledig bestaat uit zwarte pixels, bijvoorbeeld als als thresholdwaarde 255 wordt gekozen, dat de genormaliseerde city block distance gelijk is aan 0.1293.

Een baseline die wellicht meer inzicht geeft is gebaseerd op de gemiddelde afbeelding. Hierbij wordt een reconstructie gemaakt door het gemiddelde te nemen van alle trainingsafbeeldingen, het resultaat hiervan is te zien in Figuur 9. Hierbij wordt eveneens gebruik gemaakt van leave-one-out cross-validation. Dit levert een genormaliseerde city block distance op van 0.1337. Opvallend is dat deze errorwaarde hoger is dan de errorwaarde van een vol-



Figuur 9: *Gemiddelde afbeelding van alle 111 stimuli.*

ledig zwarte afbeelding. Dit lijkt op het eerste gezicht vreemd, maar met een eenvoudig voorbeeld valt in te zien dat dit wel degelijk mogelijk is. Neem als dataset de volgende vier afbeeldingen van 2×2 pixels:

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

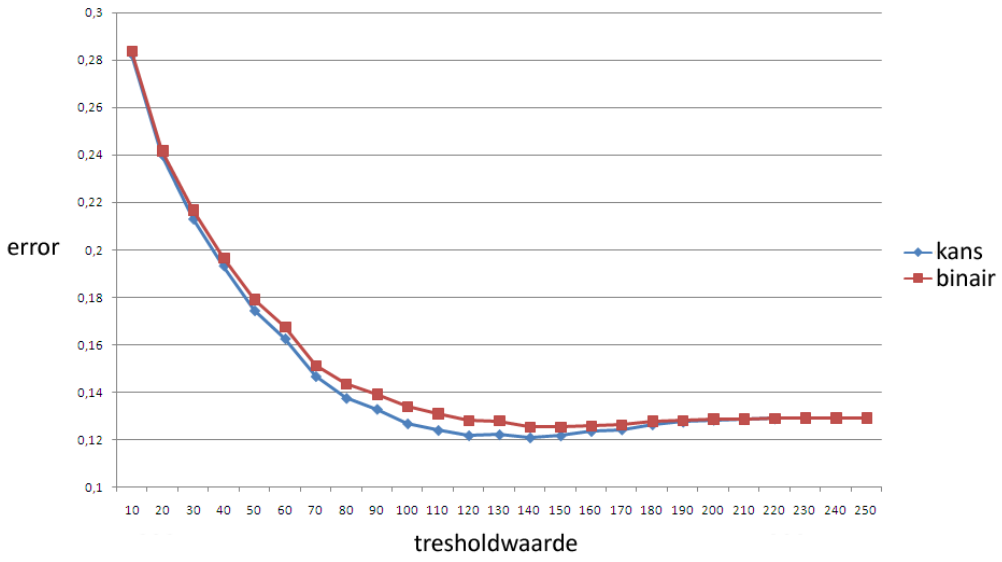
De gemiddelde afbeelding die tot stand komt met leave-one-out cross-validation heeft een city block distance van 0.5. Een volledig zwarte afbeelding heeft slechts een city block distance van 0.25.

3.2.2 Single-pixel reconstructie

Bij de reconstructie op basis van individuele pixels is gebruik gemaakt van logistische regressie als classifier. De resultaten daarvan worden hieronder besproken.

Afhankelijk van de thresholdwaarde
Om de meest geschikte thresholdwaarde te bepalen zijn reconstructies gemaakt en geëvalueerd bij thresholdwaarden variërend van 10 tot 250 (in het bereik 0-255). Dit is gedaan bij zowel een afbeeldingsgrootte van 7×7 pixels als bij een afbeeldingsgrootte van 28×28 pixels. Zodoende kan worden uitgesloten dat de keuze voor de meest geschikte thresholdwaarde zou worden gebaseerd op de resultaten bij één enkele afbeeldingsgrootte.

Figuur 11 geeft de resultaten van deze evaluatie weer. Er valt te zien dat de laagste errorwaarden van reconstructies niet veel verschillen met de errorwaarden van een volledig zwarte afbeelding als reconstructie. Desondanks is er wel te zien dat de laagste errorwaarde in het geval van 7x7 pixels hoort bij een thresholdwaarde van 140 (in het bereik 0-255). In het geval van 28x28 pixels hoort de laagste errorwaarde bij een thresholdwaarde van 110 (in het bereik 0-255). Om de keuze voor meest geschikte thresholdwaarde te



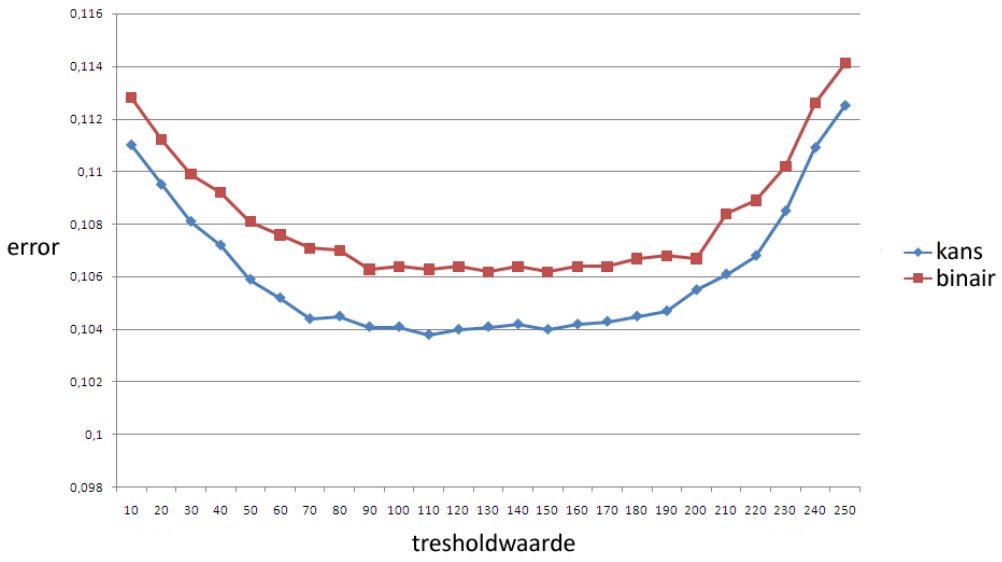
Figuur 10: *City block error bij* 7×7 *pixels bij verschillende thresholdwaarden.*

bepalen wordt gekozen om hiervan het gemiddelde te nemen, dus een thresholdwaarde van 125 (in het bereik 0-255). Dit komt overeen met een waarde van ongeveer 0.5 in het bereik 0-1.

Overigens valt ook op dat de reconstructie gebaseerd op de kans dat een pixel wit is, en die dus alle mogelijke waarden kan zijn in het bereik 0-255 in alle gevallen gelijk of beter presteert dan een binaire reconstructie, waarbij de de pixels enkel als zwart (0) of als wit (255) worden geclassificeerd.

Afhankelijk van de afbeeldingsgrootte

Om de prestaties van de reconstructies te bepalen afhankelijk van de afbeeldingsgrootte zijn reconstructies gemaakt en geëvalueerd bij afbeeldingsgrootte variërend van 2×2 pixels tot 28×28 pixels. Dat is gedaan bij een thresholdwaarde van 125. De resultaten van deze evaluatie worden weergegeven in Figuur 12. Bij erg kleine afbeeldingsgroottes, kleiner dan 4×4 pixels, is de errorwaarde relatief laag is. Echter is die lage errorwaarde niet zo interessant omdat bij die afbeeldingsgrootte zeer weinig herkenbaar is in een afbeelding. Bij alle afbeeldingsgroottes groter dan 4×4 pixels geldt dat naarmate de afbeeldingsgrootte toeneemt de errorwaarde afneemt. Mogelijk komt dit doordat een grotere afbeeldingsgrootte meer detail bevat over de vorm van de zes of de negen die de proefpersoon te zien heeft gekregen, hetgeen meer overeenkomt met de representatie van de afbeelding in de her-



Figuur 11: *City block error bij* 28×28 *pixels bij verschillende thresholdwaarden.*

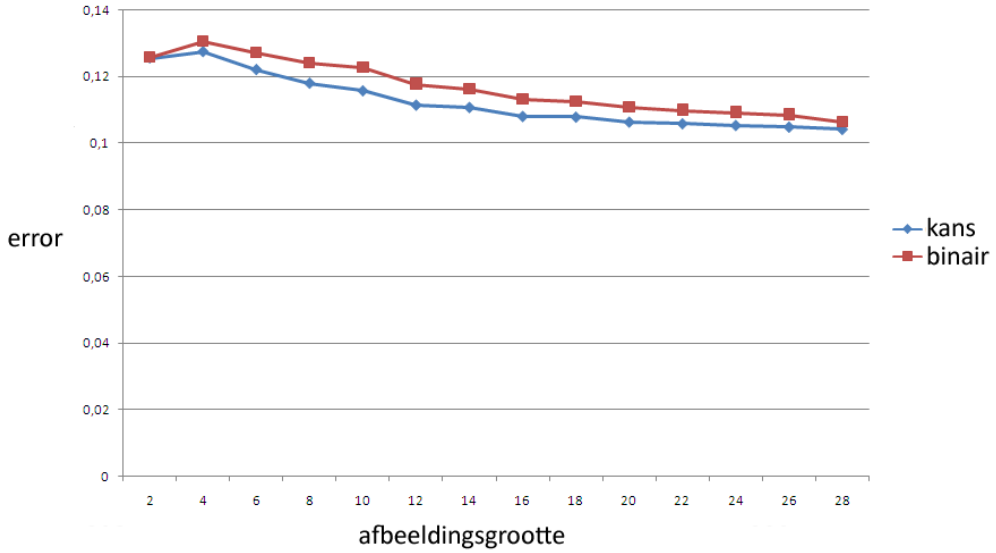
senen. Daarnaast is er bij een afbeeldingsgrootte van 28×28 geen sprake van interpolatie artefacten.

Voorbeelden

Om een indruk te geven van de prestaties van de reconstructies gemaakt met behulp van single-pixel classificatie zijn in Figuur 13 en in Figuur 14 de drie reconstructies met respectievelijk de laagste en hoogste city block error weergegeven. Zo kan worden gezien wat de prestaties kunnen zijn als een reconstructie goed geslaagd is en wat als een reconstructie minder geslaagd is.

Bekeken per pixel

In bovenstaande gevallen is enkel gekeken naar de reconstructie-error van de totale afbeelding. Er kan echter ook van elke individuele pixel nagegaan worden wat de reconstructie-error is. Dit geeft inzicht in welke delen van de afbeeldingen goed of slecht te reconstrueren zijn. Deze evaluatie is uitgevoerd bij een thresholdwaarde van 125 en een afbeeldingsgrootte van 28×28 pixels omdat dit volgens bovenstaande evaluaties een lage algehele reconstructie-error oplevert. De error per pixel is net als bij bovenstaande evaluaties berekend met behulp van leave-one-out cross-validation. Het re-

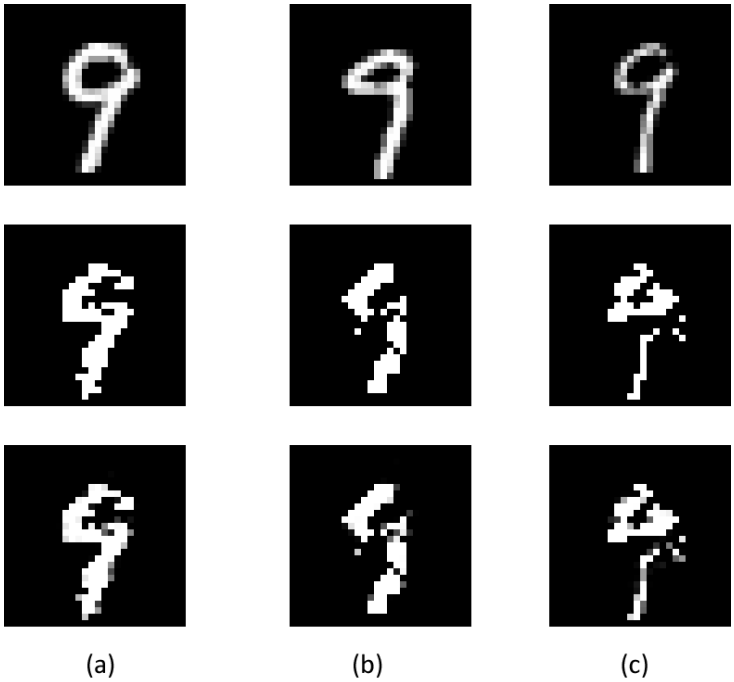


Figuur 12: *City block error (bij een thresholdwaarde van 125) bij verschillende afbeeldingsgroottes.*

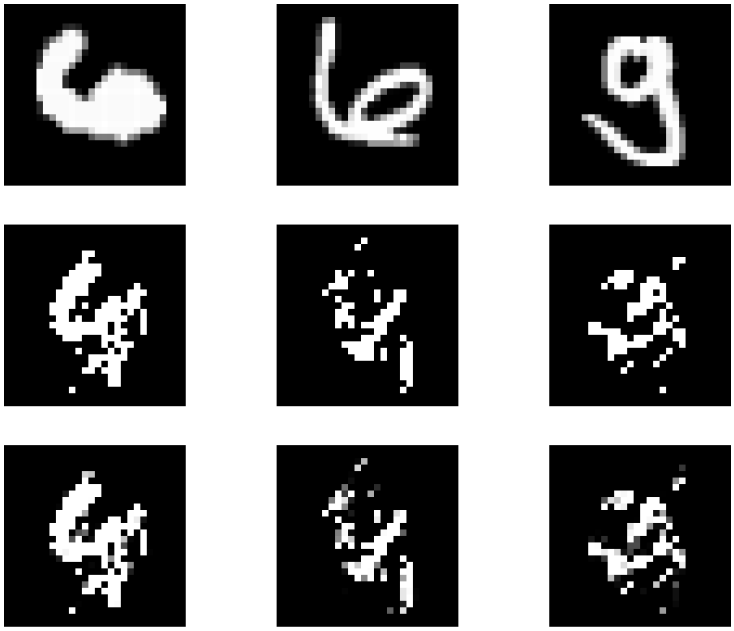
sultaat hiervan is te zien in te zien in Figuur 15a. Opvallend is dat met name in het midden van de afbeelding de reconstructie-error het grootst is. Dat is opmerkelijk, want het lijkt bijvoorbeeld niet overeen te komen met de reconstructieresultaten die in [10] te zien zijn. Om dit resultaat beter te kunnen verklaren is ook de variantie van alle 111 originele stimuli afbeeldingen per pixel uitgerekend. Het resultaat daarvan is weergegeven in Figuur 15b. Het is duidelijk te zien dat er grote gelijkenissen bestaan tussen de reconstructie-error per pixel en de variantie per pixel. Dat geeft een mogelijke verklaring waarom de reconstructie-error aan de randen lager is dan in het midden. Aan de randen is de variantie in de stimuli kleiner, want het grootste gedeelte van de randen is zwart bij alle stimuli. Daardoor is het makkelijk om de waarde van de pixels correct te voorspellen. In het midden is de variantie groter waardoor het logischerwijs minder eenvoudig is om daarbij de waarde van een pixel correct te voorspellen.

In Figuur 15c is de reconstructie-error per pixel afhankelijk van variantie in de stimuli per pixel weergegeven (waarbij nul, oftewel zwart, wordt gekozen als de variantie gelijk is aan nul). Opvallend is dat de grootste reconstructie-error te zien is aan de rand van afbeelding. Dit duidt er mogelijk op dat de randen van een stimulus minder nauwkeurig worden verwerkt in de hersenen.

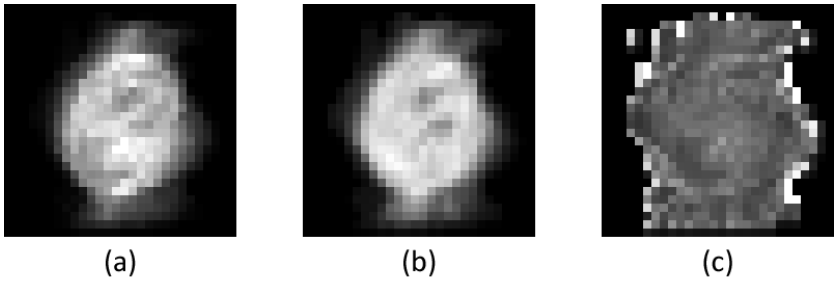
Overigens is in Figuur 15c ook te zien, weliswaar minder opvallend, dat



Figuur 13: *De drie reconstructies gemaakt met behulp van single-pixel classificatie met de laagste city block error bij een thresholdwaarde van 125 en een afbeeldingsgrootte van* 28×28 *pixels. Bovenaan worden de originele stimuli weergegeven; in het midden worden de binaire reconstructies weergegeven; onderaan worden de reconstructies gebaseerd op kansen weergegeven.*
(a) Stimulus 93 - errorwaarde binair: 0.0573 *- errorwaarde kans:* 0.0530*.*
(b) Stimulus 78 - errorwaarde binair: 0.0625 *- errorwaarde kans:* 0.0621*.*
(c) Stimulus 84 - errorwaarde binair: 0.0649 *- errorwaarde kans:* 0.0623*.*



Figuur 14: *De drie reconstructies gemaakt met behulp van single-pixel classificatie met de hoogste city block error bij een thresholdwaarde van 125 en een afbeeldingsgrootte van* 28×28 *pixels. Bovenaan worden de originele stimuli weergegeven; in het midden worden de binaire reconstructies weergegeven; onderaan worden de reconstructies gebaseerd op kansen weergegeven.*
(a) Stimulus 2 - errorwaarde binair: 0.1860 *- errorwaarde kans:* 0.1858*.*
(b) Stimulus 54 - errorwaarde binair: 0.1758 *- errorwaarde kans:* 0.1746*.*
(c) Stimulus 30 - errorwaarde binair: 0.1709 *- errorwaarde kans:* 0.1685*.*



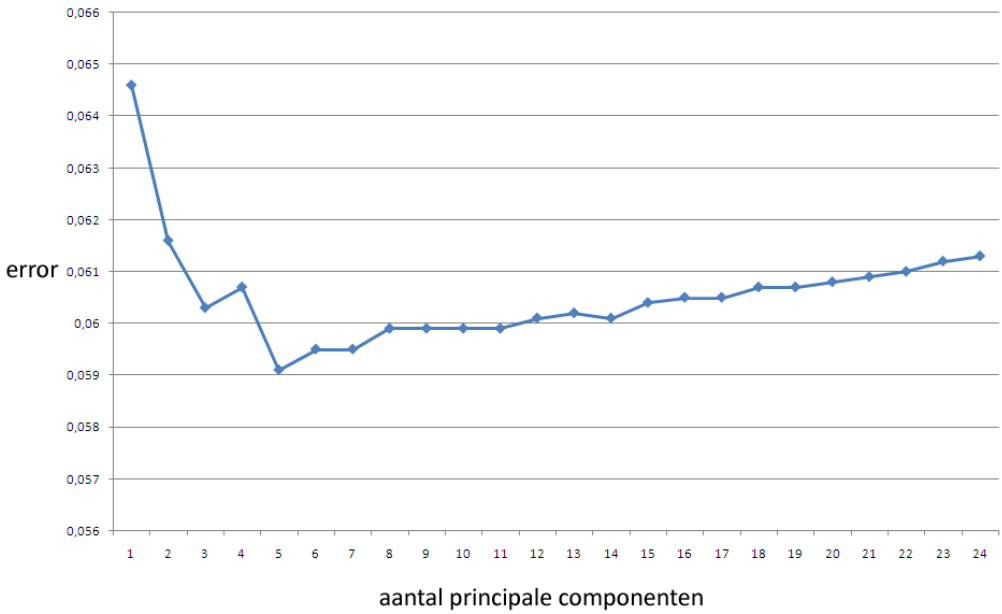
Figuur 15: *Reconstructie-error bekeken per pixel bij een afbeeldingsgrootte van* 28×28 *pixels en een thresholdwaarde van 125. Volledig zwart geeft een waarde van nul aan, volledig wit geeft de grootst vastgestelde waarde aan. (a) Reconstructie-error per pixel. (b) Variantie per pixel van alle 111 originele stimuli afbeeldingen. (c) Reconstructie-error per pixel gedeeld door de variantie per pixel (waarbij zwart wordt weergegeven als de variantie gelijk is aan nul).*

er linksboven en rechtsonder donkere rondingen zijn. Dit duidt erop dat de steel van de zes of de negen beter te reconstrueren is dan het midden van de figuur. Mogelijk wordt dit karakteristieke deel van de figuur nauwkeuriger verwerkt in de hersenen.

3.2.3 Principal component analysis

Bij de reconstructie op basis van PCA is gebruik gemaakt van ridge regressie. Er is ook geprobeerd om gebruik te maken van lineaire regressie in plaats van ridge regressie, maar daarbij waren de resultaten in alle gevallen slecht te noemen in vergelijking met ridge regressie. De resultaten waren zelfs slechter dan de baseline van een volledig zwarte afbeelding. Aangezien bij dit onderzoek geprobeerd wordt zo goed mogelijke reconstructies te maken zijn de resultaten van PCA in combinatie met lineaire regressie niet interessant om verder op in te gaan.

Afhankelijk van het aantal principale componenten
Het aantal gebruikte principale componenten is van invloed op de kwaliteit van de gereconstrueerde afbeelding. Als er te weinig principale componenten worden gebruikt kunnen niet alle details in de afbeelding correct worden gereconstrueerd. Echter naarmate het aantal gebruikte principale componenten toeneemt, die dus steeds minder duidelijke patronen representeren,



Figuur 16: *City block error bij reconstructies met behulp van PCA en ridge regressie bij een afbeeldingsgrootte van* 7×7 *pixels, afhankelijk van het aantal geselecteerde componenten.*

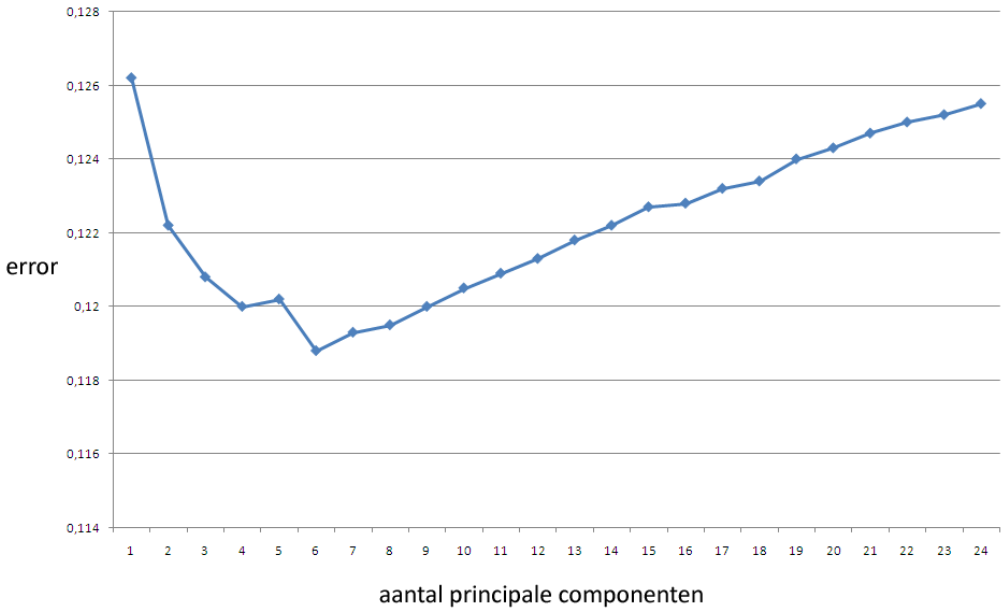
zal er ook steeds meer ruis optreden.

Om het optimale aantal principale componenten te bepalen zijn reconstructies gemaakt en geëvalueerd waarbij er gebruik gemaakt wordt van een aantal principale componenten variërend van 1 tot 24. Dit is gedaan bij zowel een afbeeldingsgrootte van 7×7 pixels als bij een afbeeldingsgrootte van 28×28 pixels. Zodoende kan worden uitgesloten dat de keuze voor het optimale aantal principale componenten zou worden gebaseerd op de resultaten bij één enkele afbeeldingsgrootte.

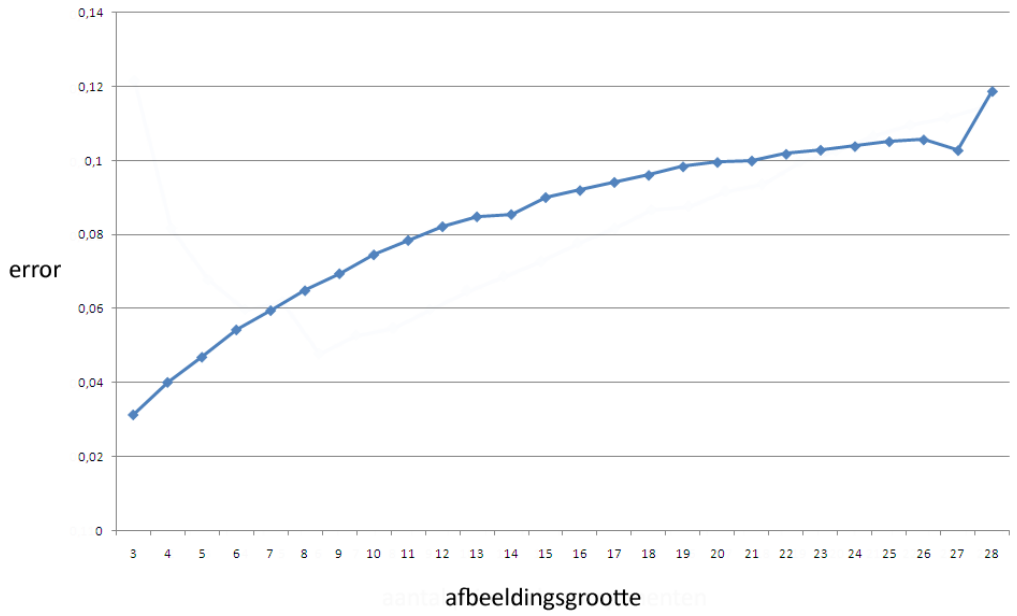
De resultaten van deze evaluatie zijn te zien in Figuur 16 en Figuur 17. Er valt te zien dat de laagste errorwaarden te zien zijn bij respectievelijk vijf en zes gebruikte principale componenten.

Afhankelijk van de afbeeldingsgrootte

Om de prestaties van de reconstructies te bepalen afhankelijk van de afbeeldingsgrootte zijn reconstructies gemaakt en geëvalueerd bij afbeeldingsgrootte variërend van 3×3 pixels tot 28×28 pixels. Daarbij is gebruik gemaakt van zes principale componenten. De resultaten van deze evaluatie



Figuur 17: *City block error bij reconstructies met behulp van PCA en ridge regressie bij een afbeeldingsgrootte van* 28×28 *pixels, afhankelijk van het aantal geselecteerde componenten.*



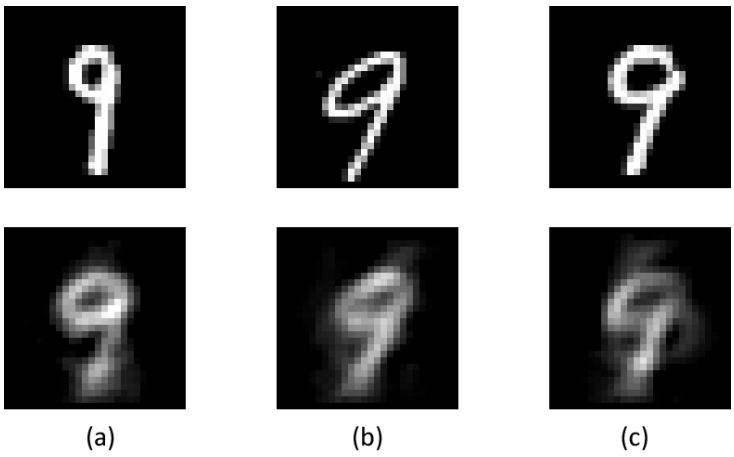
Figuur 18: *City block error afhankelijk van de afbeeldingsgroottes bij reconstructies met behulp van PCA en ridge regressie waarbij zes principale componenten zijn gebruikt.*

zijn te zien in Figuur 18.

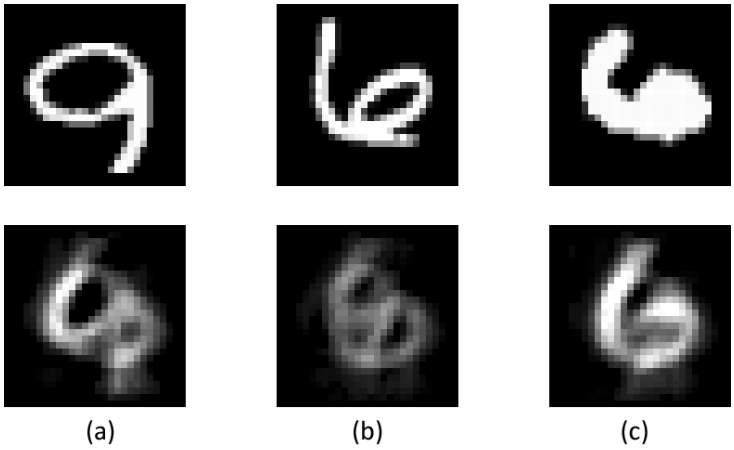
Er valt te zien dat de laagste errorwaarde te vinden is bij de kleinste afbeeldingsgrootte en naarmate dat de afbeeldingsgrootte toeneemt dat de errorwaarde ook toeneeemt. Dat is opmerkelijk, want het is tegenovergesteld ten opzichte van de resultaten bij single-pixel classificatie waarbij de errorwaarde afneemt naarmate de afbeeldingsgrootte toeneemt. Interpolatie artefacten spelen waarschijnlijk geen belangrijke rol, want bij een afbeeldingsgrootte van 28×28 pixels vindt geen interpolatie plaats, maar desondanks is de gemeten errorwaarde wel het grootst.

Voorbeelden

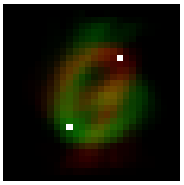
Om een indruk te geven van de prestaties van de reconstructies gemaakt met behulp van PCA zijn in Figuur 19 en 20 de drie reconstructies met respectievelijk de laagste en hoogste city block error weergegeven. Zo kan worden gezien wat de prestaties kunnen zijn als een reconstructie goed geslaagd is of als een reconstructie minder geslaagd is.



Figuur 19: *De drie reconstructies gemaakt met behulp van PCA met de laagste city block error met gebruik van zes principale componenten en een afbeeldingsgrootte van* 28×28 *pixels. Bovenaan worden de originele stimuli weergegeven, daaronder worden de reconstructies weergegeven.*
(a) Stimulus 14 - errorwaarde: 0.0573.
(b) Stimulus 60 - errorwaarde: 0.0625.
(c) Stimulus 93 - errorwaarde: 0.0649.



Figuur 20: *De drie reconstructies gemaakt met behulp van PCA met de hoogste city block error met gebruik van zes principale componenten en een afbeeldingsgrootte van* 28×28 *pixels. Bovenaan worden de originele stimuli weergegeven, daaronder worden de reconstructies weergegeven.*
(a) Stimulus 40 - errorwaarde: 0.1860.
(b) Stimulus 54 - errorwaarde: 0.1758.
(c) Stimulus 2 - errorwaarde: 0.1709.



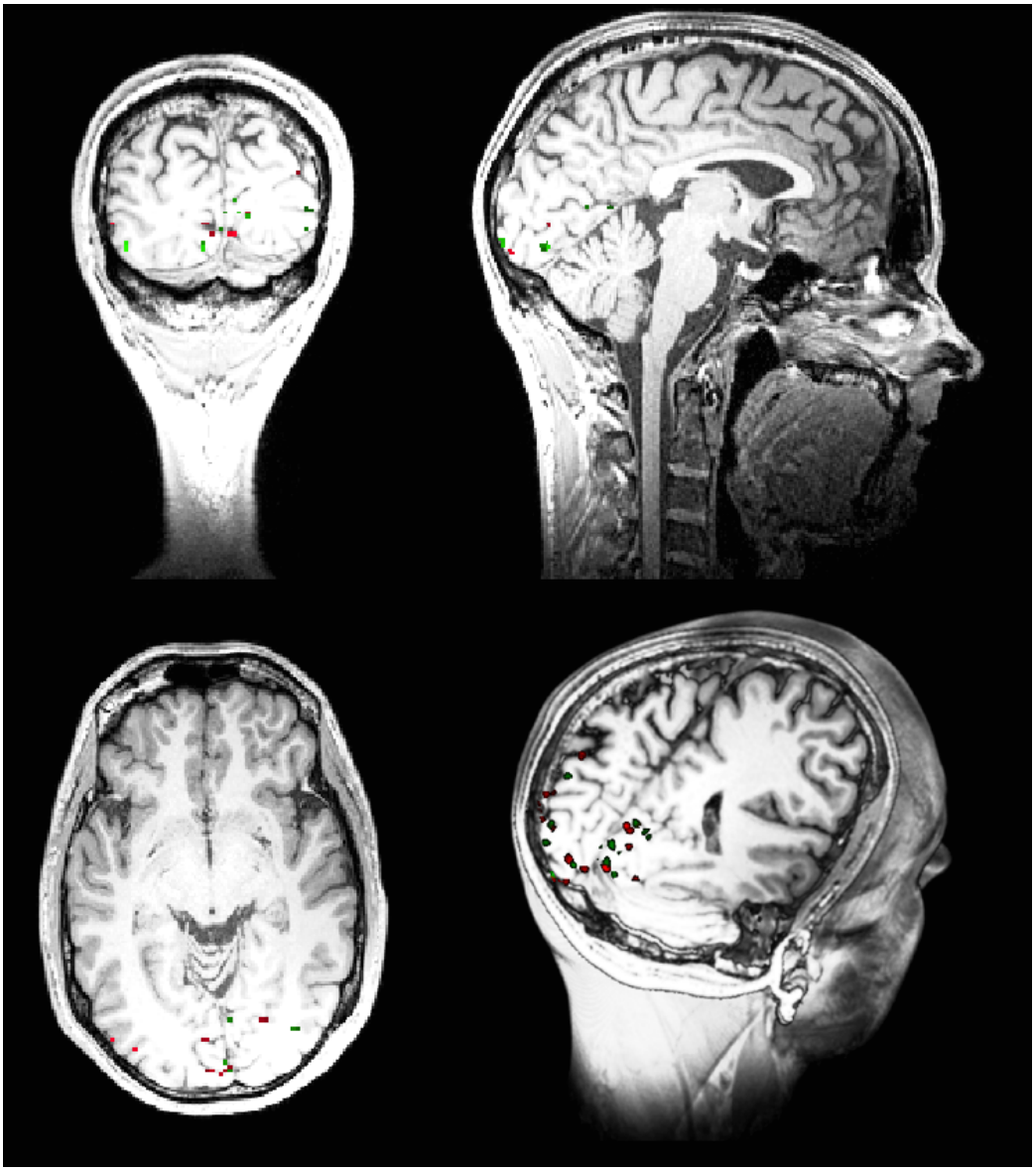
Figuur 21: *De gemiddelde afbeelding van zowel alle stimuli die het cijfer zes representeren (aangegeven met groen) als alle stimuli die het cijfer negen representeren (aangegeven met rood). Met wit zijn twee pixels aangeduid, één die vrijwel enkel voorkomt bij het cijfer zes en één die vrijwel enkel voorkomt bij het cijfer negen.*

3.3 Interpretatie

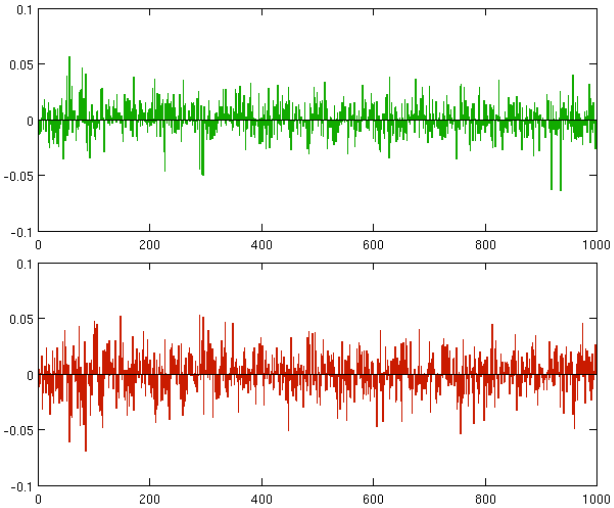
Het is niet enkel mogelijk om met machine learning reconstructies te maken, maar als machine learning is toegepast kan ook eenvoudig op basis van de geregistreerde MRI beelden een grafische weergave worden gemaakt om te kunnen zien waar de voxels die belangrijk zijn bij het maken van de reconstructies zich in de hersenen bevinden. Hierbij kan de exacte plaats waar de voxels zich in de visuele cortex bevinden worden aangegeven. Als single-pixel reconstructie is toegepast is het zelfs mogelijk om de voxels die belangrijk zijn voor de reconstructie van individuele pixels weer te geven. Een voorbeeld hiervan is te zien in Figuur 22, die gebaseerd is op de reconstructie van de twee pixels die aangegeven zijn in Figuur 21. Er is duidelijk te zien dat de voxels die van belang zijn van voor de reconstructie van een pixel die vrijwel enkel voorkomt bij het cijfer zes verschillen met de voxels die van belang zijn voor de reconstructie van een pixel die vrijwel enkel voorkomt bij het cijfer negen. Het verschil tussen welke voxels van belang zijn voor de reconstructie van de twee pixels is overigens ook weergeven aan de hand van de barplot van de beide parametervectoren zoals te zien is in Figuur 23.

4 Discussie

In dit onderzoek is nagegaan of het met bestaande relatief eenvoudige machine learning algoritmes mogelijk is om aan de hand van iemands hersenactiviteit te reconstrueren wat diegene ziet. In tegenstelling tot het onderzoek dat staat beschreven in "Visual image reconstruction from human brain activity using a combination of multiscale local image decoders" van Miyawaki et al. [10] wordt er geen gebruik gemaakt van een combinatie van verschil-



Figuur 22: *Verschillende aanzichten van de hersenen waarin de voxels die het belangrijkst zijn voor de single-pixel reconstructie op basis van logistische regressie zijn weergegeven. Met groen worden de 100 voxels weergegeven die het belangrijkst zijn voor de reconstructie van een pixel die vrijwel enkel voorkomt bij het cijfer zes (aangegeven in Figuur 21). Met rood worden de 100 voxels weergegeven die het belangrijkst zijn voor de reconstructie van een pixel die vrijwel enkel voorkomt bij het cijfer negen (aangegeven in Figuur 21).*



Figuur 23: *Barplot van twee parametervectoren na toepassen van logistische regressie. Met groen wordt de parametervector weergegeven die hoort bij de reconstructie van een pixel die vrijwel enkel voorkomt bij het cijfer zes (aangeven in Figuur 21). Met rood wordt de parametervector weergegeven die hoort bij de reconstructie van een pixel die vrijwel enkel voorkomt bij het cijfer negen (aangegeven in Figuur 21).*

lende reconstructiegroottes. Daarnaast hebben de gereconstrueerde stimuli een hogere resolutie, 28×28 pixels in plaats van 10×10 pixels. In tegenstelling tot het onderzoek dat staat beschreven in "Bayesian reconstruction of natural images from human brain activity" van Naselaris et al. [11] worden er geen natuurlijke afbeeldingen gereconstrueerd, maar enkel afbeeldingen van de handgeschreven cijfers zes en negen. Er wordt gebruik gemaakt van reconstructietechnieken die verschillen met de technieken die in de verwezen literatuur worden toegepast, maar wel relatief eenvoudig zijn. Toch blijkt uit de resultaten dat het in de meeste gevallen mogelijk is om herkenbare reconstructies te maken, zowel met behulp van single-pixel reconstructie in combinatie met logistische regressie als met PCA in combinatie met ridge regressie. Wel kan worden opgemerkt dat de reconstructies gemaakt met behulp van PCA in combinatie met ridge regressie over het algemeen van betere kwaliteit zijn dan de reconstructies gemaakt met behulp van single-pixel reconstructie in combinatie met logistische regressie.

Desondanks moet wel rekening worden gehouden dat tijdens dit onderzoek geprobeerd is de gebruikte afstandsmaat, de genormaliseerde city block distance zo laag mogelijk te houden. Deze afstandsmaat is weliswaar een-

voudig bruikbaar en is onafhankelijk van de afbeeldingsgrootte. Maar deze afstandsmaat geeft niet op alle vlakken een goede inschatting van de reconstructiekwaliteit. Zo kunnen variaties in de totale lichtintensiteit tussen de gereconstrueerde afbeelding en het origineel zorgen voor een hogere errorwaarde terwijl de gereconstrueerde vorm van de afbeelding niet wordt aangetast. Mogelijk kan in vervolgonderzoek een andere afstandsmaat worden toegepast die dit probleem kan oplossen en dus een betere indicatie is van de kwaliteit van de reconstructies. Er zou bijvoobeeld gebruik kunnen worden gemaakt van structural similarity (SSIM) zoals beschreven wordt in het artikel "Image quality assessment: From error visibility to structural similarity" van Wang et al. [15]. Deze techniek is weliswaar minder eenvoudig om toe te passen dan de city block distance, maar biedt het voordeel van gevoeligheid voor eigenschappen van het menselijke visuele systeem door te letten op structurele informatie uit de afbeeldingen. Hierdoor is deze techniek ongevoelig voor verschillen in helderheid en contrast.

In vervolgonderzoek kan mogelijk ook worden onderzocht of met behulp van andere transformaties betere reconstructies te maken zijn. Zo zou er in plaats van PCA gebruik kunnen worden gemaakt van Independent Component Analysis (ICA) zoals beschreven wordt in het artikel "Independent component analysis, a new concept?" van P. Comon [5]. Daarbij wordt de statistische onafhankelijkheid tussen de componenten geminimaliseerd.

Daarnaast zou in vervolgonderzoek mogelijk gebruik kunnen worden gemaakt van een dataset met meer voorbeelden. Nu zijn er slechts 111 voorbeelden van één proefpersoon waarin aan de hand van 1000 features de waarde van 784 pixels moet worden voorspeld. Ondanks dat er getwijfeld zou kunnen worden of op basis van dit relatief kleine aantal voorbeelden goede reconstructies te maken zijn, is er in dit onderzoek gebleken dat dat wel degelijk mogelijk is. Desondanks is het interessant om te onderzoeken of de reconstructiekwaliteit verbeterd zou kunnen worden door gebruik te maken van een dataset met meer voorbeelden.

Daarnaast is in dit onderzoek enkel gekeken naar de reconstructie van verschillende afbeeldingen in grijswaarden van het cijfer zes en negen. Dat zijn twee vormen die significant van elkaar verschillen. Zodoende is het in de meeste gevallen goed aan de reconstructie te zien welk cijfer is waargenomen. Bij vervolgonderzoek zou gekeken kunnen worden of afbeeldingen bestaande uit andere vormen of symbolen, die wellicht minder duidelijk van elkaar verschillen en afbeeldingen waarin kleuren worden gebruikt eveneens goed te reconstrueren zijn en of dat daarbij de onderlinge verschillen nog steeds in de reconstructies terug te vinden zijn.

Verder toont dit onderzoek aan dat het mogelijk is om met behulp van MRI beelden exact aan te geven welke plekken in de hersenen van belang zijn bij het maken van reconstructies. Een mogelijkheid die door neurowetenschappers gebruikt zou kunnen worden om beter in te kunnen zien waar in de hersenen bepaalde informatie verwerkt wordt.

Er kan geconcludeerd worden dat in dit onderzoek duidelijk is geworden welke methoden succesvol kunnen worden toegepast om meer inzicht te krijgen in de informatie die wordt verwerkt in de hersenen ('brain decoding'). Desondanks is er nog veel vervolgonderzoek op dit vlak mogelijk.

Referenties

[1] MNIST database of handwritten digits. `http://yann.lecun.com/exdb/mnist`.

[2] NMLT toolbox. `http://osiris.cs.ru.nl/~marcelge/code.html`.

[3] Statistical parametric mapping framework. `http://www.fil.ion.ucl.ac.uk/spm`.

[4] D.H. Brainard. The psychophysics toolbox. *Spatial vision*, 10(4):433–436, 1997.

[5] P. Comon. Independent component analysis, a new concept? *Signal processing*, 36(3):287–314, 1994.

[6] P.M.C. de Boer and C.M. Hafner. Ridge regression revisited. *Statistica Neerlandica*, 59(4):498–505, 2005.

[7] K.N. Kay and J.L. Gallant. I can see what you see. *Nature Neuroscience*, 12(3):245, 2009.

[8] T.M. Mitchell. Generative and discriminative classifiers: Naive Bayes and logistic regression. September 2006. `http://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf`.

[9] T.M. Mitchell, R. Hutchinson, R.S. Niculescu, F. Pereira, X. Wang, M. Just, and S. Newman. Learning to decode cognitive states from brain images. *Machine Learning*, 57(1):145–175, 2004.

[10] Y. Miyawaki, H. Uchida, O. Yamashita, M. Sato, Y. Morito, H.C. Tanabe, N. Sadato, and Y. Kamitani. Visual image reconstruction from

human brain activity using a combination of multiscale local image decoders. *Neuron*, 60(5):915–929, 2008.

[11] T. Naselaris, R.J. Prenger, K.N. Kay, M. Oliver, and J.L. Gallant. Bayesian reconstruction of natural images from human brain activity. *Neuron*, 63(6):902–915, 2009.

[12] J. Rennie. Logistic regression. April 2003. `http://people.csail.mit.edu/jrennie/writing/lr.pdf`.

[13] L.I. Smith. A tutorial on principal components analysis. 2002. `http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf`.

[14] P. Tan, M. Steinbach, and V. Kumar. *Introduction to data mining*. Pearson Education, 2006.

[15] Z. Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

Appendix

A Matlab code classificatie

```
load('dataset');
res = [];
for i = 1:111;
    %selecteer de trainingsdata:
    trainresponse = response;
    trainresponse(i,:) = [];
    trainlabels = labels;
    trainlabels(i,:) = [];

    %train de classifier:
    myproc = clfproc({lr}); %gebruik logistische regressie

    myproc = myproc.train(trainresponse,trainlabels);

    %test de classifier:
    post = myproc.test(response(i,:));
    if (post(1) >= post(2)) && (labels(i) == 1)
```

```
      res(i) = 1;
    elseif  (post(1) < post(2)) && (labels(i) == 2)
      res(i) = 1;
    else
      res(i) = 0;
    end
end
%toon het resultaat:
mean(res(:))
```

B Matlab code single-pixel reconstructie

```
load('dataset');
i = 23; %de te reconstrueren afbeelding
stimulisize = 28; %28x28 pixels
resizedsize = 16; %16x16 pixels
treshold = 125; tresholdwaarde van 125 in het bereik van 0-255

%resize alle stimuli afbeeldingen:
resizedpixelcount = (resizedsize)^2;
allresizedstimuli = [];
for a = 1:111
    allresizedstimuli(a,:) = reshape(imresize(reshape(stimuli(a,:),...
    stimulisize,stimulisize),[resizedsize resizedsize],'bilinear'),1,resizedpixelcount);
end

%selecteer 110 trainingsstimuli + bijbehorende response:
resizedstimuli = allresizedstimuli;
resizedstimuli(i,:) = [];

cb_bin_eval = [];
cb_eval = [];
bin_res = [];
res = [];
for p = 1:resizedpixelcount; %voor elke pixel
    %selecteer de trainingsdata:
    trainresponse = response;
    trainresponse(i,:) = [];
    trainlabels = [];
    for j = 1:110; %voor elke trainingsvoorbeeld
        %bepaal klasse:
        if (resizedstimuli(j,p) > treshold)
            trainlabels(j,:) = 2; %wit
        else
            trainlabels(j,:) = 1; %zwart
        end
```

```
    end
    %train de classifier:
    myproc = clfproc({lr}); %gebruik logistische regressie
    myproc = myproc.train(trainresponse,trainlabels);

    post = myproc.test(response(i,:));
    if (length(post) == 1) && (resizedstimuli(1,p) > treshold) %alle testpixels waren wit
        bin_res(p) = 1;
        res(p) = 1;
    elseif (length(post) == 1) %alle testpixels waren zwart
        bin_res(p) = 0;
        res(p) = 0;
    elseif (post(1) >= post(2)) %classificatie levert zwart op
        bin_res(p) = 0;
        res(p) = post(2); %geeft de kans op wit
    else %classificatie levert wit op
        bin_res(p) = 1;
        res(p) = post(2); %geeft de kans op wit
    end
end

%bereken met behulp van pdist de city block distance:
bin_res = [bin_res;allresizedstimuli(i,:)/255];
res = [res;allresizedstimuli(i,:)/255];
cb_eval = pdist(res,'cityblock')/resizedpixelcount
cb_bin_eval = pdist(bin_res,'cityblock')/resizedpixelcount
```

C Matlab code PCA reconstructie

```
load('dataset');
i = 23; %de te reconstrueren afbeelding
stimulisize = 28; %28x28 pixels
resizedsize = 16; %16x16 pixels

resizedpixelcount = (resizedsize)^2;
princompcount = 10; %aantal te gebruiken principale componenten voor de reconstructie

%resize alle stimuli afbeeldingen:
allresizedstimuli = [];
for a = 1:111
   allresizedstimuli(a,:) = reshape(imresize(reshape(stimuli(a,:),...
   stimulisize,stimulisize),[resizedsize resizedsize],'bilinear'),1,resizedpixelcount);
end


%selecteer 110 trainingsstimuli + bijbehorende response:
```

```
trainstimuli = allresizedstimuli/255;
trainstimuli(i,:) = [];
trainresponse = response;
trainresponse(i,:) = [];

%bepaal de principale omponenten van de stimuli
[pc,transformstimuli] = princomp(trainstimuli);

%selecteer de eerste kolommen (belangrijkste principale componenten):
pc = pc(:,1:princompcount);
transformstimuli = transformstimuli(:,1:princompcount);

%bereken mean:
tmean = mean(trainstimuli);

res = [];
for p = 1:princompcount; %voor elke principale component

  myproc = clfproc({ridge}); %gebruik ridge regressie
  myproc = myproc.train(trainresponse(:,1:1000),transformstimuli(:,p));

  post = myproc.test(response(i,1:1000));

  res(p) = post;

end

%transformeer het resultaat terug:
img_res = res * pc';

%tel de berekende tmean bij het resultaat op:
img_res = bsxfun(@plus,img_res,tmean);

%toon het resultaat:
imshow(reshape(img_res,resizedsize,resizedsize))

%bereken met behulp van pdist de city block distance:
img_res = [img_res;allresizedstimuli(i,:)/255];
cb_eval = mean(pdist(img_res,'cityblock')/resizedpixelcount)
```