Koko kehon PET-kuvien automaattinen segmentointi

Santeri Palonen

Pro gradu -tutkielma
Kesäkuu 2023

Tarkastajat:
Prof. Riku Klén
FT Maria Jaakkola

MATEMATIIKAN JA TILASTOTIETEEN LAITOS

TURUN YLIOPISTO
Matematiikan ja tilastotieteen laitos

Santeri Palonen: Koko kehon PET-kuvien automaattinen segmentointi
Pro gradu -tutkielma, 32 s., 5 liites.
Sovellettu matematiikka
Kesäkuu 2023

PET-kuvauksessa tuotetaan kolmeulotteinen vokselikuva ihmiskehosta. Kuvan segmentointi tarkoittaa kuvan jakamista osiin kuten päähän tai sydämeen. Manuaalisesti kuvan segmentointi on aikaa vievää, joten automatisoinnille on tarvetta. Tämä työ tutkii hill climbing- sekä flood fill -algoritmien käyttöä automaattiseen segmentaatioon. Kuvamatriisista etsitään kiinnostavat kohteet hill climbing -metodilla, koska näissä kohteissa matriisin arvot ovat on korkeammat kuin ympäristöllä. Kun nämä lokaalit maksimit on löydetty, käytetään flood fill -metodia määrittelemään segmentaatioalue. Toleranssiparametrit flood fill -metodiin etsittiin haarukoimalla. Testiaineistona käytettiin neljääkymmentä (40) rotan PET-kuvaa. Munuaisten segmentaatio onnistui vaihtelevasti ja sydämen kiitettävästi, mutta aivojen segmentaatio ei onnistunut hyvin. Segmentaatio on nopea suorittaa rottakuville, eikä tuottaisi ongelmia isommillekaan kuville. Tekniikassa on vielä paljon parannusmahdollisuuksia ja jatkokehitettävää toleranssin valintaan sekä elimien tunnistukseen liittyen.

PET-kuvantaminen, segmentaatio, automaatio, flood fill, hill climbing, Python, Numpy, Jaccard-indeksi

Sisällys

1 Johdanto

Positron emission topography (PET)-kuvantamista käytetään yleisesti ihmiskehon tutkimiseen. Yhdessä computed topography (CT) -kuvauksen kanssa sitä hyödynnetään laajalti syöpätutkimuksissa, kuten Hodginsin lymfooman sekä keuhkosyövän etsimisessä [3][4]. Lisäksi sitä voidaan käyttää kehon reaktioiden tutkimiseen. Uusien laitteiden ansiosta PET-kuvauksessa on mahdollista kuvata koko ihmiskeho pienen yksittäisen alueen sijaan[1]. Koko kehon kuvien segmentointi, eli kuvan jakaminen pienempiin kohteisiin ja näiden kohteiden tunnistaminen, vie ihmiseltä paljon aikaa. Vaarana ovat inhimilliset virheet sekä vaihtelevuus kuvien välillä. Toistuva ja yksitoikkoinen työ saattaa aiheuttaa väsymystä ja tarkkaavaisuuden heikentymistä. Nämä syyt luovat tarpeen prosessin automatisoinnille. Automaation avulla voidaan vähentää mahdollisia virheitä sekä vapauttaa ihminen mielekkäämpiin tehtäviin. Näin myös nopeutetaan työtä huomattavasti.

Kuva pitää segmentoida oikein sekä tunnistaa kyseessä olevat kehon osat. Segmentaatiossa on hyödyllistä tietää mikä elin on kyseessä. Toisaalta elimen tunnistuksessa on hyötyä siitä, että segmentaatio on hyvä. Kumpaakaan näistä tiedoista ei ole saatavilla, vaan ne pitäisi löytää ilman suuria oletuksia kuvasta.

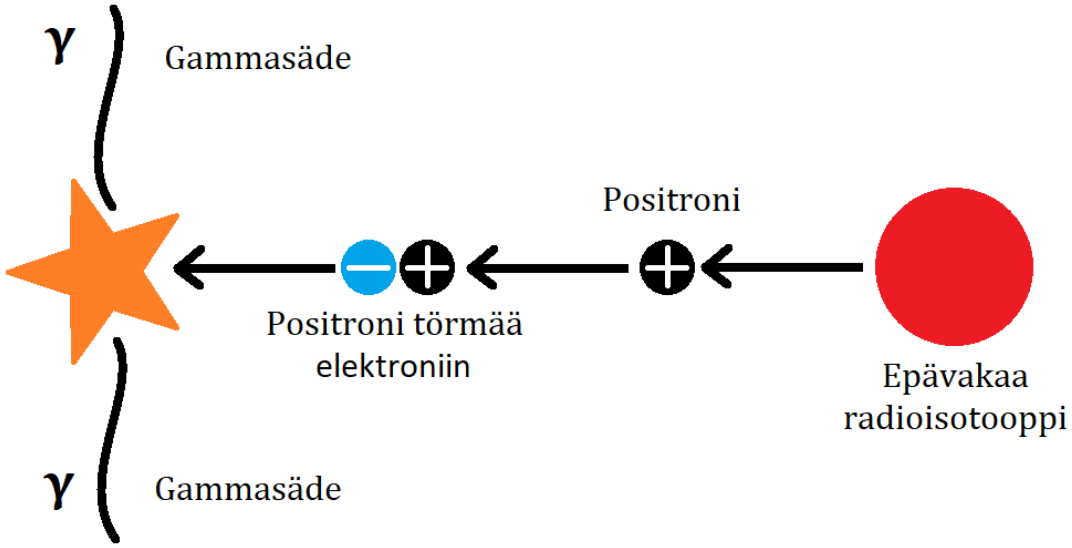
Automaattinen segmentaatio aloitetaan suodattamalla kuvasta tausta pois flood fill -metodia käyttäen. Tämän jälkeen kuvasta etsitään mahdollisia kiinnostuksen kohteita (elimiä) hill climbing -algoritmilla. Löydetyt alueet värjätään flood fill -metodilla. Näitä automaattisia segmentaatioita verrataan manuaalisesti saatuihin segmentaatioihin. Parhaat parametrit on etsitty haarukoimalla.

Työssä on käytetty ohjelmointikielenä Pythonia ja PET-kuvien tarkastelussa Carimas-ohjelmaa. Aineistona oli 40 rotan PET-kuvaa. Carimas on Turun PET-keskuksen kehittämä vokselikuvien tarkasteluun sekä mallinnusten tekemiseen tarkoitettu ohjelma[2]. Koko Python-koodin löytää Githubista[6].

2 PET-kuvaus

PET-kuvaus hyödyntää ilmiötä, jossa epävakaa radioisotooppi luovuttaa positronin liian suuren protonimäärän takia. Protoni muuttuu ytimessä neutroniksi ja vapauttaa positronin. Positroni on elektronin positiivisesti varautunut antihiukkanen. Positronin ja elektronin kohdatessaan molemmat tuhoutuvat ja kaksi 511-keV gammasädettä lähtee vastakkaisiin suuntiin tästä pisteestä (kuva 1). Kun tämä kohtaaminen tapahtuu kehossa, PET-kamera tunnistaa syntyneet gammasäteet ja laskee kohdan, josta säteet lähtivät. Radioisotooppi annetaan potilaalle yleensä injektiona.[5]

Radioisotooppi voi olla glukoosin johdannainen, jolloin aine hakeutuu elimistössä paljon glukoosia käyttäviin kudoksiin, kuten aivoihin tai syöpäkasvaimiin. Näistä kudoksista lähtee siten eniten gammasäteitä, ja ne näkyvät kolmeulotteisessa kuvassa kirkkaampina kohtina. Yksi glukoosin johdannaisista on onkologiassa paljon käytetty [18F]fluorodeoksiglukoosi (FDG). Myös tämän työn kuva-aineistossa on käytetty radioisotooppina FDG:tä.



Kuva 1: Gammasäteilyn muodostuminen. Säteet lähtevät 180^o kulmassa eri suuntiin.

2.1 PET-kuva

Kolmiulotteinen PET-kuva koostuu vokseleista, kuten tavallinen kaksiulotteinen kuva koostuu pikseleistä.

Määritelmä 1. Vokseli on kolmiulotteinen suorakulmio joillain dimensioilla ja mittayksiköillä, kuten esimerkiksi millimetreinä.

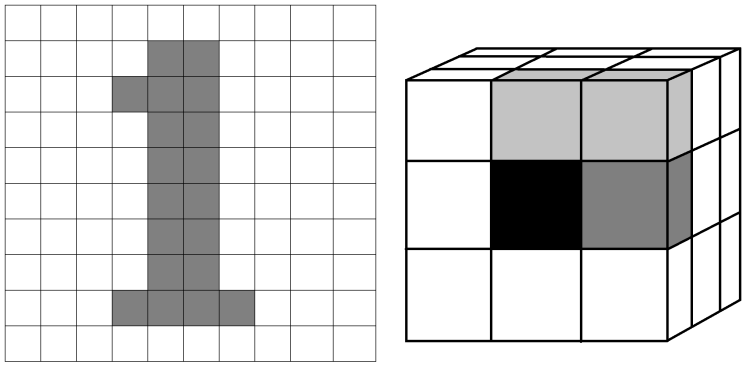
Vokselin ei tarvitse olla kuutio vaan sen dimensiot voivat olla esimerkiksi 1,65mm$\times$ 1,65mm $\times$ 3mm. Jokaisella vokselilla on väriarvo, kuten mustavalkoisella kaksiulotteisella kuvalla. Kun nämä vokselit kootaan kokonaiseksi kuvaksi saadaan aikaiseksi kolmiulotteinen kuva (kuva 2).

PET-kuva on kokoelma arvoja, jotka määräävät tietyn kohdan 'värin'. Väri ei tarkoita tässä tavanomaista väriä vaan sitä, että vokseli värjätään tietyn skaalan mukaan. Skaala taas määräytyy kuvan minimi- ja maksimiarvon mukaan. PET-kuvaa voi siis ajatella kolmiulotteisena matriisina M, jossa indeksit $i, j, k \in \mathbb{N}$ kuvaavat vokselin sijaintia kuvassa. Nämä indeksien arvot ovat välillä $[0, n]$, missä n on vokselien määrä koordinaattiakselilla. Käytetään siis seuraavaa merkintää.

Määritelmä 2. Olkoon $M \in \mathbb{R}^{x_{max} \times y_{max} \times z_{max}}$ kolmiulotteinen PET-kuva eli kolmiulotteinen kuvamatriisi, jonka dimensiot ovat $x_{max} \times y_{max} \times z_{max}$ eli jokaisen ulottuvuuden vokselimäärä. Olkoon i, j, k joitain indeksejä, joissa

- i kuvaa jotain tiettyä x-koordinaattia
- j kuvaa jotain tiettyä y-koordinaattia
- k kuvaa jotain tiettyä z-koordinaattia

$M_{i,j,k}$ tarkoittaa siis matriisin M arvoa (tai väriä) pisteessä (i, j, k).



Kuva 2: Vasen kuva on tavallinen 10×10 -mustavalkokuva. Jokainen neliö on yksi pikseli, jolla on jokin väriarvo. Tällainen kuva on kaksiulotteinen matriisi. Oikea kuva on esimerkki $3 \times 3 \times 3$ -kokoisesta mustavalkoisesta vokselikuvasta. Tämä kuva koostuu kolmiulotteisista suorakulmioista, joilla jokaisella on jokin väriarvo. Kuva on kolmiulotteinen matriisi, eli $x_{max} = y_{max} = z_{max} = 3$ ja $M^{3\times3\times3}$.

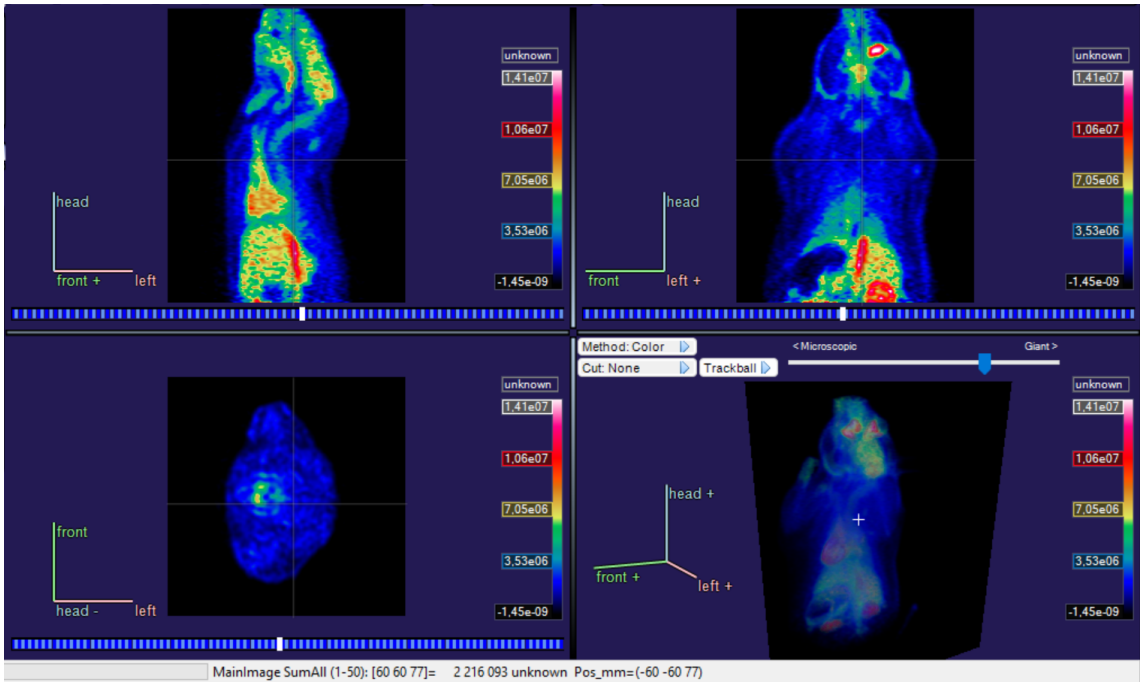
PET-kuva voi olla myös neliulotteinen. Tällainen kuva sisältää aika-akselin ja koostuu useasta kolmiulotteisesta kuvasta eri ajanhetkiltä. Kuva 3 on esimerkki kolmiulotteisesta rotan PET-kuvasta.

2.2 Manuaalinen segmentointi

Manuaalisen segmentoinnin on tehnyt Maria Rantala. Segmentointi on tehty Carimasta käyttäen[7]. Carimaksella voidaan määrittää binäärimaski, jolla merkitään volume of interest (VOI) eli kiinnostava kohde tai alue. Maski tarkoittaa, että kuvasta valitaan alue ja näiden vokselien arvot merkitään ykkösiksi ja kaikki muut nollaksi. Esimerkiksi aivojen maski tarkoittaa, että vain aivoihin kuuluvat vokselit merkitään ykkösiksi ja koko muu kuva nollaksi. Manuaaliset segmentaatiot on tehty käyttämällä viittä eri työkalua:

- Contour: Etsii kaikki vokselit, joilla on isompi arvo kuin valitulla alkuvokselilla.
- Threshold: Ottaa minimin ja maksimin käyttäjän valinnasta ja valitsee kaikki vokselit, jotka ovat näiden arvojen sisällä.
- Spread: Levittää valittua aluetta jostakin alkupisteestä käyttäjän määrittelemän matkan.
- From VOI: Lataa olemassa olevan VOI:n eli jo tehdyn maskin.
- Work Mask: Yhdistää yllämainituilla metodeilla tehdyt maskit.

Yhden elimen segmentaatio on tehty useasta eri alkupisteestä yhdistelemällä yllä lueteltuja metodeja käyttäjän haluamassa järjestyksessä.



Kuva 3: Esimerkkikuva tutkimuskohteesta. Tässä kuva on jo yhdistetty aika-akselin mukaan kolmiulotteiseksi kuvaksi. ID: Kohde 38.

3 Muita automaattisia segmentointimenetelmiä

Seuraavat menetelmät soveltuvat kaksiulotteisille kuville eli $M \in \mathbb{R}^{x_{max} \times y_{max}}$. Kuvista rajataan automaattisesti elin tai erotellaan kuvasta normaali kudos ja syöpäkudos. Esimerkiksi kasvaimien tunnistamiseen voidaan käyttää Gaussin sekamallia[12], jossa kuvan pikselit erotellaan kolmeen osaan: taustaan, epävarmaan osaan ja kohteeseen. Tämä saadaan kaavasta

$$P(z_l = c|x_l, \Psi) = \frac{\pi_c f_c(x_l|\theta_c)}{\sum_{m=1}^{C} \pi_m f_m(x_l|\theta_m)} = p_{lc},$$

missä

- z_l on vokselin l klusteri.
- x_l on vokselin l arvo.
- $\theta_c = (\mu_c, \sigma_c)$, eli parametrit todennäköisyysjakaumalle c.
- π_c on sekoitusosuus, eli painoarvo eri ryhmille. Näille on voimassa $0 \leq \pi_c \leq 1$ sekä $\sum_{c=1}^{C} \pi_c = 1$.
- $\Psi = (\pi_1, \pi_2, ...\pi_c, \Theta)$.
- $\Theta = (\theta_1, \theta_2, ...\theta_C)$.

- f_m on tiheysfunktio.
- p_{lc} on vokselin l todennäköisyys kuulua klusteriin c.

Tässä mallissa tarvitaan kuitenkin käyttäjän syötteeksi tutkittava alue. Myös alkuparametrit pitää antaa, joten ei voida puhua täydestä automaatiosta.

Maksan automaattiseen segmentaatioon kaksiulotteisista kuvista on kehitetty Active Contour Model (ACM) nimeltä Poisson Gradient Vector Flow (PGVF) yhdistettynä geneettiseen algoritmiin[13]. Tämä tekniikka etsii kuvasta muotoja sovittamalla viivaa käyttäen pikselien välisiä eroja. ACM kuvaa parametrisia käyriä pistesijaintivektorilla $\overrightarrow{X} = [x(s), y(s)]$, missä $s \in [0, 1]$. Aktiivisen ääriviivan (active contour) energia saadaan kaavasta

$$E = \int_0^1 \left\{ \frac{1}{2} \left[\alpha \left| \overrightarrow{X'}(s) \right|^2 + \beta \left| \overrightarrow{X''}(s) \right|^2 \right] + E_{ext} \right\} ds,$$

missä

- α ja β ovat painoparametreja.
- E_{ext} on ulkoinen energia.
- $\overrightarrow{X'} = d\overrightarrow{X}/ds$ ja $\overrightarrow{X''} = d^2\overrightarrow{X}/d^2s$ ovat käyrän pisteen paikkavektorin $\overrightarrow{X}$ ensimmäinen ja toinen derivaatta parametrin s suhteen.

Energia tarkoittaa matemaattista funktiota, jolla kuvataan käyrän sopivuutta reunoihin. Se saadaan pikselien värien erotuksesta. Vektorin $\overrightarrow{X}(s)$ koordinaatit ovat numeropari $[x(s), y(s)]$. Tästä voidaan ratkaista variaatiolaskennan avulla

$$\alpha x_{ss} + \beta x_{ssss} + \frac{\partial E_{ext}}{\partial s} = 0$$

sekä

$$\alpha y_{ss} + \beta y_{ssss} + \frac{\partial E_{ext}}{\partial s} = 0,$$

missä $\partial E_{ext}/\partial s$ on ulkoinen kuvan voima, x_{ss} ja y_{ss} ovat toisia derivaattoja, ja x_{ssss} ja y_{ssss} ovat neljänsiä derivaattoja x ja y koordinaateista parametrin s suhteen. Tavanomaisessa ACM:ssä ulkoinen energia on yhtä kuin intensiteettigradienttien neliö

$$E_{ext} = |\nabla M(x, y)|^2$$

Gradient Vector Flow (GVF) ACM on muokattu versio tavanomaisesta ACM-metodista reunakartalla $f(x, y) = -E_{ext}$ ja ulkoisen kuvan funktio E_{ext} on määritelty seuraavasti:

$$E_{ext} = \int \int \mu \left| \nabla^2 v \right| + |\nabla f|^2 \left| \overrightarrow{v} - \nabla f \right| dxdy$$

missä ∇^2 on Laplacen operaattori ja μ on valittu vakio. Ulkoisen kuvan voima saadaan PGVF ACM tapauksessa ratkaisemalla Poissonin yhtälö

$$\nabla^2 \phi(x, y) = f_{edge}(x, y)$$

missä f_{edge} on binäärinen reunakartta, joka löydetään Cannyn reunan tunnistuksella[14]. Kun $\phi(x, y)$ on laskettu, ulkoinen kuvan voimakenttä lasketaan vektorivirran $\overrightarrow{v} = -\nabla\phi(x, y)$ avulla. Numeerinen ratkaisu löydetään differenssimetodilla.

Myös syväoppimista on hyödynnetty laajalti kuvasegmentaatiossa. CE-Net -tekniikassa[15] kuva koodataan ensin pienempiulotteiseksi ominaisuusvektoriksi. Sen jälkeen tämä ominaisuusvektori syötetään kontekstin eristimeen (context extractoriin), joka tunnistaa kuvasta korkean tason merkityksiä kuten esimerkiksi elimen muodon. Nämä löydetyt ominaisuudet syötetään lopuksi dekooderiin, joka rekonstruoi segmentoidun kuvan tämän perusteella. Mallin alussa oleva kuvan koodauksen tekee ImageNetin[16] aineistolla koulutettu ResNet-34[17]. Tähänkin segmentaatiotyökaluun syötettävät kuvat ovat kaksiulotteisia.

4 Esikäsittely

4.1 PET-kuvan ominaisuuksia

Mikäli PET-kuva on neliulotteinen, pitää alkuperäinen kuva muuttaa kolmiulotteiseksi. Tämä tehdään summaamalla kuva aika-akselin mukaan. Otetaan siis jokaisen aikapisteen tietty x, y, z-koordinaatti, ja summataan nämä arvot yhteen yhdeksi kuvaksi.

Esimerkki 1. Jos aikapisteiden lukumäärä on 5 ja koordinaateissa (i, j, k) arvot ovat (0,9, 1,3, 1,5, 6,8, 3,1), tulee lopulliseen kolmiulotteisen kuvan koordinaatteihin (i, j, k) arvoksi 0,9 + 1,3 + 1,5 + 6,8 + 3,1 = 13,6.

Huomautus 1. Tässä työssä käytetyt PET-kuvat ovat muotoa $128 \times 128 \times 159 \times 51$. Aika-akseli pitää siis summata pois.

Vokselin väri valitaan skaalan

$$\frac{M_{i,j,k}}{M_{max}}$$

mukaan (kuva 4), missä M_{max} tarkoittaa kuvan korkeinta arvoa. Tällä on muutamia ominaisuuksia, joita tavallisella kuvalla ei ole. Koska väri määräytyy suhteen mukaan, voi arvoja liikuttaa (shift) molemmista päistä ilman, että kuva muuttuu

$$M_{i,j,k} = M_{i,j,k} + (-1 \cdot M_{min}).$$

M_{min} tarkoittaa kuvan pienintä arvoa. Tämä operaatio säilyttää kuvan samanlaisena (kunhan tietokoneen laskentatarkkuus on riittävä). Esimerkiksi, jos arvot ovat välillä [-3245, 1094094], lisäämällä +3245 molempiin arvoihin saadaan kuva siirrettyä välille [0, 1097339], mikä helpottaa käsittelyä ja on intuitiivisesti selvempi. Arvoja voidaan myös kertoa mielivaltaisesti. Jos jokainen arvo kerrotaan samalla arvolla, suhde ei muutu. Edellistä esimerkkiä jatkaen jokainen arvo voidaan kertoa luvulla $\frac{1}{1097339}$, jolloin kuvan arvot saadaan välille [0, 1] (pienin arvo 0, isoin arvo 1). Tässä työssä on suoritettu siirtäminen (shift) mutta ei kertomista, koska lukujen muuttaminen välille [0, 1] vähentää tarkkuutta.

Kuvia on monia erilaisia, ne ovat hyvin erityyppisiä, ja osassa taustakohina on voimakasta. Tämä saattaa aiheuttaa ongelmia kuvan käsittelyssä. Koska tausta on homogeeninen, se voidaan helposti poistaa soveltamalla flood fill -metodia.

Kuva 4: Väriskaala, jonka mukaan piirrettävän vokselin väri määrätään. Jos kuvan minimiarvo on 0 ja maksimi 1000 ($M_{min} = 0, M_{max} = 1000$), värjätään arvot 0 mustaksi ja arvot 1000 valkoiseksi. Arvo 500 värjäytyisi tässä tapauksessa keltaisen ja oranssin välimaastoon. Varsinaisen väriskaalan käyttäjä voi valita itse ja mahdolliset valinnat riippuvat käytettävästä ohjelmistosta. Oletus 'värjäys' on yleensä mustavalkoinen. Tässä kuvassa on käytössä Carimaksen Ocean väritys.

4.2 Flood fill

Flood fill on yhtenäisten alueiden tunnistukseen ja muokkaukseen tarkoitettu algoritmi[8] ja useat kuvanmuokkausohjelmistot käyttävät sitä muun muassa paint bucket -työkalussa, joka värittää yhtenäisen saman värisen alueen uudella värillä. Tämän työn flood fill -implementaatio on Scikit-image -kirjastosta[9]. Ohjelman optimoimaton pseudokoodi on seuraavanlainen:

1. Valitaan aloituspiste.

2. Tarkistetaan jokainen ympäröivä piste. Jos niiden arvot ovat tietyn toleranssin sisällä (ja ne eivät ole edellisiä aloituspisteitä), siirretään ne jonoon, jossa olevia pisteitä käytetään uusina aloituspisteinä.

3. Siirrytään jonon seuraavaan pisteeseen ja värjätään edellinen aloituspiste. Mikäli jono on tyhjä, on ohjelma valmis.

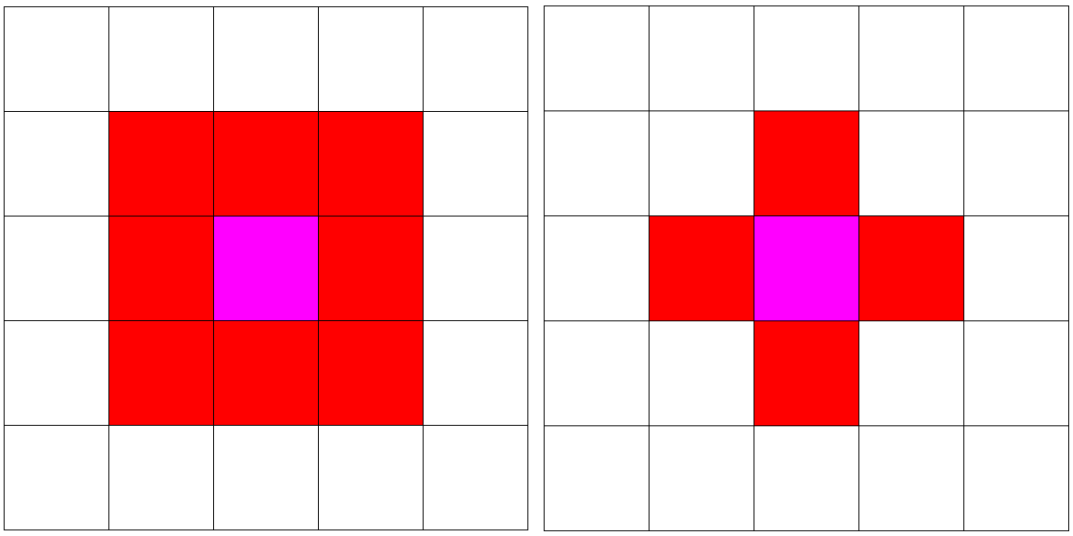
Scikit-imagen flood fill -metodiin voi myös valita *connectivity* parametrin, joka määrää, kuinka vierekkäiset vokselit määritellään naapureiksi. Vain naapurustoon kuuluvat vokselit käsitellään. Vaihtoehtoiset tavat ovat (kuva 5):

1. Naapurustoon kuuluvat kaikki vokselit, joilla on edes yksi yhteinen piste.

2. Naapurustoon kuuluvat kaikki vokselit, joilla on yksi yhteinen tahko.

Tämän työn tulokset on saatu ensimmäisellä tavalla eli naapurustoon kuuluvat kaikki vokselit, joilla on yksi yhteinen piste.

Tausta saadaan poistettua valitsemalla jokin piste, jonka tiedetään kuuluvan taustaan. Käytetään tätä flood fill -algoritmin alkupisteenä toleranssilla $0.05 \cdot M_{max}$, jolloin suurin osa taustasta voidaan muuttaa nollaksi. Tämä selkeyttää kuvaa huomattavasti. Vaihtoehtoinen tapa on muuttaa nollaksi kaikki vokselit, jotka ovat alle tietyn prosenttiosuuden maksimista.



Kuva 5: Pinkki neliö kuvastaa aloituspistettä. Sitä ympäröivä punainen alue kuvastaa tämän pisteen naapurustoa kaksiulotteisessa tapauksessa. Vasemmalla on tapaus 1 ja oikealla tapaus 2. Jos naapuruston pikselien (tai vokselien) arvot ovat toleranssin sisällä, niiden arvot muutetaan.

5 Segmentointiprosessi

Segmentointiprosessi voidaan jakaa seuraaviin osiin:

- Esikäsittely, suodatus
- Lokaalien maksimien etsiminen
- Lokaalien maksimin värjääminen

5.1 Lokaalin maksimin etsiminen 3D-kuvasta

Muokkausten jälkeen kyseessä on 3-ulotteinen matriisi, jolloin voidaan hyödyntää melko yksinkertaista hill climbing -metodia. Toiminta perustuu siihen, että tutkittavat alueet näkyvät kirkkaampina (isompina arvoina) kuvassa. Algoritmin pseudokoodi toimii seuraavasti:

1. Valitaan aloituspiste.
2. Tarkistetaan alkuarvopisteen ympäristö jollakin säteellä. Tämä tarkoittaa kuutiota alkuarvon ympäriltä.
3. Jos tästä ympäristöstä löytyy suurempi arvo kuin aloituspisteessä, valitaan tämä uudeksi aloituspisteeksi ja aloitetaan kohdasta 1. Muuten palautetaan nykyinen aloituspiste.

Yksityiskohtaisempi kuvaus ohjelman kulusta on alla sekä kokonaisena liitteessä 3. Olkoon tässä esimerkkitapauksessa (kuva 6) $M \in \mathbb{R}^{9\times 9}$ ja pikselien indeksit

$i, j \in (0, 1, ...8)$. Kuvamatriisin piste (0,0) on vasemmassa yläkulmassa. Kaikki operaatiot tapahtuvat samalla tavalla korkeammissa ulottuvuuksissa. Hakualueella tarkoitetaan osa-aluetta kuvasta, jota kohdellaan omana erillisenä pienempänä kuvana (matriisina) omilla paikallisilla koordinaateillaan.

```
import numpy as np

def HillClimbing(image_data_3d, seed_point, scope):
    # image_data_3d: 3-ulotteinen kuvamatriisi
    # seed_point:    aloituspiste
    # scope:         hakualue

    # Etsitään korkein arvo ympäristöstä.
    # Alkupiste on keskellä.
    # Jokaisen akselin suuntaan on sama hakuetäisyys.
    x_scope = scope
    y_scope = scope
    z_scope = scope

    # Tarkistetaan ollaanko reunalla, ja mikäli ollaan, pienennetään aluetta
    # sen verran että ei jouduta ulkopuolelle.
    if(seed_point[0]-scope < 0):
        x_scope = seed_point[0]

    if(seed_point[1]-scope < 0):
        y_scope = seed_point[1]

    if(seed_point[2]-scope < 0):
        z_scope = seed_point[2]


    # Etsitään maksimiarvo alkupisteen ympäriltä.
    max_value = np.max(image_data_3d[
            seed_point[0] - x_scope:seed_point[0] + scope+1,
            seed_point[1] - y_scope:seed_point[1] + scope+1,
            seed_point[2] - z_scope:seed_point[2] + scope+1
            ])
```

NumPy[20] on suosittu numeeriseen laskentaan tarkoitettu Python-kirjasto. Sen avulla on helppo käsitellä useampiulotteisia matriiseja. Itse metodin argumentit ovat kuvamatriisi *(image_ data_ 3d)*, aloituspiste *(seed_ point)* sekä hakualue *(scope)*. Aloituspiste on kolmen kokonaisluvun tripletti (i, j, k). Hakualue on kokonaisluku, jonka verran aloituspisteestä ympäristöä tutkitaan x, y ja z -akselien suhteen.

Hyödynnetään NumPyn viipalointi-ominaisuutta *(slicing)*, jolla valitaan matriisista jokin osuus tietyltä väliltä. Jokaiselle akselille voidaan valita väli, jolta kaikki arvot otetaan valintaan. Kuvan reunoilla hakualuetta täytyy pienentää niin, että indeksit eivät mene kuvan dimensioiden yli. Hakualuetta pienennetään muuttamalla

94	35	31	20	119	43	103	14	26
152	59	43	11	77	46	83	23	102
96	12	43	139	62	76	108	101	127
129	34	107	168	89	29	53	45	58
38	56	78	297	195	38	95	38	76
45	11	87	251	362	245	75	52	106
84	42	122	76	354	354	369	14	80
123	37	77	114	367	460	346	80	94
58	86	92	107	256	228	268	57	29

Kuva 6: Kuva kaksiulotteisesta tapauksesta. Numerot kuvaavat pikselin (vokselin kolmannessa ulottuvuudessa) väriarvoa näissä koordinaateissa. Aloituspiste on sininen ruutu. Parametri *scope* on tässä tapauksessa 2, koska alkupisteestä lähdetään kaksi pikseliä jokaisen akselin suuntaan. Vain tästä hakualueesta (lihavoitu neliö) etsitään uutta korkeampaa arvoa, joka on 362. Hakualueen sisäiset koordinaatit arvolle 362 ovat (4,3) ja koko kuvan koordinaatit (5,4).

ylimenevän dimension hakualue samaksi kuin alkupisteen tämän dimension koordinaatti. Tämä täytyy tehdä vain reunoilla, jotka menevät yli nollan puolelta. NumPy ei anna virhettä, jos viipalointi tehdään toiselta puolelta yli, joten siitä ei tarvitse huolehtia.

```
# Etsitään maksimiarvon koordinaatit (sekä se, onko niitä useampia).
current_hill_local_multiples = np.where(image_data_3d[
        seed_point[0] - x_scope:seed_point[0] + scope+1,
        seed_point[1] - y_scope:seed_point[1] + scope+1,
        seed_point[2] - z_scope:seed_point[2] + scope+1
        ] == max_value)
```

NumPyn *where()* funktiolla voidaan löytää matriisista tietyn ehdon täyttävien alkioiden koordinaatit. Etsitään sen avulla hakualueen korkeimman arvon koordinaatit. Funktio kohtelee hakualuetta omana erillisenä matriisina ja palauttaa koordinaatit sen lokaalissa kontekstissa. Tämä pitää muistaa ottaa huomioon myöhemmässä vaiheessa. Esimerkkikuvassa (kuva 6) palautettaisiin koordinaatit (4,3) eli maksimiarvon 362 koordinaatit hakualueen (lihavoitu neliö) suhteen.

```
# Alustetaan muuttuja johon koordinaatit tallennetaan.
current_hill_local = [0, 0, 0]

# Jos maksimiarvoja on useampia, valitaan näistä ensimmäinen joka
#tulee vastaan.
if(np.shape(current_hill_local_multiples)[1] >= 2):
    for i in range(scope):
        for j in range(scope):
            for k in range(scope):
                if(image_data_3d[seed_point[0]-scope + i]
                [seed_point[1]-scope + j]
                [seed_point[2]-scope + k]
                == max_value):
                    # Maksimi koordinaatit lokaalissa kontekstissa.
                    # Asetetaan uuden korkeimman
                    # kohdan koordinaatit tässä
                    # mikäli maksimeita oli useampi
                    current_hill_local[0] = i
                    current_hill_local[1] = j
                    current_hill_local[2] = k
                    break
                else:
                    continue
            break
```

Tarkistetaan, että löytyikö useampia identtisiä maksimiarvoja. Tämä on epätodennäköistä, mutta epätoivottujen tuloksien välttämiseksi se on hyvä tarkistaa. Mikäli useampia maksimiarvoja löytyy, valitaan niistä listan ensimmäinen ja jätetään loput huomiotta.

```
# Usempia arvoja ei löytynyt, asetetaan lokaalissa
# kontekstissa koordinaatit (pienemmässä scope x scope x scope
# kuutiossa olevat).
else:
    current_hill_local[0] = current_hill_local_multiples[0][0]
    current_hill_local[1] = current_hill_local_multiples[1][0]
    current_hill_local[2] = current_hill_local_multiples[2][0]

# Lasketaan globaalit koordinaatit lokaalien koordinaattien
# sekä hakuhaarukka parametrin avulla.
# Tämä saadaan summaamalla
# (alkupiste - hakuhaarukka) + lokaalikoordinaatti
current_hill_x = seed_point[0]-x_scope + current_hill_local[0]
current_hill_y = seed_point[1]-y_scope + current_hill_local[1]
current_hill_z = seed_point[2]-z_scope + current_hill_local[2]
```

Lasketaan hakualueen korkeimman arvon koordinaatit koko kuvan suhteen. Tämä tehdään kaavalla

$$a_{globaali} = a_{alkupiste} - a_{hakualue} + a_{lokaalimaksimi},$$

missä a on kokonaislukukoordinaatti joltakin akselilta. Esimerkkikuvan (kuva 6) tapauksessa uusi maksimi löytyy hakualueen (lihavoitu neliö) suhteen koordinaateista (4,3). Tämän lokaalin maksimin sijainti koko kuvasta saadaan laskemalla

$$x_{globaali} = 3 - 2 + 3 = 4$$
$$y_{globaali} = 3 - 2 + 4 = 5.$$

Globaalit koordinaatit ovat (4,5), mikä voidaan myös vahvistaa kuvasta (kuva 6).

```
# Tarkistetaan oliko käytetty aloituspiste myös
# lokaalimaksimi scope x scope x scope kuution sisällä.
# Mikäli oli, on ohjelman suoritus valmis ja
# palautetaan nämä koordinaatit.
if(image_data_3d[seed_point[0]][seed_point[1]][seed_point[2]]
   == max_value):
    return max_value, current_hill_x, current_hill_y, current_hill_z

# Jos näin ei ollut, toistetaan HillClimbing uudelle
# korkeimman arvon omaavalle vokselille.
temp_tuple = (current_hill_x, current_hill_y, current_hill_z)

# Rekursio, annetaan löydetty maksimi funktiolle
# uudeksi aloituspisteeksi
return HillClimbing(image_data_3d, temp_tuple, scope)
```

Nyt tarkistetaan, onko aloituspiste sama kuin hakualueen suurimman arvon omaava piste. Mikäli näin on, palautetaan nämä koordinaatit, sillä ohjelma ei voi jatkaa. Jos taas löydettiin alkupisteen arvoa korkeampi arvo, funktio kutsuu itseään rekursiivisesti antamalla uusimman korkeimman arvon pisteen koordinaatit syötteeksi (kuva 7).

Jokaisesta vokselista päädytään aina johonkin lokaaliin maksimiin, koska kuvan dimensiot sekä arvot ovat rajoitettuja. Loputtomia silmukoita ei siis synny. Jotta jokainen lokaali maksimi varmasti löytyy kuvasta, varminta olisi toistaa hill climbing jokaiselle vokselille. Tämä on ongelmallista laskentatarpeen kannalta. Kohteen ulkopuolinen alue ei ole kiinnostuksen kohteena. Taustaa on myös huomattavasti enemmän kuin kuvattavaa kohdetta. Laskentatehoa käytetään turhaan, mikä vie resursseja ja aikaa. Toisaalta mikäli tausta on suodattamaton ja myös nämä pisteet käydään läpi, löytyy taustastakin väjäämättä lokaaleja maksimeja. Näillä maksimeilla ei ole merkitystä ja ne tulisi jättää huomiotta.

94	35	31	20	119	43	103	14	26
152	59	43	11	77	46	83	23	102
96	12	43	139	62	76	108	101	127
129	34	107	168	89	29	53	45	58
38	56	78	297	195	38	95	38	76
45	11	87	251	362	245	75	52	106
84	42	122	76	354	354	369	14	80
123	37	77	114	367	460	346	80	94
58	86	92	107	256	228	268	57	29

94	35	31	20	119	43	103	14	26
152	59	43	11	77	46	83	23	102
96	12	43	139	62	76	108	101	127
129	34	107	168	89	29	53	45	58
38	56	78	297	195	38	95	38	76
45	11	87	251	362	245	75	52	106
84	42	122	76	354	354	369	14	80
123	37	77	114	367	460	346	80	94
58	86	92	107	256	228	268	57	29

Kuva 7: Seuraavat kierrokset. Ohjelman suoritus päättyy, kun hakualueelta ei löydetä enää korkeampia arvoja.

5.1.1 Suodatus

Seuraavaksi vähennetään tarvittavien aloituspisteiden määrää pudottamalla kaikki pisteet, jotka ovat tietyn kynnyksen alapuolella. Kuvan taustan poistaminen on suoritettu esikäsittelyssä. Koska tausta on poistettu eli muutettu arvoksi 0, riittää, että suodatetaan kaikki arvot, jotka ovat alle 1% maksimista. Mikäli $M_{i,j,k} < 0{,}01 \cdot M_{max}$, piste sivuutetaan ja mennään seuraavaan. Käytännössä tämä toteutetaan niin, että otetaan lista kaikista pisteistä, joiden arvo on yli $0{,}01 \cdot M_{max}$, minkä jälkeen suoritetaan hill climbing jokaiselle näistä pisteistä. Tässä työssä suodatus on toteutettu poistamalla vain tausta flood fill -metodin avulla, käyttäen toleranssiparametrina $0.05 \cdot M_{max}$ sekä sen jälkeen alkupisteiden suodatuksessa arvoa $0.01 \cdot M_{max}$.

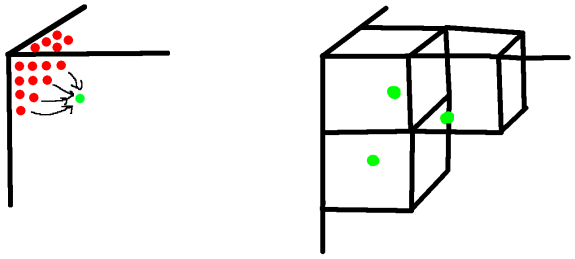
Huomautus 2. Suodatus voitaisiin toteuttaa myös niin, että taustaa ei tarvitse esikäsitellä. Voidaan tehdä esimerkiksi siten, että valitaan suodatusparametriksi $0{,}10 \cdot M_{max}$, jolloin tausta melko varmasti putoaa automaattisesti pois. Toisaalta tällöin jätetään huomiotta myös kuvattavan kohteen sisällä olevat pisteet, jotka ovat alle tämän arvon.

5.1.2 Hill climbing -algoritmin hakuhaarukka

Isommalla hakuhaarukalla mennään suuremmilla harppauksilla lokaaliin maksimiin. Tämä taas saattaa pudottaa tietyt lokaalit maksimit kokonaan pois. Vaikka oltaisiinkin yhdessä lokaalissa maksimissa ja hakualue on hyvin iso, tästä saatetaan vielä päästä uuteen korkeamman arvon omaavaan lokaaliin maksimiin. Tämä ei muodostu ongelmaksi, sillä elimet ovat suhteellisen isoja ja riittää, että niistä löydetään vain yksi arvoltaan suurin lokaali maksimi. Se on jopa toivottavaa, sillä se helpottaa myöhemmin elimen värjäystä.

5.1.3 Laiska haku

Prosessia voidaan vieläkin nopeuttaa jakamalla kuva osiin ja ottamalla näistä osista maksimiarvo. Tätä maksimiarvoa käytetään sitten alkuarvona hill climbing -algoritmille. Idea on, että likimääräisesti kaikki saman alueen pisteet kulkevat ohjelman suorituksen aikana yhden saman pisteen kautta. Koska ohjelma on deterministinen ja tästä yhdestä pisteestä päästään joka tapauksessa aina lokaaliin maksimiin, on turha laskea jokaiselle vokselille erikseen reittiä lokaaliin maksimiin, kun pelkkä lopputulos on merkityksellinen (kuva 8). Nopeushyöty on todella merkittävä, sillä tutkittavien vokselien määrä pienenee kolmessa ulottuvuudessa.



Kuva 8: Laiskan haun demonstraatio. Kuution särmän pituus on d.

5.1.4 Teoreettinen nopeushyöty laiskalla haulla

Jokaisen vokselin läpikäyminen tarkoittaa, että algoritmi on ajettava vähintään $x_{max} \cdot y_{max} \cdot z_{max}$ kertaa, missä $x_{max}, y_{max}, z_{max}$ ovat kuvan dimensiot vokseleissa. Kun mahdollisten alkupisteiden määrä supistetaan, ajokertoja jää jäljelle

$$\left\lceil \frac{x_{max}}{d} \right\rceil \cdot \left\lceil \frac{y_{max}}{d} \right\rceil \cdot \left\lceil \frac{z_{max}}{d} \right\rceil,$$

missä d on laiskan haun hakukuution särmän pituus vokseleissa. Mikäli jako ei mene tasan, viimeinen siivu voi olla vajaa. Testiaineiston kuvien dimensiot ovat $x_{max} = 128, y_{max} = 128, z_{max} = 159$. Jokaisen vokselin läpikäyminen vaatisi hill climbing -algoritmin ajamista $128 \cdot 128 \cdot 159 = 2605056$ kertaa. Tässä työssä hakusiivun parametri $d = 8$. Tällä arvolla mahdollisten alkuvokselien määrä putoaa dramaattisesti ja lopullinen määrä on

$$\left\lceil \frac{128}{8} \right\rceil \cdot \left\lceil \frac{128}{8} \right\rceil \cdot \left\lceil \frac{159}{8} \right\rceil = 16 \cdot 16 \cdot 20 = 5120.$$

Tämä on ≈ 509 kertaa pienempi kuin kaikkien vokselien määrä. Parhaimmillaan, jos jako menee tasan, nopeushyöty on teoriassa

$$\frac{x_{max} \cdot y_{max} \cdot z_{max}}{d^3}.$$

On tosin huomioitava, että liian iso hakusiivu saattaa aiheuttaa sen, että kaikkia lokaaleja maksimiarvoja ei välttämättä löydetä. Tällä on hyöty- sekä haittapuolia. Se suodattaa turhia lokaaleja maksimeja pois, mutta jos elimet ovat lähekkäin, vain suuremman arvon omaava löytyy.

5.1.5 Suoritusaika

Suoritusaikojen testauksessa on käytetty samaa tietokonetta, Lenovo ThinkPad L14 Gen 2 (AMD). Tämän konemallin prosessori on AMD Ryzen 5 PRO 5650U 2,30 GHz kellotaajuudella, näytönohjain on AMD Radeon Graphics integroitu grafiikkasuoritin ja välimuisti on 16,0 Gt:n suuruinen. Kyseessä on keskiverto kannettava tietokone. Suoritusaika ilman mitään yllä mainittuja optimointikeinoja on melko hidas. Jokainen vokseli käydään läpi niin, että hakuhaarukka alkuarvon ympärillä on 5 vokselia jokaiseen suuntaan. Koko $128 \times 128 \times 159$ vokselin kokoisen kuvan läpikäymiseen menee useita tunteja. Lisäksi kuva on täynnä turhia lokaaleja maksimeja taustassa sekä kuvattavassa kohteessa (kuva 11). Pudottamalla alkupisteitä saadaan huomattava hyöty. Kaikkia ylläolevia keinoja käyttämällä suoritusnopeus voidaan pudottaa alle sekuntiin. Suoritusajat eri parametreilla löytyvät taulukoista 1, 2, 3 ja 4 sekä kuvista 9 ja 10.

Huomautus 3. Suoritusajat riippuvat tietokoneesta sekä käynnissä olevista taustaohjelmista. Yllä olevat suoritusajat on otettu samalla tietokoneella, ilman päällä olevia taustaohjelmia ja samoissa olosuhteissa. Tulokset ovat keskiarvoja kymmenestä eri ajokerrasta, poikkeuksena optimointia käyttämätön ajo. Tällöin on tehty vain yksi xy-siivu kuvasta ja kerrottu tämä z-akselin pituudella.

	Suoritusaika (s)
Hakuhaarukka 5 (11^3)	33,10
Hakuhaarukka 8 (17^3)	35,34

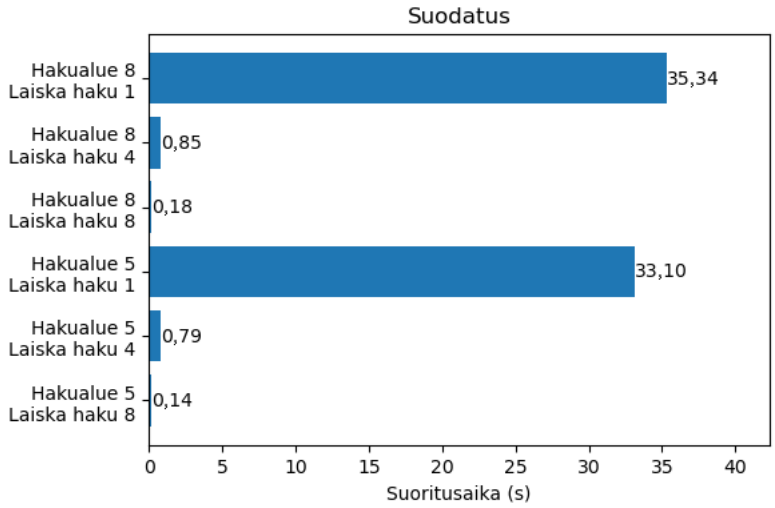
Taulukko 1: Suodatus, laiska haku 1 (1). Suluissa olevat arvot ovat hakukuutioiden vokselien määrät.

	Suoritusaika (s)
Laiska haku 4 (64)	0,79
Laiska haku 8 (512)	0,14

Taulukko 2: Suodatus, hakuhaarukka 5 (11^3).

	Suoritusaika (s)
Laiska haku 4 (64)	0,85
Laiska haku 8 (512)	0,18

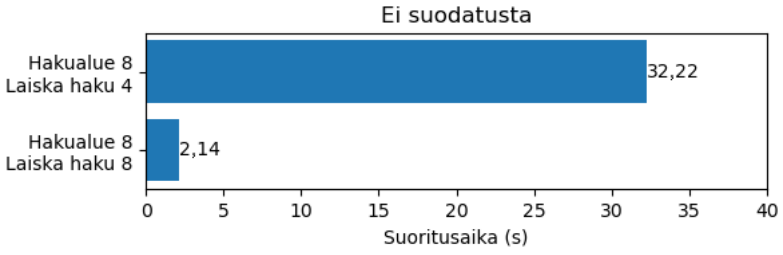
Taulukko 3: Suodatus, hakuhaarukka 8 (17^3).



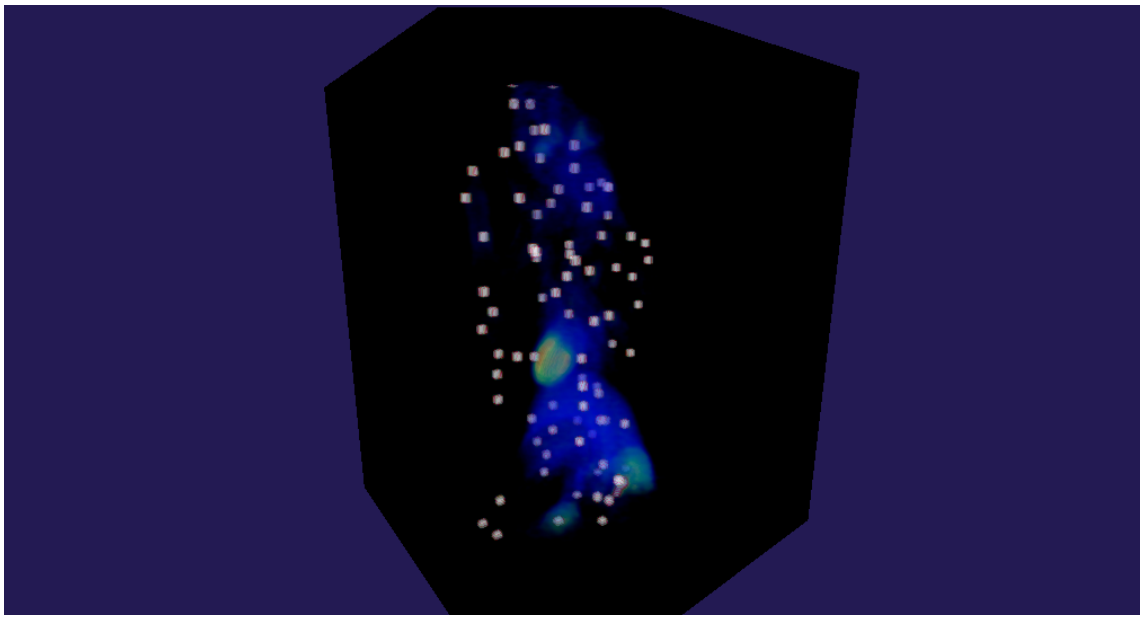
Kuva 9: Suoritusajat, kun kuvaa suodatetaan ensin.

	Suoritusaika (s)
Laiska haku 1 (1)	8478,71
	($\sim 66{,}24 \cdot 128$)
Laiska haku 4 (64)	32,22
Laiska haku 8 (512)	2,14

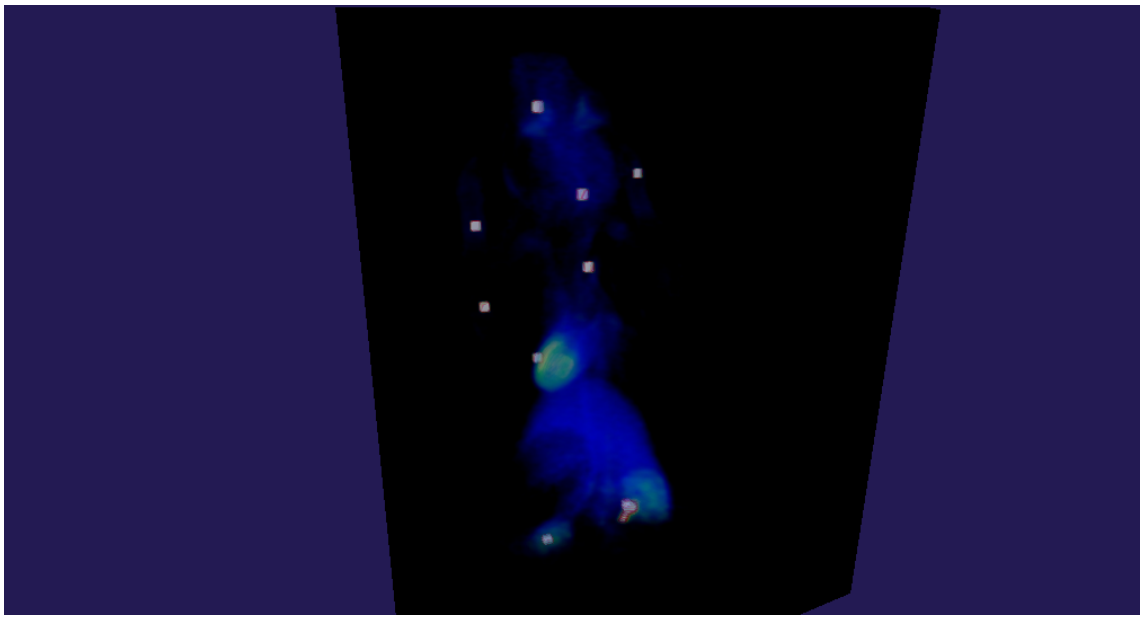
Taulukko 4: Ei suodatusta, hakuhaarukka 8 (17^3)



Kuva 10: Suoritusajat, kun kuvaa ei suodateta. Laiska haku arvolla 1 on jätetty pois kuvasta selkeyden säilyttämiseksi (8478,71 sekuntia $\approx$ 2 tuntia 21 minuuttia).



Kuva 11: Suodatus, laiska haku 8 (512), hakuhaarukka 5 (11^3), keskiarvo suoritusaika 0,15 sekuntia. ID: Kohde 1. Valkoiset pisteet ovat ohjelman löytämiä lokaaleja maksimeja.



Kuva 12: Suodatus, laiska haku 8 (512), hakuhaarukka 15 (31^3), keskiarvo suoritusaika 0,36 sekuntia. ID: Kohde 1. Valkoiset pisteet ovat ohjelman löytämiä lokaaleja maksimeja.

Kun käytetään suodatusta, laiskaa hakua arvolla 8 (512) sekä hakuhaarukalla 15 (31^3) saadaan kuva, jossa on merkitty vain yhdeksän lokaalia maksimia. Suuri hakuhaarukka hill climbing -algoritmille takaa sen, että suurin osa turhista maksimeista tippuu pois (kuva 12) ilman suurempaa työtä.

Jos metodia sovellettaisiin ihmisen koko kehon PET-kuvalle, on vokselimäärä paljon isompi kuin näissä kuvissa. $400 \times 400 \times 600$ kuvassa on $\frac{15625}{424} \approx 36.85$ kertaa enemmän vokseleita kuin $128 \times 128 \times 159$ kuvassa. Jos oletetaan, että suoritusaika nousee suurin piirtein lineaarisesti vokselien määrän mukaan, kuvan lokaalien maksimien löytäminen veisi ≈ 13 sekuntia. Metodi on siis käyttökelpoinen isommillekin kuville.

5.2 Jaccard-indeksi

Jaccard-indeksi on keino mitata, kuinka hyvin kaksi aluetta ovat samanmuotoisia vertaamalla binäärimaskeja alueista. Tämä tarkoittaa, että segmentaatiokuvassa elimen alueella olevien vokselien arvot on muutettu ykkösiksi ja loput nollaksi. Jaccard-indeksi voidaan määritellä kahden diskreetin joukon välille (kuva 13).[11]

Määritelmä 3. Olkoon A manuaalisesti piirretty referenssisegmentaatio ja B ohjelman löytämä segmentaatio. Tällöin A:n ja B:n välinen Jaccard-indeksi on:

$$J(A, B) = \frac{M_{11}}{M_{01} + M_{10} + M_{11}}$$

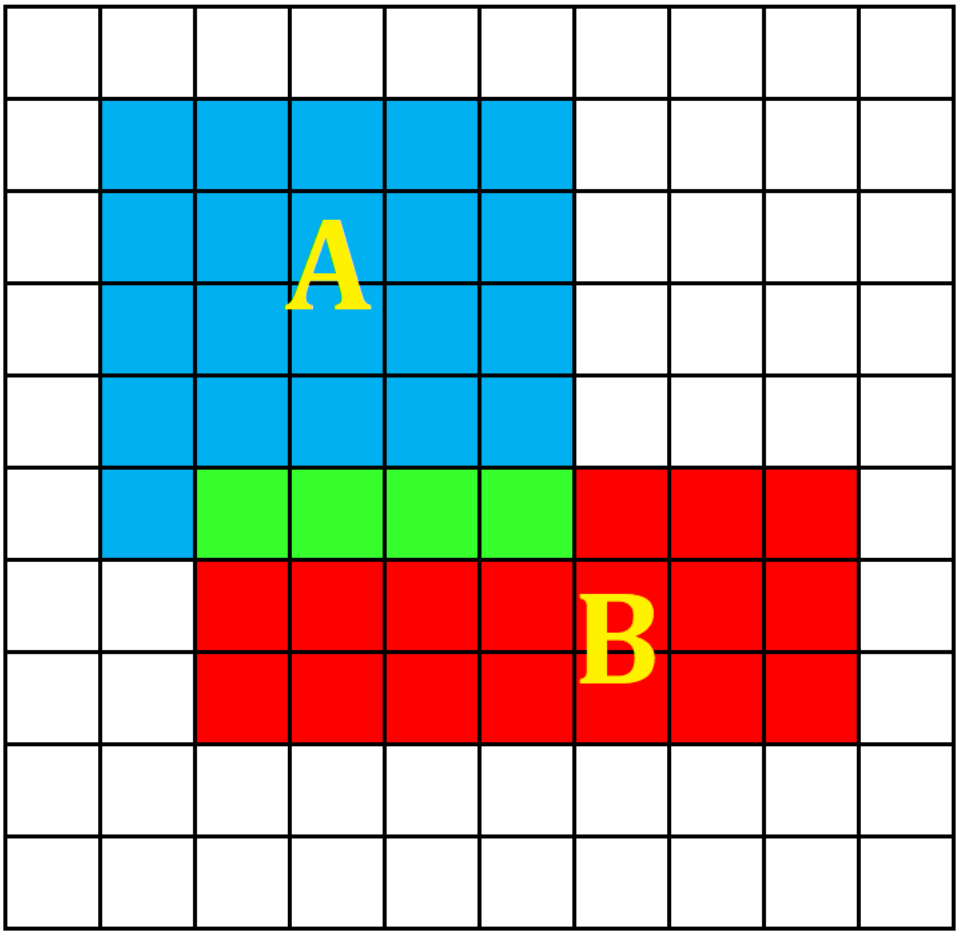
missä

- M_{11}: vokseli$_{i,j,k} \in A,\ B$
- M_{01}: vokseli$_{i,j,k} \in A$ ja $\notin B$
- M_{10}: vokseli$_{i,j,k} \notin A$ ja $\in B$
- M_{00}: vokseli$_{i,j,k} \notin A,\ B$

Esimerkkikuvassa (kuva 13) Jaccard-indeksiksi J tulee $\frac{4}{4+21+17} \approx 0{,}095$. Täydellinen vastaavuus tarkoittaisi, että $J = 1$. Vastaavasti, jos alueet ovat täysin erillään, $J = 0$. Sama periaate toimii myös kolmannessa (sekä useammassa) ulottuvuudessa. Näin saadaan yksi metriikka mitata segmentoinnin onnistumista, kun verrataan manuaalisesti piirrettyä kuvaa sekä ohjelman automaattisesti luomaa kuvaa.

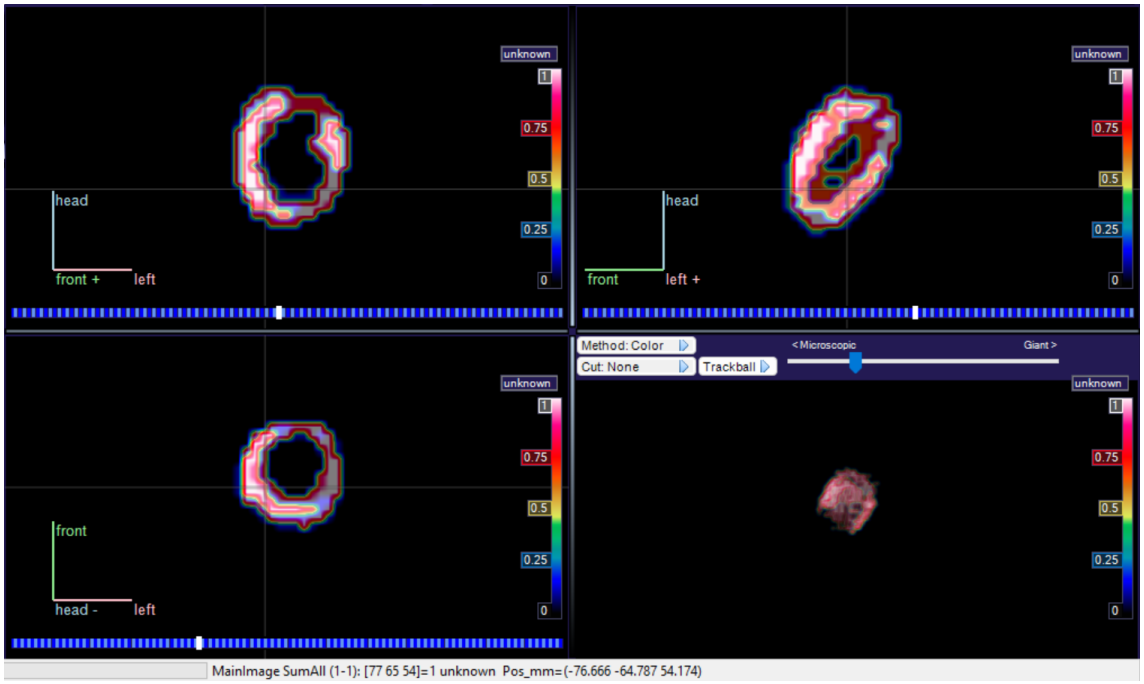
5.3 Lokaalien maksimien värjääminen ja parametrien valinta

Kun kuvasta on löydetty lokaalit maksimit, hyödynnetään flood fill -algoritmia näihin pisteisiin. Jotta flood fill -algoritmin värjäys onnistuisi mahdollisimman hyvin, täytyy valita sopiva toleranssiparametri. Hyvän toleranssin löytäminen on tärkeää segmentaation kannalta. Liian pieni toleranssi johtaa siihen, että koko elintä ei värjätä ja segmentaatio jää vajaaksi. Liian suuri toleranssi puolestaan johtaa segmentaatioalueen vuotoon yli varsinaisen elimen rajojen. Esimerkkejä epäonnistuneista



Kuva 13: Sininen alue kuvaa joukkoa M_{01}. Punainen alue kuvaa joukkoa M_{10}. Vihreä alue kuvaa joukkoa M_{11}. M_{00} on kummankin A ja B ulkopuolinen alue. Ulkopuolisesta alueesta ei tarvitse pitää lukua.[11]

toleranssin määrityksistä on esitetty kuvissa 14 ja 15. Koska kuvien arvot vaihtelevat paljon, on parasta käyttää prosentuaalista parametria. Tämä tarkoittaa, että ohjelmaan syötettävä parametri on prosenttiluku lokaalin maksimin omasta arvosta. Tällöin parametrin arvot rajoittuvat alueelle $(0, 1)$. Arvo 0 tarkoittaisi, että vain tämä piste kuuluisi väritettävään alueeseen ja arvo 1, että kaikki vokselit alle tämän pisteen arvon väritettäisiin. Toleranssi on jokin funktio $f(\mathbf{x})$, jonka arvot sijoittuvat välille $[0, 1]$ ja jossa $\mathbf{x}$ on vektori ominaisuuksista, joiden perusteella toleranssi määrätään. Käytännössä tämän funktion f löytäminen on vaikeaa. Siksi toleranssi on tässä tapauksessa etsitty haarukoimalla.

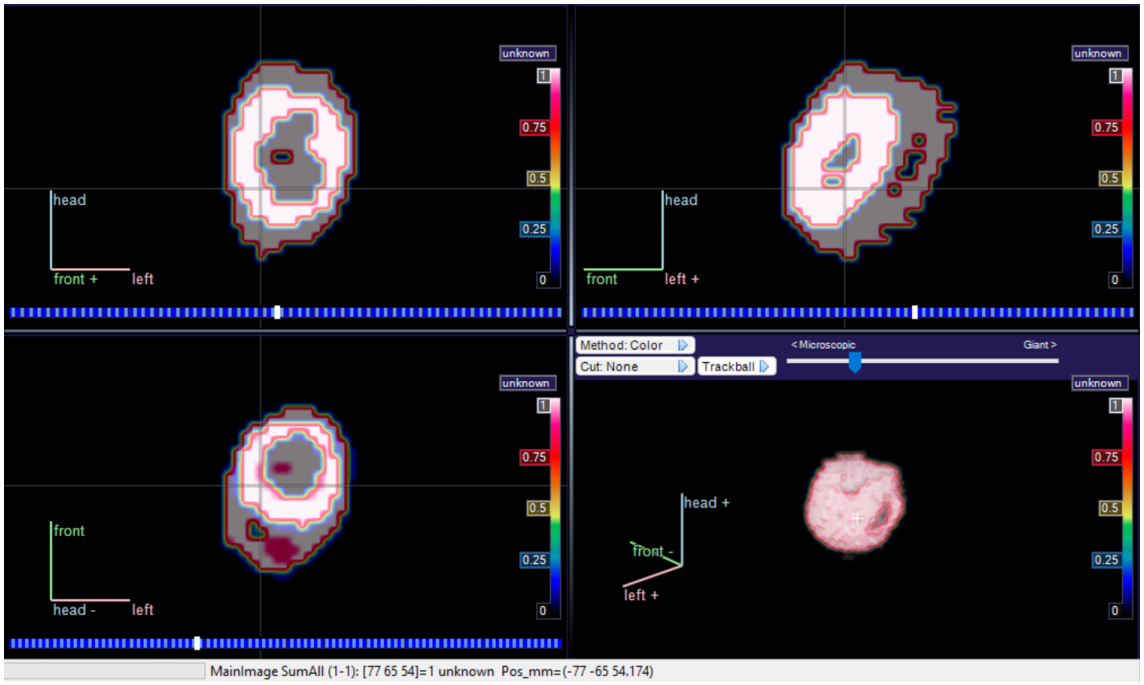


Kuva 14: Taustalla oleva haaleampi alue on manuaalinen segmentaatio ja kirkas osuus ohjelman automaattisesti värjäämä. Sydän jää alivärjätyksi. Käytetty toleranssi oli $0{,}2 \cdot M_{lokaalimaksimi}$. $J = 0{,}3797$, ID: Kohde 1.

6 Tulokset

6.1 Manuaalisten segmentontien alkukäsittely

Manuaalisesti segmentoidut kuvat vaativat hieman käsittelyä ennen kuin niitä pystyttiin vertaamaan. Niissä x- ja y-akselit olivat vaihtaneet paikkaa sekä z-akseli oli peilautunut. Lisäksi munuaisten manuaaliset segmentaatiot olivat samassa kuvassa ja nämä erotettiin kahdeksi omaksi kuvakseen.



Kuva 15: Kirkas osuus on manuaalinen segmentaatio ja taustalla oleva haaleampi alue on ohjelman automaattisesti värjäämä alue. Sydän värjätään reippaasti yli rajojen. Käytetty toleranssi oli $0{,}65 \cdot M_{lokaalimaksimi}$. $J = 0{,}3298$, ID: Kohde 1.

6.2 Tulosten keräys

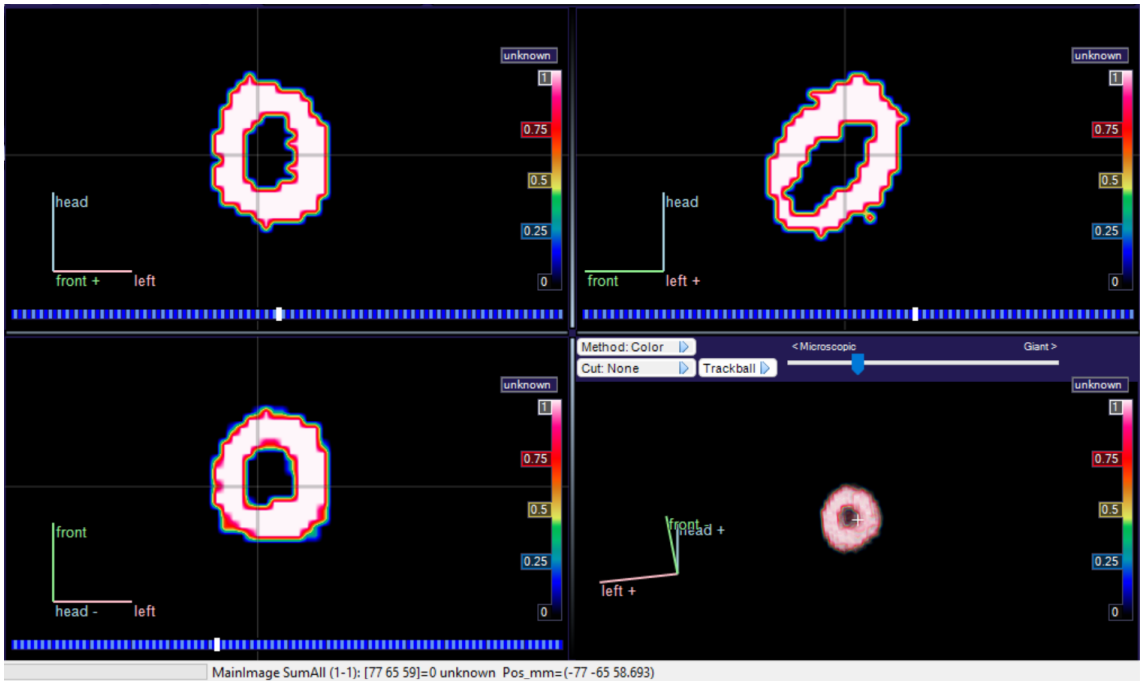
Tulokset kerättiin niin, että manuaalinen segmentointi sovitettiin mahdollisimman pienen suorakulmion sisään ja valittiin kaikki löydetyt lokaalit maksimit tämän sisältä. Tämän jälkeen kaikki pisteet värjättiin samalla toleranssiparametrilla. Lopulta tätä tulosta verrattiin manuaaliseen segmentaatioon Jaccard-indeksin avulla.

Koska hyvän toleranssin löytäminen on vaikeaa, käytettiin haarukointia hyvän pohjatuloksen saamiseen. Haarukointivälit olivat $(0{,}20, 0{,}25, ..., 0{,}90, 0{,}95)$. Jokaisella arvolla värjättiin tietty lokaali maksimi ja tätä verrattiin manuaaliseen segmentaatioon. Paras tulos otettiin talteen. Tulokset ovat taulukoituna liitteessä 1.

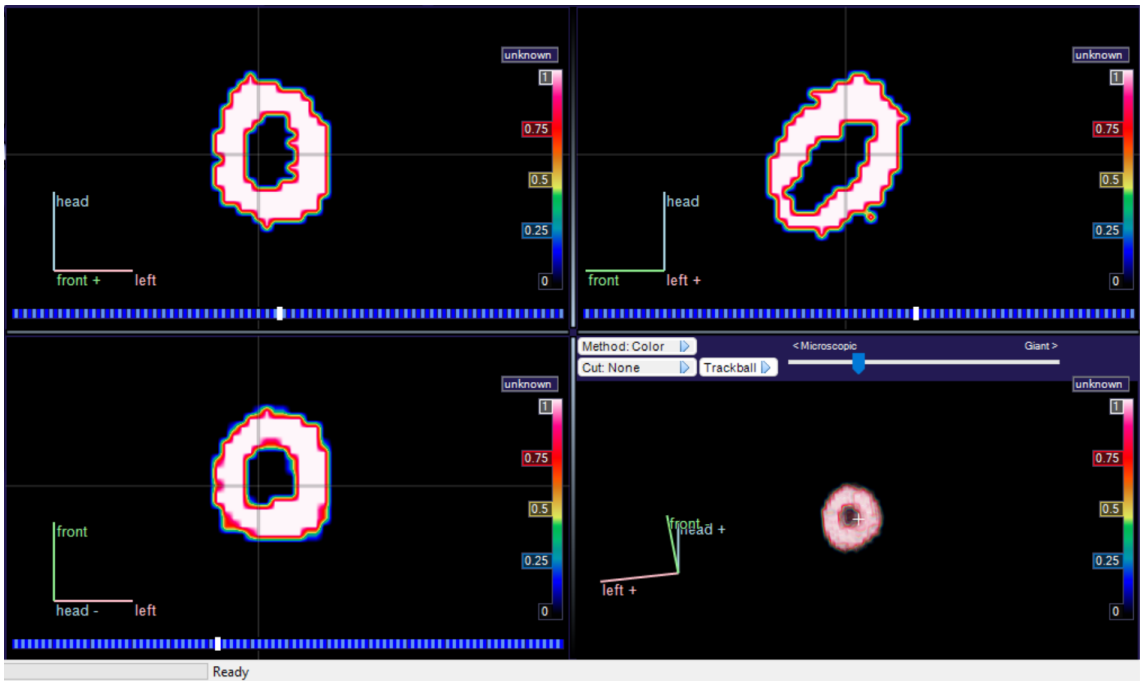
Taulukon 5 tuloksista nähdään, että menetelmä toimii melko hyvin sydämen löytämiseen. Automaattisen ja manuaalisen segmentaation ero on demonstroitu kuvissa 16, 17 ja 18. Sydämen hyvät tulokset johtuvat siitä, että se on eristynyt elin, jolla ei ole useampia lokaaleja maksimeja tai muita elimiä lähellä. Vastakohtana ovat aivot, joissa tapahtuu 'vuotoa' (kuva 19)(kuva 20) silmien takia, jotka näkyvät myös lokaaleina maksimeina kuvassa kirkkaampina kuin aivot (kuva 21). Munuaisten kohdalla tapahtuu joissakin kuvissa ylivärjäystä. Tämä nähdään kuvista 22, 23 ja 24. Ylivärjäys johtuu siitä, että munuaisissa on osa, joka ei kuulu segmentoitavaan alueeseen ja joka on kirkkaampi kuin munuainen itse (kuva 25). Toinen ongelma on, että kaikissa kuvissa munuaiset eivät ole kokonaan kuvassa. Kuvat päättyvät rotan keskivartalon kohdille. Kirkkain kohta ei siis välttämättä ole kuvassa, joten värjäykseen käytetään jotain arvoa, joka ei ole 'oikea' lokaali maksimi (kuva 26), ja todellinen piste on kuvan ulkopuolella.

	Aivot	Sydän	Oikea munuainen	Vasen munuainen
Keskiarvo:	0,3619	0,9362	0,6519	0,7681
Minimi:	0,2034	0,6936	0,2864	0,1439
Maksimi:	0,5273	0,9974	0,9666	0,9500
Kpl:	40	40	40	25

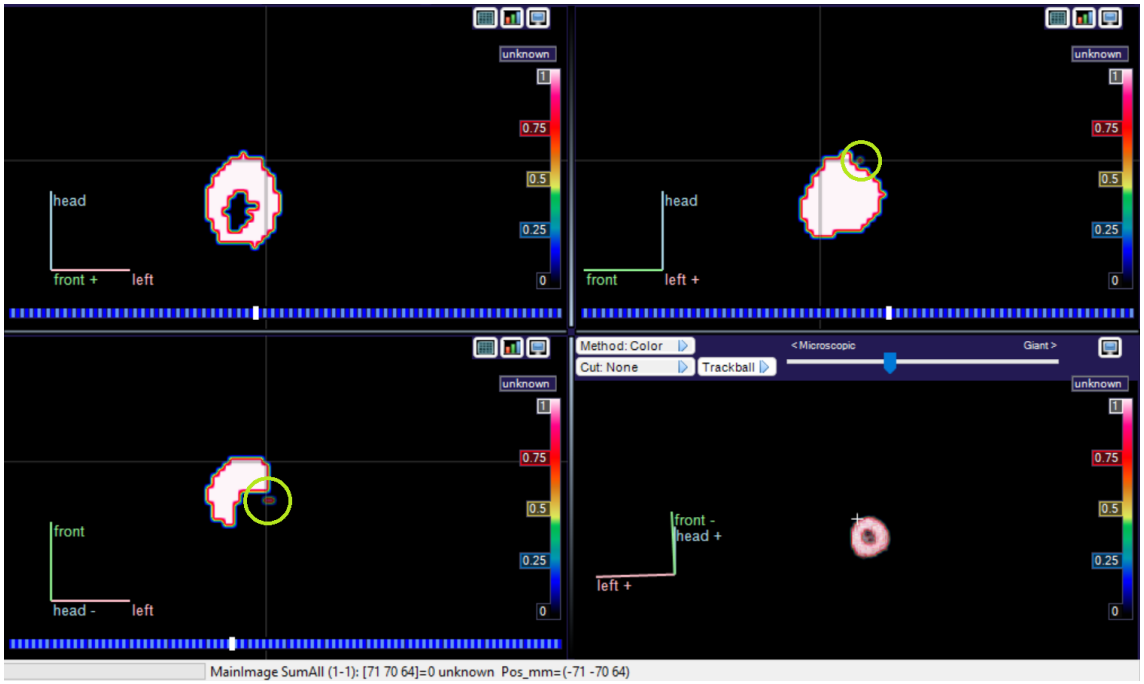
Taulukko 5: Jaccard-indeksin keskiarvotulokset sekä huonoimmat ja parhaimmat tulokset.



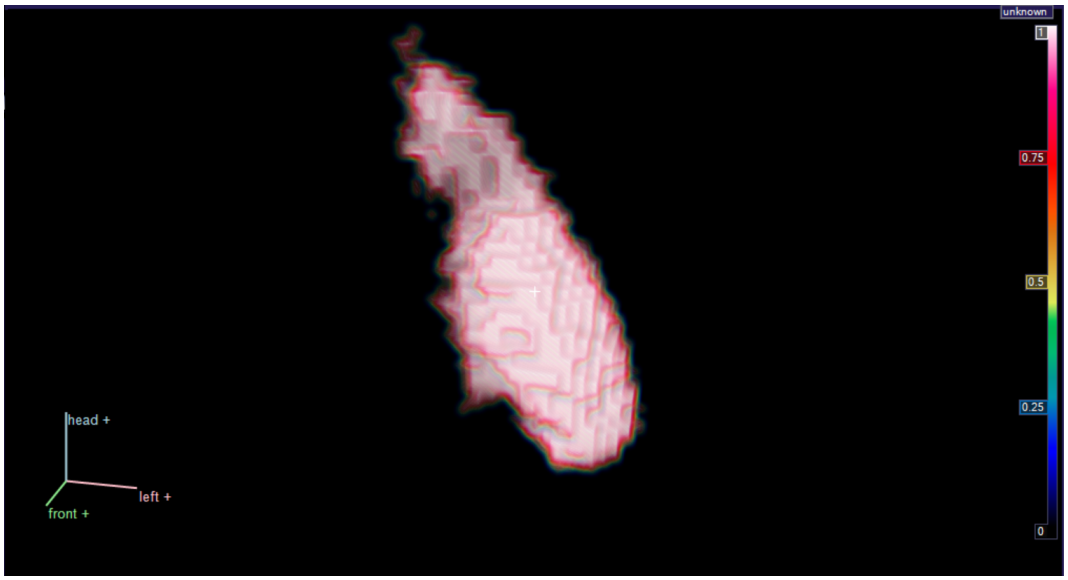
Kuva 16: Manuaalinen segmentaatio sydämestä.



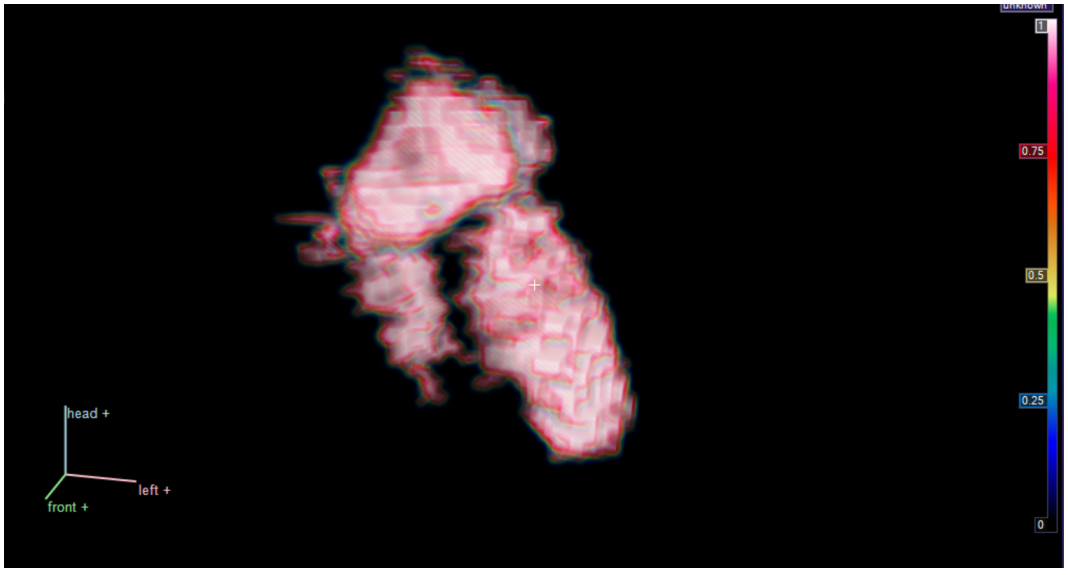
Kuva 17: Ohjelmalla tehty automaattinen segmentaatio sydämestä.



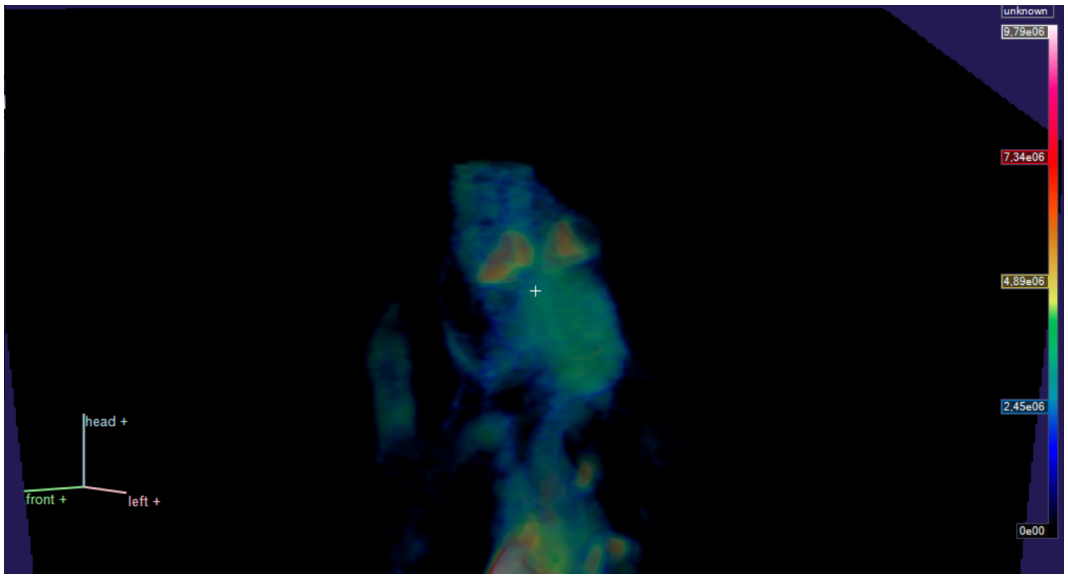
Kuva 18: Ero on huomattavissa vain yhdessä kohdassa. $J = 0{,}9974$ [Liite 1], ID: Kohde 7.



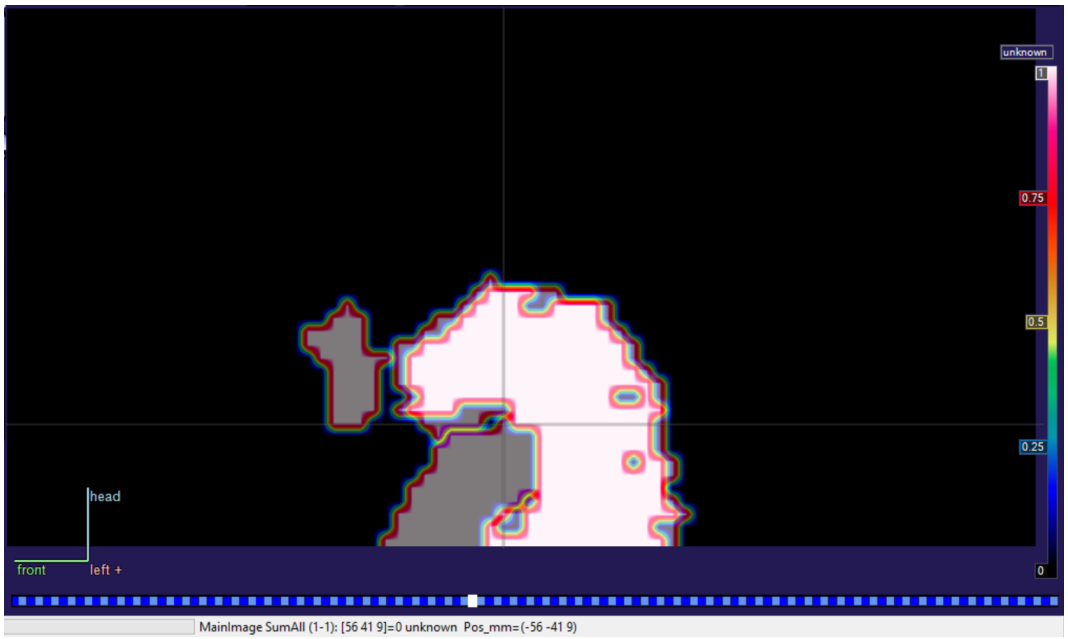
Kuva 19: Manuaalinen segmentaatio, kolmiulotteinen kuva. ID: Kohde 34.



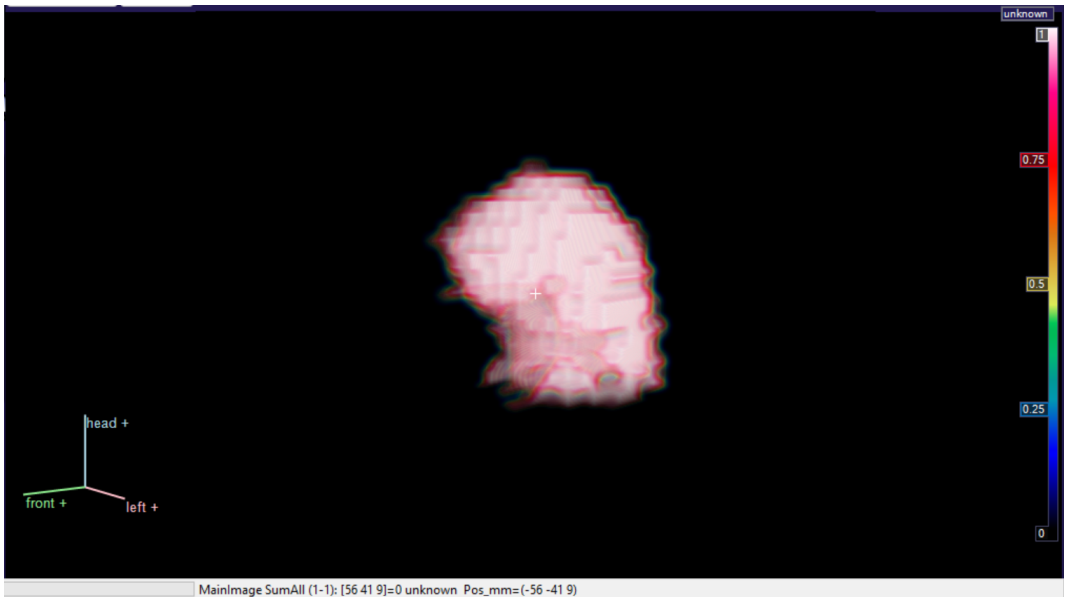
Kuva 20: Automaattinen segmentaatio, kolmeulotteinen kuva. $J = 0,4186$ [Liite 1], ID: Kohde 34.



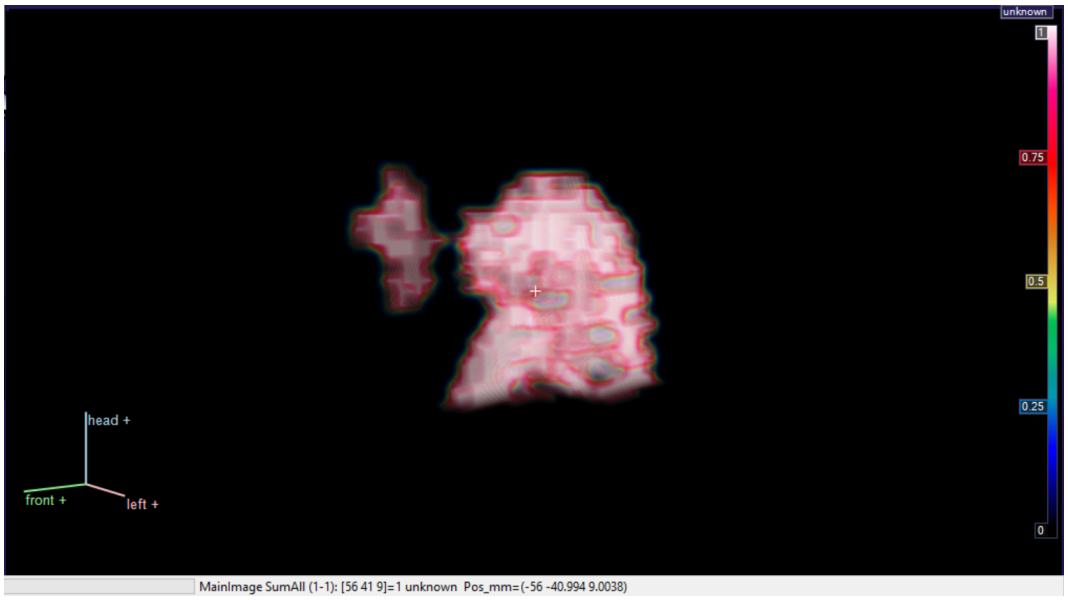
Kuva 21: Silmät (punaiset alueet) ovat kirkkaammat kuin aivot (vihreä alue silmien alapuolella). Värjäys alkaa näistä aivojen sijaan. ID: Kohde 1.



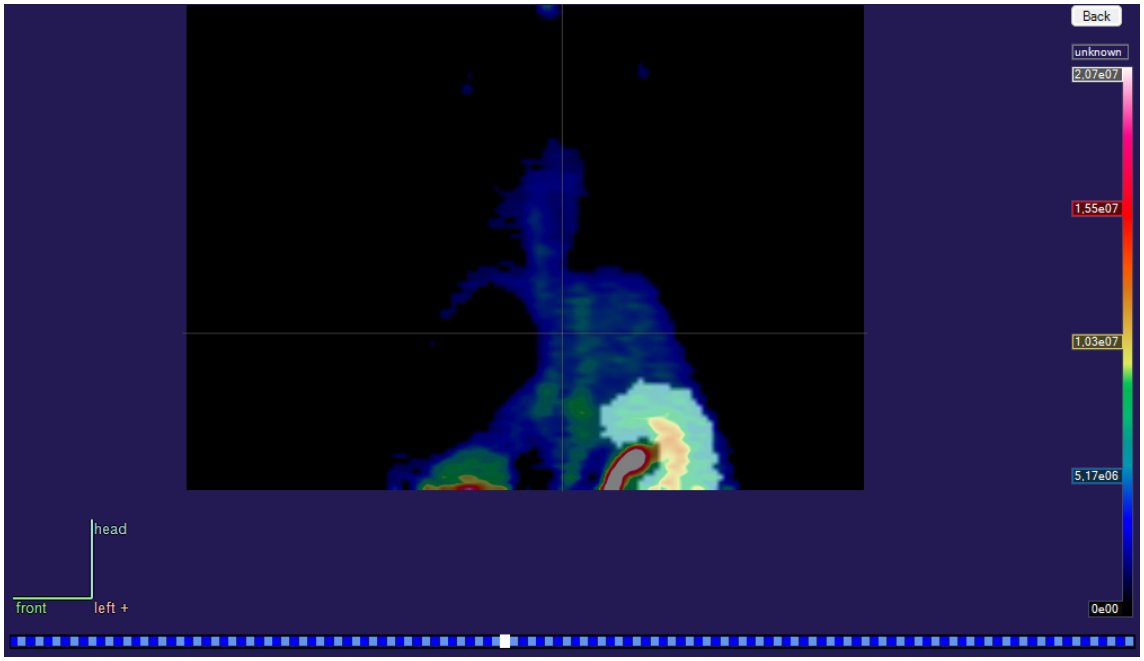
Kuva 22: Kirkas alue on manuaalinen segmentaatio, ja tummempi tausta on ohjelman löytämä. $J = 0{,}6169$ [Liite 1], ID: Kohde 1.



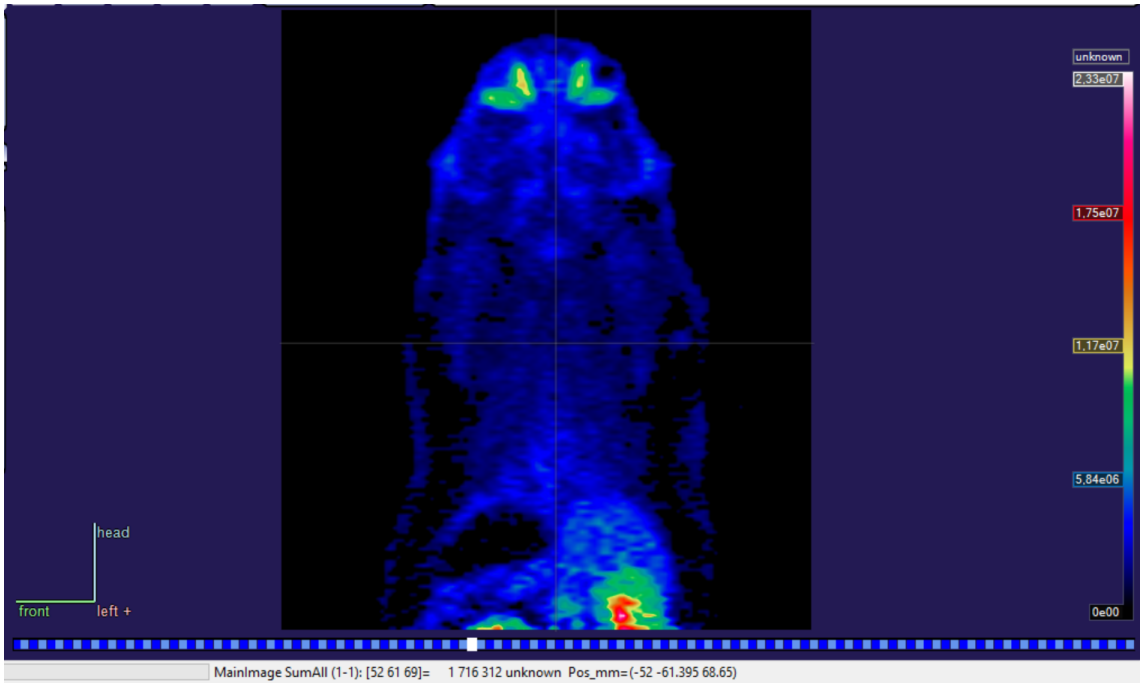
Kuva 23: ID: Kohde 1, manuaalinen segmentaatio, kolmiulotteinen kuva.



Kuva 24: ID: Kohde 1, automaattinen segmentaatio, kolmiulotteinen kuva.



Kuva 25: Valkoinen alue on manuaalinen segmentaatio. Kuvasta huomataan, että kirkkain osa on munuaiseen kiinnittyvä rakenne. Ohjelma löytää maksimin kirkkaammasta kohdasta. ID = Kohde 1.



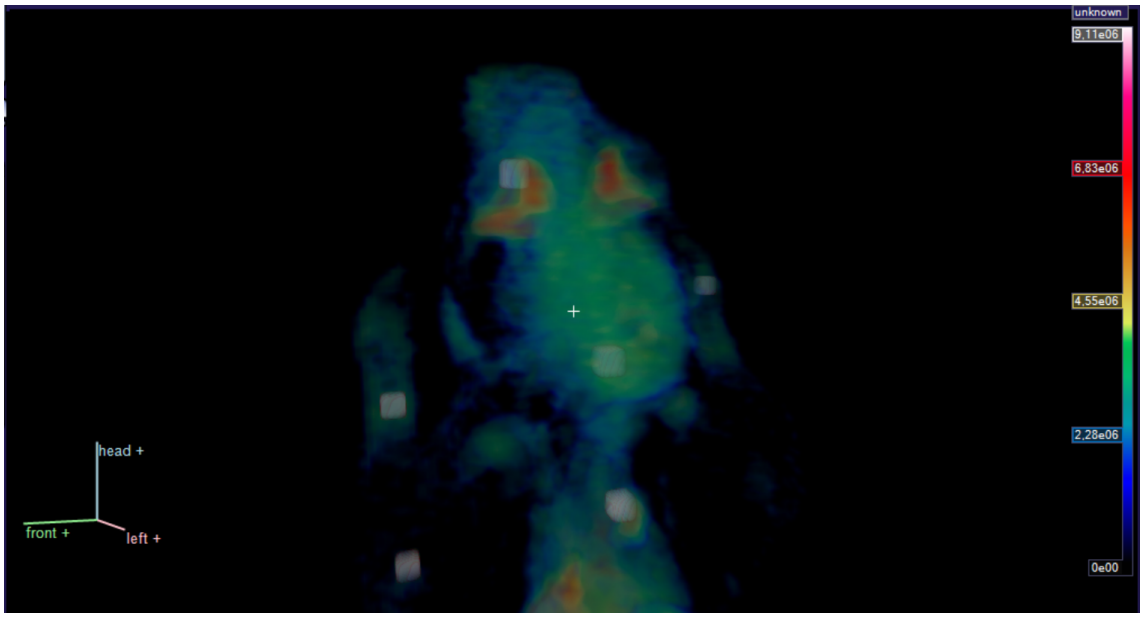
Kuva 26: Osassa kuvista munuaiset ovat leikkaantuneet keskeltä. Tämä saattaa vaikuttaa siihen, kuinka hyvin elin tunnistetaan. Varsinainen oikea lokaali maksimi saattaa jäädä kuvan ulkopuolelle. $J = 0{,}9061$ (vasen munuainen), $J = 0{,}7702$ (oikea munuainen) [Liite 1], ID: Kohde 35.

7 Pohdinta

Tämän tutkielman tavoitteena oli automaattinen PET-kuvan segmentointi. Suurimmat automatisointiin liittyvät vaikeudet käytetyissä metodeissa olivat flood fill -parametrin valinta sekä segmentaation ylivuotaminen, kun elimet eivät selkeästi erotu taustakudoksesta. Tässä luvussa esitellään korjausmahdollisuuksia näille ongelmille.

7.1 Flood fill sekä hill climbing

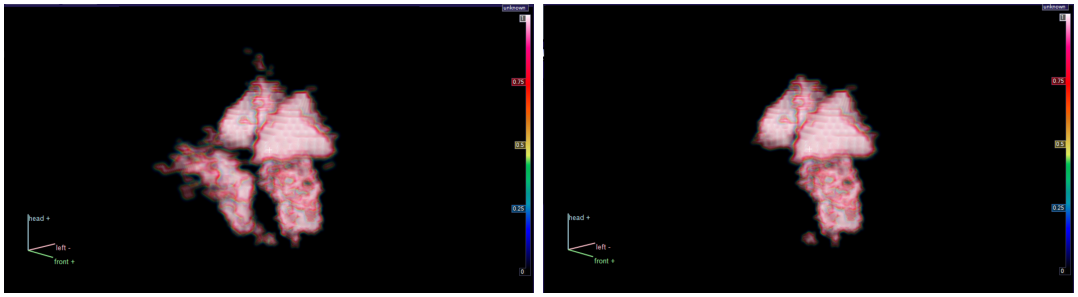
Flood fill- ja hill climbing-metodeissa ongelmana on oletus siitä, että elin tai tarkasteltava alue on selvästi irrallinen muista kehon osista. Tämän voi huomata esimerkiksi pään alueesta, jossa aivot ovat luonnollisesti yhteydessä silmiin. Silmät ovat kirkkaammat kuin aivot, joten hill climbing löytää ne aivojen lisäksi (kuva 27). Jotta aivot saadaan värjättyä, täytyy toleranssin olla korkea. Tästä syystä aivoja ympäröivät alueet värjäytyvät myös helposti, koska ne eivät eroa arvoiltaan niin paljoa kuin esimerkiksi sydän eroaa ympäristöstään. Flood fillin liiallista leviämistä voi hieman estää vaihtamalla *connectivity* asetusta, jolloin vokselin naapurusto pienenee (kuva 28). Tämä ei kuitenkaan aina paranna tuloksia, ja mahdollinen parannus on usein minimaalinen.



Kuva 27: Silmä sekä aivot merkitään lokaaleina maksimeina (valkoiset pisteet). Värjäyksen aloituspisteinä on käytetty molempia. Subject ID = Kohde 1.

Toinen mahdollinen parannus olisi muuttaa toleranssin käyttäytymistä. Flood fill toimii tällä hetkellä niin, että se etsii arvoja, jotka ovat toleranssin rajoissa pienempiä sekä isompia kuin aloituspiste, eli värjätään kaikki naapuruston vokselit, joiden arvot ovat välillä

$$(M_{i,j,k} - t, M_{i,j,k} + t),$$



Kuva 28: *Connectivity* parametrin vaikutus segmentaatioon. Vasemmalla *connectivity* = 1 ja oikealla *connectivity* = 2. $J = 0,3622$ [Liite 1] ja $J = 0,4237$, ID: Kohde 21. Tässä kuvassa parannus on melko hyvä, mutta usein Jaccard-indeksin tulos parani vain noin 0,005 verran.

missä t on toleranssin absoluuttinen arvo. Tämä voitaisiin muuttaa niin, että värjättäisiin arvot väliltä

$$(M_{i,j,k} - t_1, M_{i,j,k} + t_2),$$

missä $t_2 < t_1$ tai jopa $t_2 = 0$. Flood fill -parametrin rajoitus johtuu tällä hetkellä käytetystä kirjastosta [9].

7.2 Suurennettu suodatus

Flood fill -metodia käytettäessä saattaa tapahtua vuotoa, kuten päänalueella huomataan. Vuotoa voitaisiin ehkäistä nostamalla suodatusparametria korkeammaksi. Tällöin elimiä ympäröivä alue saattaa muuttua nollaksi, jolloin flood fill ei jatka sinne asti, vaan lopettaa reunaan.

7.3 Parametrin valinta ja koneoppiminen

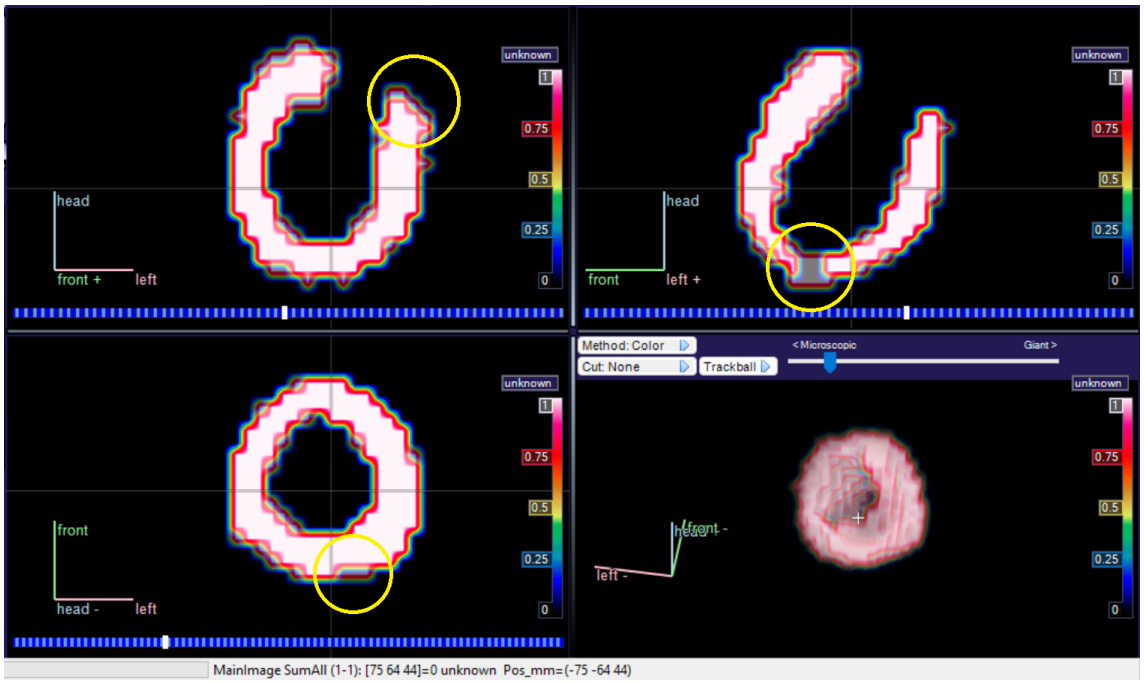
Testien perusteella flood fill voi saavuttaa melko hyvän Jaccard-indeksin arvon, sillä parhaimmat arvot sydämen kohdalla ovat ≈ 0.98 vastaavia. Tämä vaatii kuitenkin hyvän toleranssiparametrin valinnan flood fill -algoritmille. Ongelmana on, että hyvää funktiota f on vaikea löytää. Tässä työssä olevat tulokset on saatu haarukoimalla paras mahdollinen toleranssiparametrin arvo. Se ei kuitenkaan ole käytännöllistä automaation kannalta. Ensinnäkin se vie paljon aikaa sekä vaatii loppujen lopuksi ihmisen työpanosta erottelemaan, mikä haarukointi antaa parhaimman tuloksen.

Tässä on mahdollista hyödyntää koneoppimista. Regressiomallilla voi ennustaa mahdollisen toleranssin antamalla syötteeksi esimerkiksi lokaalin maksimin prosentuaalisen arvon maksimista sekä tiedon elimestä ja sijainnista. Kyseessä olevan elimen voi tunnistaa sijainnin perusteella. Sillä oletuksella, että kaikki kuvat ovat samoin päin z-akselin suhteen, voidaan jo näiden kuvien kohdalla päätellä, että onko kyseessä aivot, sydän vai jompikumpi munuainen. Munuaiset voidaan erotella toisistaan y-akselin mukaan.

Sklearnin ExtraTreesRegressorilla[18] tehty testimalli onnistui jonkinasteisesti (kuva 29). Käytetyt parametrit olivat oletusparametrit, poikkeuksena estimaattorien määrä, joka oli 1000.

Malli tehtiin vain sydämelle, jotta se pysyisi yksinkertaisena ja helposti tulkittavana. Näin toisten elimien arvot eivät sekoita mallia. Sydän valittiin siksi, että se segmentoitiin hyvällä tarkkuudella lähes kaikissa kuvissa. Koulutukseen käytettiin 33 kuvaa ja 7 kuvaa toimi testijoukkona. Testijoukko valittiin satunnaisesti. Tulokset ovat taulukossa 6.

Tuloksiin vaikuttaa pieni datamäärä, mallin yksinkertaisuus ja nopea kehitystyö. Mallille annettiin vain z-akselin sijainti sekä lokaalin maksimin prosentuaalinen arvo maksimiin verrattuna. Ennustaminen näin pienestä määrästä ominaisuuksia on haastavaa. Lisäksi on huomattava, että haarukoidut tulokset olivat koulutusaineistona, eivätkä haarukoidut arvot ole todellisia parhaimpia arvoja toleranssille (paras arvo löytyy todennäköisesti haarukointipisteiden välistä). Taulukossa 7 on esitetty haarukoidut toleranssit verrattuna koneoppimismallin arvoihin. Tämä kuitenkin näyttää, että nopeasti tehdyllä yksinkertaisella mallilla saatiin suhteellisen hyviä tuloksia, kun taas pidemmälle kehitetty malli voisi pystyä paljon parempaankin.



Kuva 29: Reunoilla näkyvä haaleampi alue on manuaalinen segmentaatio, kun taas kirkas alue on saatu koneoppimisen toleranssiparametrilla. Kuvista on ympyröity muutama kohta, joista huomaa eron. Haarukointi: $J = 0{,}9443$, ExtraTreesRegressor: $J = 0{,}8626$, [Taulukko 6] ID: Kohde 4.

7.3.1 Adaptiivinen flood fill -toleranssi

Flood fill -metodin toleranssin voisi määritellä vokselikohtaiseksi eikä kiinteäksi arvoksi, jota käytetään kaikkialla. Kun viereisen vokselin arvo eroaa tarpeeksi jyrkästi nykyisestä vokselista, tämä jätettäisiin ulos valinnasta (kuva 30).

Toinen tapa voisi olla pienentää toleranssia sitä pienemmäksi mitä kauemmaksi lokaalista maksimista siirrytään. Näin voitaisiin estää ylivärjääminen. Nämä kak-

ID	Haarukoimalla saatu Jaccard-indeksi	Koneoppimismallin Jaccard-indeksi
Kohde 1	0,9566	0,8032
Kohde 4	0,9443	0,8626
Kohde 11	0,9848	0,8159
Kohde 21	0,9065	0,6953
Kohde 23	0,9427	0,8789
Kohde 25	0,9541	0,9216
Kohde 39	0,9277	0,7201

Taulukko 6: Haarukoinnilla saatu Jaccard-indeksi vastaan koneoppimismalli.

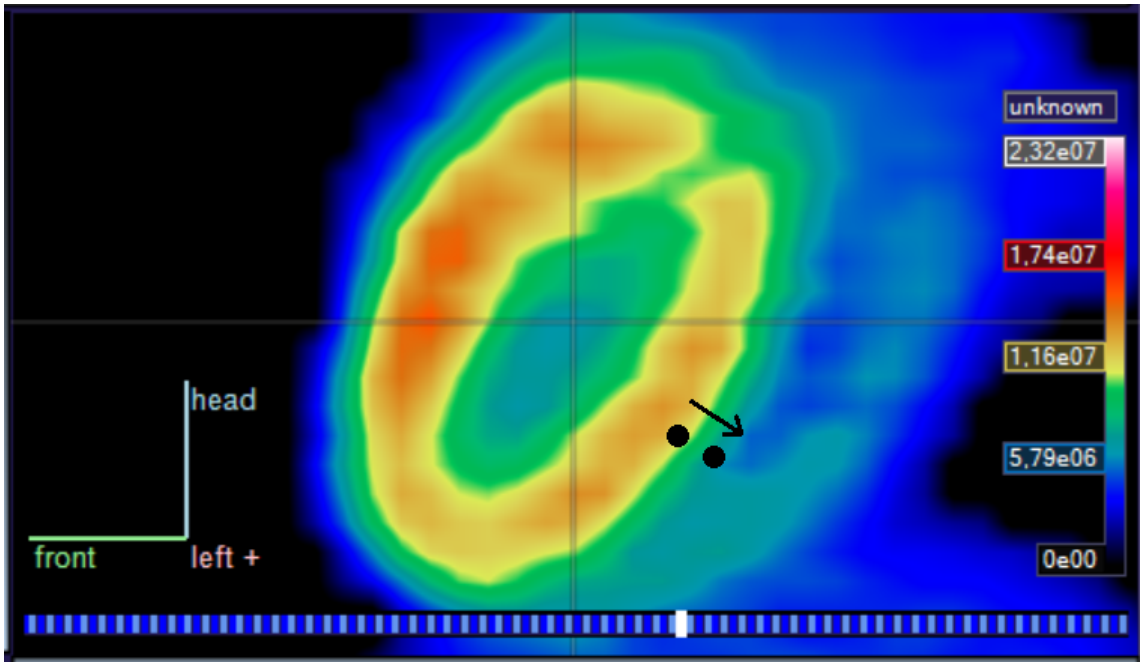
ID	Haarukoitu toleranssiparametri	Koneoppimismallin toleranssiparametri
Kohde 1	0,55	0,3940
Kohde 4	0,45	0,4616
Kohde 11	0,45	0,4000
Kohde 21	0,35	0,4266
Kohde 23	0,55	0,4340
Kohde 25	0,45	0,5381
Kohde 39	0,35	0,4284

Taulukko 7: Haarukoidut toleranssit vastaan koneoppimismallilla ennustetut toleranssit.

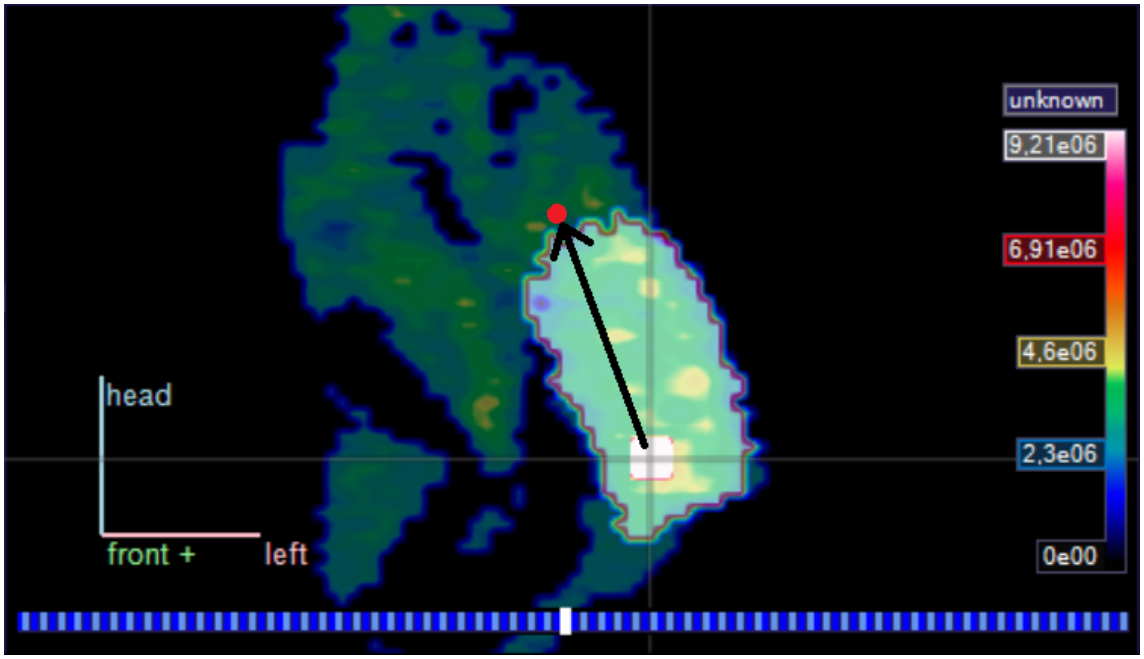
si keinoa voitaisiin myös yhdistää niin, että lähellä lokaalia maksimia vaadittava arvojen ero tulisi olla suurempi kuin kaukana (kuva 31).

7.4 Elinten tunnistaminen

Hill climbing -algoritmi löytää hyvin lokaalit maksimit. Ongelmana on, että kuva sisältää kuitenkin melko runsaasti epäkiinnostavia lokaaleja maksimeja. Nämä pitäisi erotella jotenkin. Tähän voisi myös soveltaa koneoppimista. Syötteet voisivat olla lokaalin maksimin sijainti sekä aika-aktiivisuuskäyrä, ja malli voitaisiin kouluttaa tunnistamaan elimen näiden perusteella. Tämän jälkeen flood fill -parametrin voisi valita paremmin sen perusteella, mikä elin on kyseessä. Toisaalta, mikäli elimen värjäys onnistuu hyvin, voisi koneoppimismallin kouluttaa tunnistamaan saatu maski. Tällaisia kaksiulotteisia menetelmiä on jo olemassa ja niitä voisi tässäkin tehtävässä hyödyntää.



Kuva 30: Kuvaan merkittyjen vokselien arvot eroavat jyrkästi toisistaan. Tällaisessa tilanteessa lopetetaan värjäys tähän.



Kuva 31: Punaisen pisteen kohdalla toleranssi olisi paljon pienempi kuin lähtöpisteen lähellä. Näin voisi estää värjäystä vuotamasta liian pitkälle. Vaalea alue on manuaalinen segmentointi erottamaan aivojen rajat.

Viitteet

[1] https://www.siemens-healthineers.com/fi/molecular-imaging/pet-ct/biograph-vision-quadra, luettu 7.6.2023.

[2] Rainio O. Han, C., Teuho J., Nesterov S. V., Oikonen V., Piirola S., Laitinen T., Tättäläinen M., Knuuti J. & Klén R. Carimas: An Extensive Medical Imaging Data Processing Tool for Research. Journal of Digital Imaging, 2023, https://doi.org/10.1007/s10278-023-00812-1.

[3] McCarten K. M., Nadel H. R., Shulkin B. L., & Cho S. Y. Imaging for Diagnosis, Staging and Response Assessment of Hodgkin Lymphoma and Non-Hodgkin Lymphoma. Pediatric Radiology, vol. 49, no. 11, 2019, pp. 1545–64, https://doi.org/10.1007/s00247-019-04529-8.

[4] Agrawal A. & Venkatesh R. PET/CT in Lung Cancer. Springer, 2018. https://doi-org.ezproxy.utu.fi/10.1007/978-3-319-72661-8

[5] Gupta T. Radiation, Ionization, and Detection in Nuclear Medicine. Springer, Berlin. (2013) s.13-14

[6] https://github.com/spalonen/APS/

[7] https://turkupetcentre.fi/carimas/files/archive/Html/d67.html, luettu 7.6.2023.

[8] Burtsev S. V. & Kuzmin Ye. P. An Efficient Flood-Filling Algorithm, Comput. & Graphics Vol. 17, No. 5, pp. 549-561, 1993

[9] https://scikit-image.org/docs/stable/auto_examples/segmentation/plot_floodfill.html, luettu 7.6.2023.

[10] https://scikit-image.org/, luettu 7.6.2023.

[11] Chung N. C., Miasojedow B., Startek M. & Gambin, A. Jaccard/Tanimoto Similarity Test and Estimation Methods for Biological Presence-Absence Data." BMC Bioinformatics, vol. 20, no. Suppl 15, 2019, pp. 644–644, https://doi.org/10.1186/s12859-019-3118-5.

[12] Aristophanous M., Penney B. C., Martel M. K. & Pelizzari C. A. A Gaussian Mixture Model for Definition of Lung Tumor Volumes in Positron Emission Tomography. Medical Physics (Lancaster), vol. 34, no. 11, 2007, pp. 4223–35, https://doi.org/10.1118/1.2791035.

[13] Hsu C.-Y., Liu C.-Y. & Chen C.-M. Automatic Segmentation of Liver PET Images. Computerized Medical Imaging and Graphics, vol. 32, no. 7, 2008, pp. 601–10, https://doi.org/10.1016/j.compmedimag.2008.07.001.

[14] Canny J. A Computational Approach to Edge Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986, doi: 10.1109/TPAMI.1986.4767851.

[15] Gu Z., Cheng J., Fu H., Zhou K., Hao H., Zhao Y., Zhang T., Gao S. & Liu J. CE-Net: Context Encoder Network for 2D Medical Image Segmentation. IEEE Transactions on Medical Imaging, vol. 38, no. 10, 2019, pp. 2281–92, https://doi.org/10.1109/TMI.2019.2903562.

[16] https://www.image-net.org/, luettu 7.6.2023.

[17] He K., Zhang X., Ren S. & Sun J. Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

[18] Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., Vanderplas J., Passos A., Cournapeau D., Brucher M., Perrot M. & Duchesnay E. Scikit-learn: Machine Learning in Python, Journal of Machine Learning Research, 12,2825–2830, 2011.

[19] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble. ExtraTreesRegressor.html, luettu 7.6.2023.

[20] Harris C.R., Millman K.J., van der Walt S.J., Gommers R., Virtanen P., Cournapeau D., Wieser E., Taylor J., Berg S., Smith N. J., Kern R., Picus M., Hoyer S., van Kerkwijk M. H., Brett M., Haldane A., Del Río J. F., Wiebe M., Peterson P., Géard-Marchant, Sheppard K., Reddy T., Weckesser W., Abbasi H., Gohlke C. & Oliphant T. E. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2.

Liite 1: Kaikki tulokset haarukoinnilla

ID	Aivot	Sydän	Oikea munuainen	Vasen munuainen
Kohde 1	0,4007	0,9566	0,6169	0,8875
Kohde 2	0,4611	0,9423	0,4557	0,8467
Kohde 3	0,2034	0,6936	0,6052	NaN
Kohde 4	0,4623	0,9443	0,7788	0,9500
Kohde 5	0,3511	0,9836	0,4116	0,6038
Kohde 6	0,4560	0,9866	0,5074	0,7260
Kohde 7	0,3874	0,9974	0,8669	0,9078
Kohde 8	0,5080	0,9956	0,3996	0,9007
Kohde 9	0,2993	0,9576	0,8567	0,8579
Kohde 10	0,3499	0,9721	0,6158	0,1439
Kohde 11	0,3764	0,9848	0,604	NaN
Kohde 12	0,3155	0,9447	0,7259	0,5223
Kohde 13	0,5273	0,9853	0,3819	0,3777
Kohde 14	0,4882	0,9089	0,5897	NaN
Kohde 15	0,3257	0,8401	0,4366	0,3133
Kohde 16	0,2839	0,9113	0,9069	0,8287
Kohde 17	0,4155	0,9038	0,9666	0,8531
Kohde 18	0,3024	0,9636	0,2912	0,7119
Kohde 19	0,3521	0,9473	0,7650	NaN
Kohde 20	0,2989	0,9769	0,6513	NaN
Kohde 21	0,3622	0,9065	0,7224	NaN
Kohde 22	0,3682	0,9791	0,6750	0,9339
Kohde 23	0,3585	0,9427	0,8538	0,7762
Kohde 24	0,2917	0,9790	0,7943	NaN
Kohde 25	0,4276	0,9541	0,5634	0,8278
Kohde 26	0,3344	0,9084	0,5011	NaN
Kohde 27	0,3594	0,9667	0,9388	0,9114
Kohde 28	0,3171	0,8958	0,7157	NaN
Kohde 29	0,3343	0,9034	0,2864	NaN
Kohde 30	0,2900	0,9341	0,8263	NaN
Kohde 31	0,3094	0,9775	0,8078	0,8774
Kohde 32	0,3328	0,9577	0,6802	NaN
Kohde 33	0,3245	0,9251	0,5940	NaN
Kohde 34	0,4186	0,9645	0,6109	NaN
Kohde 35	0,3492	0,9061	0,7702	0,9139
Kohde 36	0,3569	0,8730	0,6169	0,9252
Kohde 37	0,3564	0,9402	0,7178	0,8673
Kohde 38	0,3828	0,8868	0,7486	0,8864
Kohde 39	0,3114	0,9277	0,5289	NaN
Kohde 40	0,3271	0,9234	0,6901	0,8512

Taulukko 8: Punaisella värjätyt tulokset ovat huonoimmat ja vihreällä värjätyt parhaimmat. Oranssi väri kuvaa sitä, että ohjelman automaatio ei toiminut kunnolla ja vaati manuaalista säätöä asetuksiin. NaN tarkoittaa, että toista munuaista ei ollut kuvassa (tai automaatio ei löytänyt sitä)

Liite 2: Kaikkien kuvien haarukoinnilla löydetyt toleranssiparametrit

ID	Aivot	Sydän	Oikea munuainen	Vasen munuainen
Kohde 1	0,40	0,35	0,85	0,75
Kohde 2	0,55	0,30	0,90	0,90
Kohde 3	0,30	0,35	0,50	NaN
Kohde 4	0,45	0,45	0,85	0,40
Kohde 5	0,60	0,45	0,35	0,50
Kohde 6	0,40	0,50	0,65	0,65
Kohde 7	0,50	0,35	0,60	0,55
Kohde 8	0,45	0,50	0,85	0,90
Kohde 9	0,45	0,50	0,95	0,75
Kohde 10	0,35	0,40	0,90	0,50
Kohde 11	0,45	0,35	0,75	NaN
Kohde 12	0,50	0,40	0,90	0,90
Kohde 13	0,40	0,40	0,40	0,40
Kohde 14	0,35	0,45	0,80	NaN
Kohde 15	0,45	0,40	0,55	0,45
Kohde 16	0,30	0,40	0,85	0,55
Kohde 17	0,45	0,60	0,95	0,50
Kohde 18	0,75	0,50	0,90	0,95
Kohde 19	0,45	0,55	0,75	NaN
Kohde 20	0,45	0,40	0,60	NaN
Kohde 21	0,45	0,45	0,65	NaN
Kohde 22	0,45	0,55	0,45	0,60
Kohde 23	0,50	0,45	0,55	0,55
Kohde 24	0,50	0,45	0,70	NaN
Kohde 25	0,40	0,55	0,80	0,75
Kohde 26	0,40	0,40	0,50	NaN
Kohde 27	0,50	0,40	0,70	0,45
Kohde 28	0,45	0,40	0,65	NaN
Kohde 29	0,40	0,40	0,50	NaN
Kohde 30	0,40	0,55	0,80	NaN
Kohde 31	0,40	0,55	0,80	0,85
Kohde 32	0,40	0,55	0,50	NaN
Kohde 33	0,50	0,50	0,90	Nan
Kohde 34	0,45	0,60	0,80	Nan
Kohde 35	0,40	0,55	0,75	0,60
Kohde 36	0,50	0,55	0,85	0,90
Kohde 37	0,55	0,50	0,80	0,80
Kohde 38	0,50	0,40	0,55	0,50
Kohde 39	0,45	0,55	0,85	NaN
Kohde 40	0,45	0,50	0,80	0,60

Taulukko 9: Saadut toleranssiparametrit. Toleranssiparametri t tarkoittaa osuutta lokaalin maksimin arvosta eli $M_{lokaalimaksimi} \cdot t$ on lopullinen toleranssi, jota käytetään värjäämiseen.

Liite 3: Hill climbing -algoritmi kokonaisuudessaan

```
import numpy as np

def HillClimbing(image_data_3d, seed_point, scope):
    # image_data_3d: 3-ulotteinen kuvamatriisi
    # seed_point:    aloituspiste
    # scope:         hakualue

    # Etsitään korkein arvo ympäristöstä.
    # Alkupiste on keskellä.
    # Jokaisen akselin suuntaan on sama hakuetäisyys.
    x_scope = scope
    y_scope = scope
    z_scope = scope

    # Tarkistetaan ollaanko reunalla, mikäli ollaan, pienennetään aluetta
    # sen verran että ei jouduta ulkopuolelle.
    if(seed_point[0]-scope < 0):
        x_scope = seed_point[0]

    if(seed_point[1]-scope < 0):
        y_scope = seed_point[1]

    if(seed_point[2]-scope < 0):
        z_scope = seed_point[2]


    # Etsitään maksimiarvo alkupisteen ympäriltä.
    max_value = np.max(image_data_3d[
            seed_point[0] - x_scope:seed_point[0] +
            scope+1, seed_point[1] - y_scope:seed_point[1] +
            scope+1, seed_point[2] - z_scope:seed_point[2] +
            scope+1
            ])

    # Etsitään maksimiarvon koordinaatit (sekä se, onko niitä useampia).
    current_hill_local_multiples = np.where(image_data_3d[
            seed_point[0] - x_scope:seed_point[0] + scope+1,
            seed_point[1] - y_scope:seed_point[1] + scope+1,
            seed_point[2] - z_scope:seed_point[2] + scope+1
            ] == max_value)

    # Alustetaan muuttuja johon koordinaatit tallennetaan.
    current_hill_local = [0, 0, 0]
```

```
# Jos maksimiarvoja on useampia, valitaan näistä ensimmäinen joka
#tulee vastaan.
if(np.shape(current_hill_local_multiples)[1] >= 2):
    for i in range(scope):
        for j in range(scope):
            for k in range(scope):
                if(image_data_3d[seed_point[0]-scope + i]
                [seed_point[1]-scope + j]
                [seed_point[2]-scope + k]
                == max_value):
                    # Maksimi koordinaatit lokaalissa kontekstissa.
                    # Asetetaan uuden korkeimman
                    # kohdan koordinaatit tässä
                    # mikäli maksimeita oli useampi
                    current_hill_local[0] = i
                    current_hill_local[1] = j
                    current_hill_local[2] = k
                    break
                else:
                    continue
            break

# Usempia arvoja ei löytynyt, asetetaan lokaalissa
# kontekstissa koordinaatit (pienemmässä scope x scope x scope
# kuutiossa olevat).
else:
    current_hill_local[0] = current_hill_local_multiples[0][0]
    current_hill_local[1] = current_hill_local_multiples[1][0]
    current_hill_local[2] = current_hill_local_multiples[2][0]

# Lasketaan globaalit koordinaatit lokaalien koordinaattien
# sekä hakuhaarukka parametrin avulla.
# Tämä saadaan summaamalla
# (alkupiste - hakuhaarukka) + lokaalikoordinaatti
current_hill_x = seed_point[0]-x_scope + current_hill_local[0]
current_hill_y = seed_point[1]-y_scope + current_hill_local[1]
current_hill_z = seed_point[2]-z_scope + current_hill_local[2]

# Tarkistetaan oliko käytetty aloituspiste myös
# lokaalimaksimi scope x scope x scope kuution sisällä.
# Mikäli oli, on ohjelman suoritus valmis ja
# palautetaan nämä koordinaatit.
if(image_data_3d[seed_point[0]][seed_point[1]][seed_point[2]]
   == max_value):
    return max_value, current_hill_x, current_hill_y, current_hill_z
```

```
# Jos näin ei ollut, toistetaan HillClimbing uudelle
# korkeimman arvon omaavalle vokselille.
temp_tuple = (current_hill_x, current_hill_y, current_hill_z)

# Rekursio
return HillClimbing(image_data_3d, temp_tuple, scope)
```