

VENTSPILS AUGSTSKOLA
INFORMĀCIJAS TEHNOLOGIJU FAKULTĀTE

MAĢISTRA DARBS

**Zemes pārseguma klasifikācija, izmantojot
tālizpētes datus.**

Autors:
Ventspils Augstskolas
Informācijas tehnoloģiju fakultātes
maģistra studiju programmas ”Datorzinātnes”
2. kursa studente
Katrīna Zvaigzne
Matr. Nr. 15080010

(paraksts)

Fakultātes dekāns:
doc. Dr. phys. Māris Ēlerts

(paraksts)

Zinātniskā vadītāja:
Mg.sc.comp. Linda Gulbe

(paraksts)

Recenzents:

(Ienemamais amats, zinātniskais nosaukums, vārds, uzvārds)

(paraksts)

Ventspils, 2017

ANOTĀCIJA

Darba nosaukums: Zemes pārseguma klasifikācija, izmantojot tālizpētes datus.

Darba autors: Katrīna Zvaigzne

Darba vadītājs: mg. sc. comp. Linda Gulbe

Darba apjoms: 48 lpp., 2 tabulas, 24 attēli, 28 bibliogrāfiskās norādes.

Atslēgas vārdi: ORTOFOTOKARTES, ATTĒLU SEGMENTĀCIJA, VADĪTĀ KLASIFIKĀCIJA, DZIĻĀS APMĀCĪBAS NEIRONU TĪKLI.

Maģistra darbā ir apskatīta attēlu segmentācijas metode JSEG un vairākas citas eksistējošas metodes krāsu attēlu segmentācijā. Izanalizēti šo metožu trūkumi un priekšrocības salīdzinoši ar JSEG attēlu segmentācijas metodi. Papildus analizēta arī JSEG iekļautā segmentēto reģionu apvienošanas metode un atrastas alternatīvas tās aizvietošanai.

Autore izstrādājusi darbplūsmas ne tikai ortofotokaršu segmentēšanā izmantojot JSEG, bet arī implementējusi atrastās alternatīvas reģionu klasificēšanā. Lai noteiktu, kura no klasifikācijas metodēm sniedz labākus rezultātus, veikta klasificēto reģionu validācija gan klasifikācijai, izmantojot k-tuvāko kaimiņu metodi, gan, izmantojot neuronu tīklus.

Maģistra darba rezultātā veikta ortofotokartes segmentēšana izmantojot attēlu segmentēšanas algoritmu JSEG, izpētītas klasifikācijas metodes zemes pārseguma tipu noteikšanai augstas telpiskās izšķirtspējas multispektrālajos datos un izstrādātas darbplūsmas metožu pielietošanai.

ABSTRACT

Title: Land cover classification using remote sensing data.

Author: Katrina Zvaigzne

Supervisor: mg. sc. comp. Linda Gulbe

Volume: 48 pages, 2 tables, 24 images, 28 citations.

Key words: ORTHOPHOTOMAP, IMAGE SEGMENTATION, SUPERVISED CLASSIFICATION, DEEP LEARNING NEURAL NETWORKS.

This master's thesis examines JSEG and a number of other color image segmentation methods. An analysis on these methods was done and pros and cons of each of those methods was found. Additional analyzes also included JSEG segmented region merging method and found alternatives.

The author created a workflow not only for orthophotomap segmentation using JSEG but also implemented regional classification alternatives. To determine which of the classification methods gives better results, conducted classified region validation using the k-nearest neighbor method and using neural networks.

As a result of master's thesis, orthophotomap segmentation was performed using JSEG, methods for segmented region classification into land cover types were researched and created workflows were applied.

Saturs

Ievads	5
1. Izmantotie dati un pētāmais apgabals	7
1.1. Tālizpētes dati, attēls un tā izšķirtspējas veidi	7
1.1. 1. Attēls un tā izšķirtspējas veidi	7
1.2. Izmantotie dati	8
1.3. Pētāmais apgabals	8
2. Zemes pārseguma atpazīšana augstas izšķirtspējas datos	10
2.1. Klasiskā pieeja: segmentācija un vadītā klasifikācija	10
2.1. 1. Apmācības datu sagatavošana: deskriptoru rēķini	10
2.1. 2. JSEG un tā realizācija	17
2.1. 3. K-tuvāko kaimiņu metode un tās realizācija	27
2.2. Neironu tīklu pielietojums	30
2.2. 1. Apmācības datu sagatavošana	30
2.2. 2. Neironu tīkla teorētisks apskats	32
2.2. 3. Neironu tīkla izmantošana JSEG reģionu klasifikācijai	38
3. Validācija un rezultāti	43
3.1. JSEG segmentācijas rezultātu klasifikācijas ar k-tuvāko kaimiņu metodi validācijas rezultāti	44
3.2. JSEG segmentācijas rezultātu klasifikācijas, izmantojot neironu tīklus, validācijas rezultāti	45
Secinājumi un priekšlikumi	47
Izmantotās literatūras un avotu saraksts	51
Galvojums	52

Ievads

Zemes pārsegums apraksta Zemes virsmas fizisko pārkājumu un parasti to sagrupē tipos atkarībā no nepieciešamās informācijas, piemēram, zemes pārsegumu var iedalīt lauk-saimniecības platībās, ūdeņos, mežos, ceļos un apdzīvotās teritorijās. Zemes pārseguma tipu izšķiršana ir nozīmīga, lai būtu iespējams noteikt izmaiņas, ko izraisījušas kādas pārmaiņas dabā. [3]

Piemēram, zemes pārseguma tipu kartes ir noderīgas gan pirms ugunsgrēkiem, lai modelētu potenciālo risku, gan ugunsgrēka laikā, lai atbalstītu drošības personālu ugunsgrēku apkarošanā un evakuācijas organizēšanā, gan arī pēc ugunsgrēka, lai mazinātu ugunsgrēka ietekmi uz dabu. Vēl ir nozīmīgi izšķirt apdzīvotas teritorijas no veģetācijas plašā ģeogrāfiskā apgabalā, lai, balstoties uz ēku atrašanās vietām attiecībā pret viegli uzliesmojošām teritorijām, būtu iespējams novērst zaudējumus, kas var rasties savvaļas ugunsgrēku dēļ. [5]

Nepārtrauktā vides mainība ir viens no iemesliem, kādēļ zemes pārseguma tipu kartes ātri noveco un kļūst nederīgas. Aktuāla informācija par interesējošo reģionu ir nepieciešama lielu zemes platību īpašniekiem un apsaimniekotājiem, kam fiziska platību apzināšana ne vienmēr ir un arī patēri ļoti daudz laika. Līdzīga situācija ir māklieriem un dažādiem īpašumu pārpircējiem, kam ne vienmēr ir interese fiziski un padziļināti apsekot katru nekustamo īpašumu.

Ortofotokartes sadalīšana zemes pārseguma tipos notiek divos soļos - attēla segmentēšana un segmentācijas rezultātu klasifikācija zemes pārseguma tipos. Džollija un Gupta savā pētījumā *"Color and Texture Fusion: Application to Aerial Image Segmentation and GIS Updating [15]"* krāsu aerofotogrāfiju segmentēšanu veic, izmantojot maksimālās ticamības segmentēšanu (angl. *maximum likelihood segmentation*). Lai arī šī metode sniedz labus rezultātus, tās ierobežojumi saistāmi ar spēju labi segmentēt tikai viendabīgas tekstūras vai krāsas attēlus. [15]

Klīvs un citi savā pētījumā *"Classification of the wildland–urban interface: A comparison of pixel- and object-based classifications using high-resolution aerial photography [5]"* veic zemes pārseguma tipu klasifikāciju, izmantojot divas metodes. Pirmā pieeja ir balstīta uz atsevišķu pikselu klasifikāciju, izmantojot ISODATA nevadītās klasifikācijas metodi, bet otrā uz atsevišķu segmentētu reģionu klasifikāciju, izmantojot fuzzy un *tuvāko kaimiņu* vadītās klasifikācijas metodes. Otrajā pieejā aerofotogrāfijas segmentēšana notiek, izmantojot aplikāciju *Definiens eCognition*. [5]

Attēlu segmentēšana ir nozīmīga tālizpētes, attēlu analīzes un tēlu pazīšanas (LZA, angl. *pattern recognition*) daļa. Tieši no segmentācijas rezultātiem iespējams iegūt nozīmīgu informāciju par interesējošiem reģioniem vai objektiem attēlā, kas noder turpmākai attēlu analīzei.[2]

Tā kā zemes pārsegums ir ļoti dažāds kā krāsas, tā tekstūras ziņā, bet lielākā daļa segmentēšanas algoritmu labi strādā ar homogēniem krāsu reģioniem vai tiem nepieciešami

precīzi tekstūras novērtējumi, kas ne vienmēr ir viegli nosakāmi, nepieciešami automātiski segmentēšanas algoritmi, kas ir spējīgi tikt galā ar dažādiem scenārijiem attēlā.[2],[7],[8]

Kā risinājums vairākos literatūras avotos ([2], [7], [8], [23], [26]) tiek piedāvāts JSEG algoritms ar vai bez dažādām modifikācijām.

Segmentācijas algoritmi, tajā skaitā JSEG, attēlā identificē krāsas un tekstūras ziņā līdzīgus reģionus. Nākamais solis pēc segmentācijas ir reģionu klasifikācija, kura identificē konkrētu reģiona zemes pārseguma tipu. Balstoties uz vairākos pētījumos pausto informāciju, ka ne vienmēr JSEG iekļautais apvienošanas algoritms veiksmīgi veic segmentēto reģionu apvienošanu jēgpilnās reģionu grupās, tika pieņemts lēmums apvienot JSEG segmentēšanas metodi ar vadītās klasifikācijas k-tuvāko kaimiņu (LZA (Latvijas Zinātņu Akadēmija), angl. *k-nearest neighbors*) metodi un arī veikt klasifikāciju, izmantojot neironu tīklus.

Maģistra darba mērķis ir izpētīt klasifikācijas metodes zemes pārseguma tipu noteikšanai ļoti augstas telpiskās izšķirtspējas multispektrālajos datos un izstrādāt darbplūsmas metožu pielietošanai.

Lai sasniegtu maģistra darba mērķi, tika izvirzīti sekojoši uzdevumi:

- veikt ortofotokartes segmentāciju, izmantojot JSEG segmentēšanas metodi;
- JSEG segmentēšanas gaitā veikt tādu parametru, kā pudurošanas pēc vidējiem (LZA, angl. *kmeans*) parametru k , mazākā loga izmēra, ortofotokartes fragmenta izmēru testēšanu, lai iegūtu eksperimentāli labākos rezultātus;
- izpētīt metodes ortofotokaršu segmentācijas rezultātu klasifikācijā;
- sagatavot parauga datus;
- izveidot validācijas datu komplektu;
- klasificēt segmentācijas rezultātā iegūtos reģionus, izmantojot k-tuvāko kaimiņu metodi, un veikt rezultātu validāciju;
- klasificēt segmentācijas rezultātā iegūtos reģionus, izmantojot neironu tīklus, un veikt rezultātu validāciju;
- salīdzināt abu klasifikācijas metožu rezultātus un veikt secinājumus.

Maģistra darbs sastāv no ievada, trīs nodaļām, secinājumiem un priekšlikumiem. Pirmajā nodaļā apskatīti izmantotie dati un pētāmais apgabals, kā arī definēts tālizpētes attēls un tā izšķirtspēju veidi. Otrajā nodaļā apskatīti risinājumi zemes pārseguma atpazīšanai augstas telpiskās izšķirtspējas multispektrālajos datos un izstrādātas darbplūsmas risinājumu pielietošanai. Trešajā nodaļā tiek veikta klasificēto segmentācijas rezultātu validēšana un apkopoti rezultāti.

Pētījums tika realizēts, izmantojot salīdzinājuma pētniecības (LZA, angl. *comparative study*) un izmēģinājuma studijas metodes. Pētījumā tika veikta literatūras avotu analīze un pētījuma praktiskā daļa tika veikta, izmantojot programmatūras *MatLab*, *QGIS*, programmēšanas valodu *Python* un bibliotēku *TensorFlow*.

1.Izmantotie dati un pētāmais apgabals

1.1. Tālizpētes dati, attēls un tā izšķirtspējas veidi

Tālizpēte dažādos literatūras avotos ir definēta atšķirīgi, tomēr visām definīcijām ir kopīga iezīme, ka tālizpēte ir informācijas ieguve no attāluma bez tieša kontakta ar pētāmo objektu. Šāda definīcija ir ļoti vispārīga un gadījumos, kad tālizpētes metodes tiek izmantotas, lai iegūtu vispārīgu informāciju par Zemes pārsegumu, definīcija tiek precīzēta uz informācijas ieguvī, izmantojot atstaroto vai izstaroto elektromagnētisko enerģiju.

Kempbels un Vains savā grāmatā *Introduction to Remote Sensing* definē: "Tālizpēte ir informācijas iegūšanas prakse par Zemes pārsegumu, izmantojot attēlus, kas iegūti no satelītu vai aero platformām, un, kuros reģistrēts Zemes virsmas izstarotais un/vai atstarotais elektromagnētiskais starojums (LZA, angl. *electromagnetic radiation*) vienā vai vairākos elektromagnētiskā spektra reģionos [4]."

1.1. 1. Attēls un tā izšķirtspējas veidi

No tālizpētes definīcijas izriet, ka biežāk izmantotais datu formāts ir attēls. Attēls ir divdimensionāla funkcija $f(x, y)$, kur x un y ir plaknes koordinātes, bet amplitūdu katrā punktā (x, y) sauc par attēla intensitāti vai pelēkā līmeni šajā punktā. Attēla ieguvē nozīmīgi ir tādi divi procesi kā diskretizācija un digitalizācija. Diskretizācija praktiski ir signāla telpiska sadalīšana konkrēta skaita iedaļās, ko vispārīgā gadījumā nosaka pusvadītāju sensoru skaits. Digitalizācija ir signāla intensitātes sadalīšana galīgās vienībās. Piemēram, ienākošo signālu iespējams digitalizēt divās vienībās izmantojot 1 un 0, kur ar 1 apzīmē tuvāk baltam esošos toņus, bet ar 0 tumšos toņus. Lai informācija, kas iegūstama no attēla, tiktu kvalitatīvi reprezentēta, jāpievērš liela uzmanība informācijas nesēja - attēla kvalitātei. Tālizpētes attēlu kvalitāti iespējams raksturot, izmantojot četru veidu izšķirtspējas:

- Telpiskā izšķirtspēja - tā ir izšķirtspēja, kas bieži veido primāro asociāciju ar šo terminu. Telpiskā izšķirtspēja nosaka to, cik liels ir mazākais objekts, kurš ir saskatāms attēlā. Realitātē telpisko izšķirtspēju izsaka kā attēla piksela dabiskos izmērus dabā. Piemēram, ar telpisko izšķirtspēju viens metrs iespējams pat izšķirt automašīnas, koku vainagus vai pat dažādus dzīvniekus, bet ar telpisko izšķirtspēju 30 metri šādas detaļas saplūst ar apkārtējo vidi.
- Spektrālā izšķirtspēja raksturo elektromagnētiskā spektra daļas platumu nanometros (vai mikrometros), kuru uztver sensors. Šo elektromagnētiskā starojuma daļu sauc arī par kanālu vai frekvenču joslu. Esot augstākai spektrālai izšķirtspējai, iespējams uztvert mazāku vilņa garumu diapazonu. Multispektrālie sensori ievāc katrai frekvenču joslai atsevišķu pelēko toņu attēlu. Apvienojot zilā, sarkanā, zaļā un tuvā infrasarkanajā joslā iegūtos attēlus, iegūst dabisko krāsu attēlu.
- Radiometriskā izšķirtspēja - kā jau iepriekš tika minēts, digitalizācija pārveido attēlu galīga skaita vienību toņos. Tieši šo toņu skaits arī ir radiometriskā izšķirtspēja un to

raksturo bitu dziļums jeb skaitļa divi pakāpe, kas nepieciešama, lai iegūtu nepieciešamo toņu skaitu.

- Temporālā izšķirtspēja nosaka to, cik bieži pieejami konkrētās ģeogrāfiskās vietas attēli. [13]

1.2. Izmantotie dati

Maģistra darbā izmantota Latvijas Geotelpiskās Informācijas Aģentūras (LGIA) ortofotokarte. Ortofotokarte ir zemes virsmas attēls, kas izgatavots no aerofotogrāfijām un kam, izmantojot speciālas, šim mērķim paredzētas datorprogrammas, novērsti visi sagrozījumi, kas rodas fotografejot reljefa un citu faktoru ietekmē. [18]

Aerofotogrāfijas, no kā iegūta ortofotokarte, fotografētas 5. ciklā 2013. gada jūnijā, izmantojot 4 lidojumus - 1. un 2. jūnijā, 21. jūnijā, 23. jūnijā un 30. jūnijā. Ortofotokartes ir pieejamas *TIFF* formātā un tās ir piesaistītas LKS-92 TM koordinātu sistēmai. [17]

Ortofotokarte (redzama 1.1. attēlā) ir kvadrātiska ar izmēru $10\,000 \times 10\,000$ pikseli, kas dabā aptver $16\,km^2$ lielu teritoriju jeb vienas dimensijas garums ir 4 kilometri. Ortofotokartes telpiskā izšķirtspēja ir 0,4 metri uz pikseli. Ortofotokartes spektrālā izšķirtspēja ir redzamās gaismas RGB joslas.



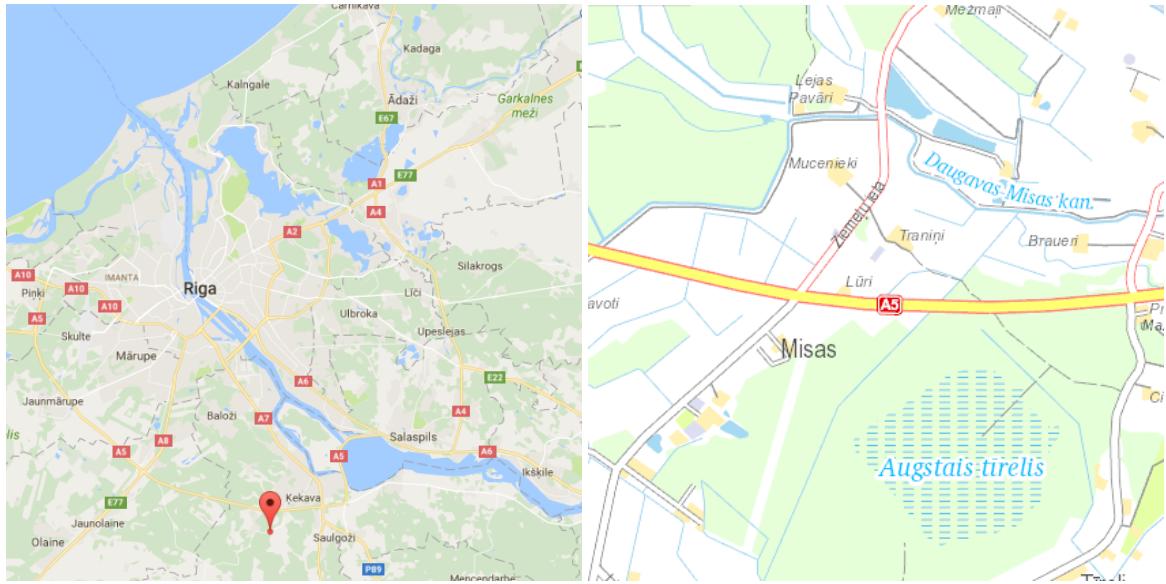
Att. 1.1. Maģistra darbā izmantotā ortofotokarte mērogā 1:10 000 © Latvijas Geotelpiskās informācijas aģentūra, 2013.

Maģistra darba īstenošanai, kā parauga dati tiek izmantoti tās pašas ortofotokartes fragmenti, kura tiek segmentēta un klasificēta zemes pārseguma tipos.

1.3. Pētāmais apgabals

Pētāmais apgabals ir $16\,km^2$ plaša teritorija Ķekavas novadā, dienvidrietumu virzienā no Ķekavas un aptver Latvijai raksturīgu piepilsētas teritoriju. Pētāmais apgabals ir ļoti

daudzveidīgs un ietver aramzemju platības, pļavas, kanālus, dīķus, viensētas, ciematu ”Jaunsils”, plašas mežu platības kā arī Augsto Tīreli pētāmā apgabala dienvidu un dienvidaustrumu daļā. Pētāmā apgabala atrašanās vieta redzama 1.2.a attēlā. 1.2.b attēlā redzams kartes fragments, kas parāda, kādi vides objekti sastopami pētāmajā apgabalā.



(a) Pētāmā apgabala atrašanās vieta © Google, (b) Ķekavas apkārtnes karte © Valsts zemes dienests, pirmpublicējuma gads 1999-2012.

Att. 1.2. Pētāmā apgabala ģeogrāfiskais novietojums.

2.Zemes pārseguma atpazīšana augstas izšķirtspējas datos

2.1. Klasiskā pieeja: segmentācija un vadītā klasifikācija

Klasiskā pieeja ortofotokartes sadalīšanai zemes pārseguma tipos sastāv no ortofotokartes segmentācijas un rezultātā iegūto segmentācijas reģionu klasifikācijas izmantojot kādu no mašīnmācīšanās metodēm. Šajā gadījumā segmentāciju veic, izmantojot krāsu attēlu segmentācijas metodi JSEG, bet rezultātus klasificē zemes pārseguma tipos, izmantojot vadītās klasifikācijas k-tuvāko kaimiņu metodi. Vadītā klasifikācija paredz, ka metode izmanto iepriekš sagatavotus parauga datus.

2.1. 1. Apmācības datu sagatavošana: deskriptoru rēķini

Deskriptori un tēli

Pēc tam, kad attēls ir sadalīts reģionos, nepieciešams izvēlēties un aprēķināt reģionu skaitiskās īpašības jeb deskriptorus, kas tiek izmantoti reģionu aprakstīšanai. To ir iespējams izdarīt, izmantojot reģiona ārējās raksturīgās īpašības, kā piemēram, reģiona robežu vai tā iekšējās īpašības, kā piemēram, pikseļu vērtību raksturielumus, ko satur reģions. [13]

Ārējos reģiona raksturielumus izmanto, kad interesējošajam reģionam ir raksturīga specifiska forma, bet iekšējās raksturīgās īpašības izmanto, kad interesē reģiona krāsa un tekstūra. Reizēm nepieciešamā rezultāta sasniegšanai jāizmanto gan ārējās, gan iekšējās raksturīgās īpašības. Uzmanība jāpievērš tam, vai pazīmju vērtības (LZA, angl. *feature*), kas tiek izvēlētas kā deskriptori (LZA, angl. *descriptor*), būtiski neietekmē tādi pārveidojumi kā izmēra izmaiņas, rotācija un translācija. [13]

Tālāk apskatītas dažādas pieejas, kā raksturot attēla reģionu. Divi no reģiona vienkāršākajiem deskriptoriem ir tā platība (angl. *area*), ko raksturo pikseļu skaits reģionā, un perimetrs jeb reģiona robežas garums pikselos. Atsevišķi šos deskriptorus attēla reģiona raksturošanai izmanto tikai tādos gadījumos, ja reģiona izmēri un forma nemainās. Biežāk tiek izmantots tāds deskriptors, kā kompaktums (angl. *compactness*). Kompaktumu izsaka, kā perimetra kvadrāta dalījumu ar platību ($(\text{perimetrs}^2)/\text{platība}$). Kompaktums ir bezdimensjonāls lielums, kas ir mazs diskā formas reģioniem. Citi vienkāršie deskriptori ir reģiona pelēko tonu vidējā (angl. *mean*) vērtība un mediāna (angl. *median*), reģiona maksimālā un minimālā pelēkā tonā vērtība, kā arī to pikseļu skaits, kas reģionā atrodami virs un zem reģiona vidējās vērtības. [13]

Tēls ir deskriptoru vektors. Bieži tēlu pazīšanā tiek izmantots vārds "pazīme", ko attiecina uz deskriptoriem. Tēlu klase (LZA, angl. *pattern class*) ir vairāku tēlu kopa, kam raksturīgas līdzīgas deskriptoru vērtības. Tēlu klases apzīmē ar $\omega_1, \omega_2, \dots, \omega_W$, kur W apzīmē klašu skaitu. Automātiskā tēlu pazīšana sevī iekļauj tādas metodes, kur katrs tēls tiek piešķirts kādai klasei ar pēc iespējas mazāku iejaukšanos no cilvēka pusēs. Tēlu vektorus apzīmē ar

treknrakstā izceltiem, mazajiem alfabēta burtiem un pieraksta formā

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \quad (2.1.)$$

kur katrs loceklis x_i apzīmē i -to deskriptoru un n ir kopējais deskriptoru skaits. Tēla vektors ir attēlots kā kolonas matrica ar izmēru $n \times 1$ un līdz ar to, to iespējams izteikt arī kā $\mathbf{x} = (x_1, x_2, \dots, x_n)^T$, kur T apzīmē transponēšanas operāciju. [13]

Darbā izmantotie deskriptori

Darbā kā deskriptori tiks izmantoti attēla reģiona vidējā vērtība, mediāna un standartnovirze. Katrs no šiem deskriptoriem arī turpinājumā tiek apskatīts detalizētāk.

Vidējā vērtība (LZA, angl. *mean value*) attēlam ir aritmētiskā vidējā vērtība, un tā tiek definēta, kā visu spilgtuma vērtību summas dalījums ar apskatīto pikseļu skaitu. Attēla reģiona vidējā vērtība ir populārs raksturlielums. Vidējo vērtību μ_k atsevišķai krāsu joslai, kas sastāv no n pikseļu vērtībām $BV_{i,k}$ aprēķina, izmantojot formulu:

$$\mu_k = \frac{\sum_{i=1}^n BV_{i,k}}{n} \quad (2.2.)$$

Lai attēlam aprēķinātu standartnovirzi (LZA, angl. *standard deviation*), nepieciešams vispirms aprēķināt attēla dispersiju (LZA, angl. *variance*). Attēla dispersija ir vidējā kvadrātiskā novirze visām pikseļu vērtībām no attēla vidējās vērtības. To aprēķina, izmantojot formulu:

$$var_k = \frac{\sum_{i=1}^n (BV_{i,k} - \mu_k)^2}{n} \quad (2.3.)$$

Attēla standartnovirze ir kvadrātsaknes vērtība no dispersijas un to aprēķina, izmantojot formulu:

$$s_k = \sqrt{var_k} \quad (2.4.)$$

Maza standartnovirze kādā attēla reģionā raksturo to, ka vērtības šajā reģionā ir tuvu reģiona vidējai vērtībai un vienas otrai. Turpretī, liela standartnovirzes vērtība nosaka to, ka pikseļu vērtības reģionā ir izkliedētas un atrodas tālu no vidējās vērtības. [11]

Attēla mediāna ir vērtība skaitļu virknē pēc pozīcijas, kad visas pikseļu vērtības ir sarindotas augošā secībā [11].

Praktiskā realizācija: datu sagatavošana un deskriptoru aprēķināšana

Koordinātu pārrēkināšana reģionu failiem Lai veiktu deskriptoru aprēķinus iepriekš sagatavotajiem parauga datiem, parauga datu maskas nepieciešams sareizināt ar segmentācijas rezultātā iegūtajiem reģioniem. Lai izmantotu *QGIS* spraudni deskriptoru rēķiniem, *Mat-*

lab vidē sagatavoto attēlu koordinātu sistēmai ir nepieciešams piesaistīt kartes projekcijas koordinātes. Šādu koordinātu pārrēķināšanu nepieciešams arī veikt segmentācijas rezultātā iegūtajam reģionu failam.

Koordinātu sistēmu pārrēķina, izmantojot *Python* skriptu. Skriptā vispirms importē bibliotēkas *gdal* un *osr*, kas nepieciešamas, lai veiktu koordinātu pārrēķināšanu, kā arī papildus funkcijas no faila *additionalFunctions* (funkciju klase, kas ir izstrādāta VeA IZI “VSRC” projektu ietvaros), kurām turpmāk skriptā varēs piekļūt, izmantojot klases objektu *af*.

```
1 import gdal
2 import osr
3 from additionalFunctions import additionalFunctions
4
5 af = additionalFunctions()
```

Pēc tam uzstāda mainīgā *origPath* vērtību, norādot pilno ceļu, kur atrodama aerofotogrāfija, no kuras iespējams nolasīt ar kartes projekciju saistīto informāciju, *imPath* ar pilno ceļu uz to direktoriju, kurā atrodami faili, kam nepieciešams mainīt koordinātu sistēmu (šajā gadījumā faili ar reģionu matricām) un *names*, kas ir to failu nosaukumu masīvs, kam nepieciešams veikt koordinātu pārrēķināšanu.

```
1 origPath = 'C:\Path\to\\aerophotography.tif'
2 imPath = 'C:\Path\to\region\folder\\'
3 names = [ 'citsRegion.tif', 'lauksRegion.tif', 'udensRegion.tif', ...
           'zaliensRegion.tif', 'kokiRegion.tif', 'region.tif' ]
```

Tālāk, ciklā ejot cauri reģionu failiem, fails tiek atvērts un mainīgajā *imb* tiek saglabāta attēla 1. rastra (LZA, angl. *raster*) josla. Tad šī iegūtā rastra josla tiek ielasīta mainīgajā *im* kā masīvs. Tālāk rīkojas ar oriģinālo aerofotogrāfiju to ielasot mainīgajā *imd1* un iegūstot tās ar kartes projekciju saistīto informāciju, kas tiek saglabāta mainīgajā *geot*.

```
1 for n in names:
2     imd = gdal.Open(imPath + n)
3     imb = imd.GetRasterBand(1)
4     im = imb.ReadAsArray()
5     imd1 = gdal.Open(origPath)
6     geot = imd1.GetGeoTransform()
```

Pēc tam, izmantojot *osr* pakotnes klasi *SpatialReference* tiek izveidots klases objekts ar nosaukumu *Projection*, kas tiek aizpildīts, izmantojot projekcijas atsauci no oriģinālās aerofotogrāfijas. Tālāk, izmantojot iepriekš aizpildītās projekcijas, tiek iegūta informācija par EPSG ģeodēzisko parametru datu kopu. Izmantojot visu iepriekš iegūto informāciju tiek izveidots jauns fails, kas tiek saglabāts tajā pašā direktorijā, kur reģiona fails, kam bija nepieciešams mainīt koordinātas.

```

1   Projection = osr.SpatialReference()
2   Projection.ImportFromWkt(imd1.GetProjectionRef())
3   EPSG = (Projection.GetAttrValue("AUTHORITY", 1))
4   rasterOrigin = [geot[0], geot[3]]
5   af.array2raster(imPath + 'G' + n, rasterOrigin, geot[1], ...
                  geot[5], im, 1, gdal.GDT_UInt16, 3059)

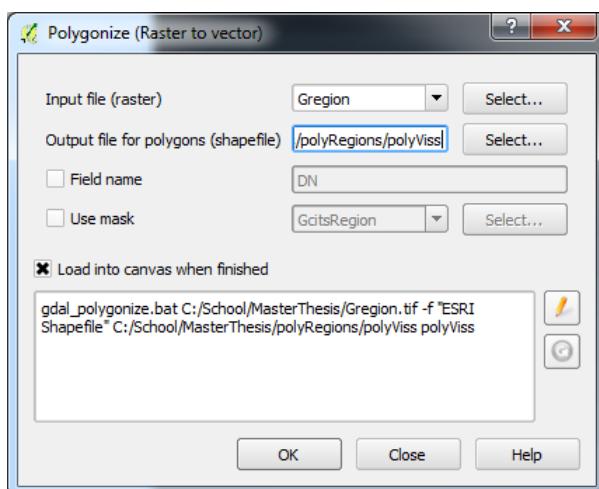
```

Lai veiktu projekciju pārrēķināšanu un veiksmīgi importētu nepieciešamās pakotnes, skriptu, izmantojot tā pilno celiņu, nepieciešams izpildīt no konsoles. Konsoli ar visām, vaja-dzīgajām importētajām bibliotēkām var atvērt izpildot C:\Program Files\QGIS 2.14\OSGeo4W.bat failu.

Deskriptoru aprēķināšana, izmantojot QGIS programmatūru Kad koordinātas pārrēķinātas visiem parauga datu reģionu failiem un segmentācijas rezultātā iegūtajam reģionu failam, tad iespējams aprēķināt deskriptorus, izmantojot programmatūru *QGIS*. Darbā tiek izmantota *QGIS 2.14.11-Essen* versija. Lai veiktu reģiona faila poligonizāciju, interesējošais fails tiek ielasīts *QGIS Browser Panel* logrīkā, veicot dubultklikšķi uz tā reģiona faila nosaukuma, ko nepieciešams ielasīt.

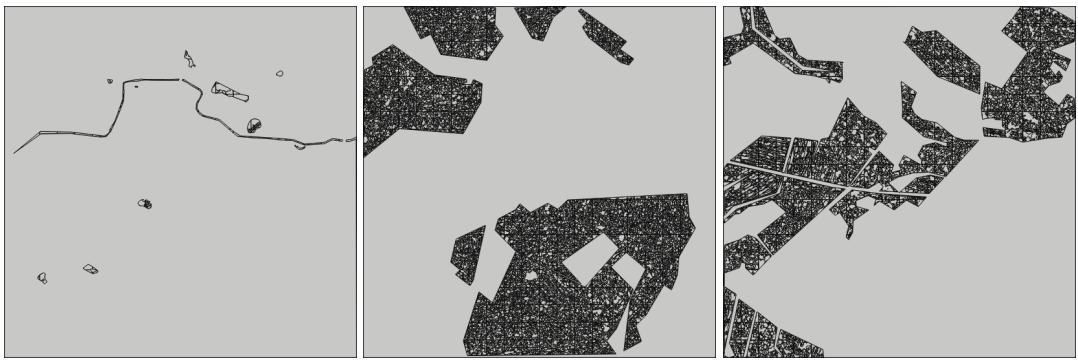
Attēla poligonizācija ir attēla sadalīšana dažādas formas un izmēru poligonos, veidojot poligonu tīklu. Katrā poligonā esošos pikselus vieno kāda pazīme (šajā gadījumā reģiona numurs, kam pikseļi pieder). [14]

Tālāk *Raster → Conversion → Polygonize* tiek atvērts poligonizācijas rīks, kur *Input file* ir ielasītais reģionu fails, kam nepieciešams veikt poligonizāciju, bet pie *Output file for polygons*, klikšķinot uz "Select" pogu atveras logs jaunas mapes izveidei, kur tiks uzglabāti poligonizācijas rezultāti. Kā redzams attēlā 2.1., pārējās opcijas un mainīgos atstājam nemainītus un, spiežot "OK", apstiprinām izvēli.

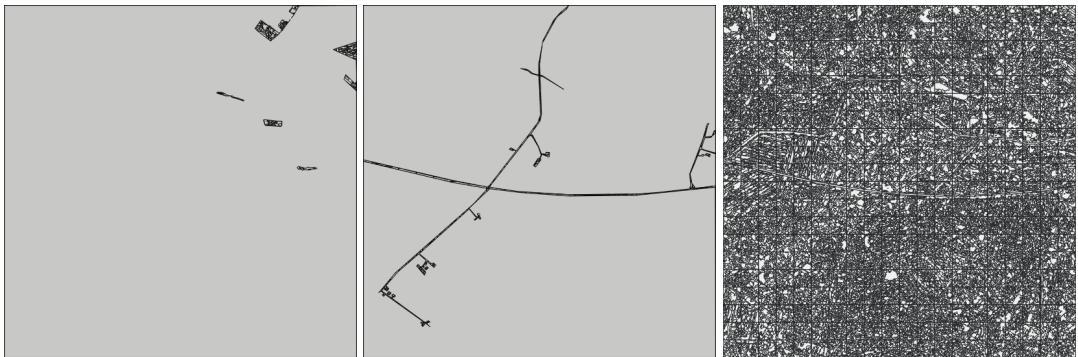


Att. 2.1. Rīks attēla poligonizācijai.

Iepriekš aprakstītās darbības tiek veiktas visiem reģionu failiem un gala rezultātā tiek iegūti 6 poligonu faili, kas redzami 2.2. attēlā.



(a) Poligonu attēls ūdens pārauga datiem. (b) Poligonu attēls koku pārauga datiem. (c) Poligonu attēls zāliena pārauga datiem.



(d) Poligonu attēls lauku pārauga datiem. (e) Poligonu attēls cita parau- ga datiem. (f) Poligonu attēls visiem reģioniem.

Att. 2.2. Rezultāti pēc poligonizācijas.

Izmantojot *Plugins → Python Console*, augšējā joslā tiek nospiesta poga "Show Editor", lai atvērtu rīku *Python* koda ievadīšanai. Šajā teksta logā ar pirmo komandu importējam pakotni *QgsZonalStatistics*, kam pēc tam seko nodefinētas atrašanās vietas direktorijai ar poligonizācijas rezultātiem, poligonizācijas failu nosaukumiem un originālajam attēlam, kā arī masīvs ar joslas apzīmējošiem burtiem. Tālāk ciklā katram poligonu failam ielasa tā poligonu slāni un ejot ciklā cauri katram attēla slānim aprēķina slānī esošo reģionu izmēru pikselos, vidējo vērtību, mediānu un standartnovirzi.

Skripta augšpusē, nospiežot pogu "Run script", izpilda skriptu, kas pēc veiksmīgas skripta izpildes, deskriptorus saglabā atribūtu tabulā katram poligonu slānim.

```

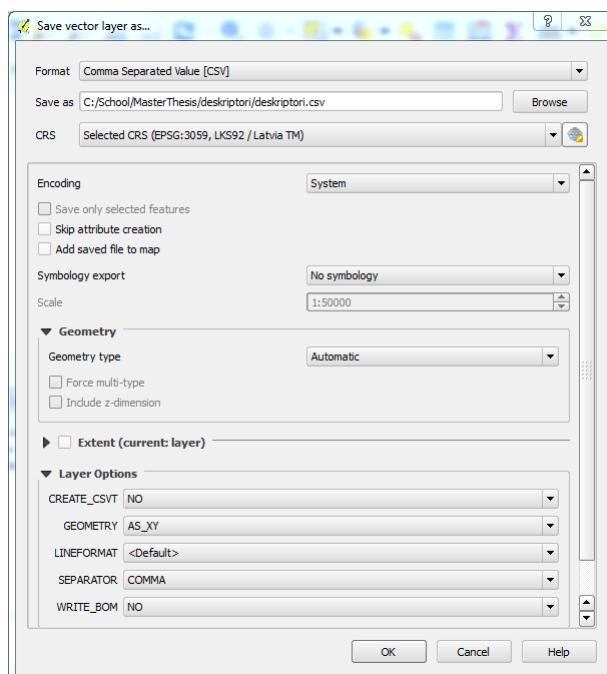
1 from qgis.analysis import QgsZonalStatistics
2
3 folder = 'C:/ School/MasterThesis/polyRegions/'
4 pathes = [ 'polyCits/polyCits.shp' , 'polyKoki/polyKoki.shp' , ...
            'polyLauks/polyLauks.shp' , 'polyUdens/polyUdens.shp' , ...
            'polyZaliens/polyZaliens.shp' , 'polyViss/polyViss.shp' ]
5 rasterFilePath = 'C:/ School/3333-53_1.tif'
6 bands = [ "R" , "G" , "B" ]
7
8 for path in pathes:
9     polygonLayer = QgsVectorLayer( folder+path , 'polygon' , "ogr")
```

```

10     for i in range(1,4):
11         zoneStat = QgsZonalStatistics (polygonLayer, ...
12             rasterFilePath, bands[i-1], i, QgsZonalStatistics.Count ...
13             | QgsZonalStatistics.Mean | QgsZonalStatistics.Median | ...
14             QgsZonalStatistics.StDev)
15         zoneStat.calculateStatistics(None)

```

Lai šos rezultātus saglabātu *csv* failos, ar labo peles pogu uzspiežot uz interesējošā poligona faila, atver izvēlni, kur izvēlas opciju *Save As...*, kas atver logrīku vektoru slāņa saglabāšanai. Iestatījumus atstāj gandrīz nemainītus, izņemot opciju faila nolasīšanai, kā redzams 2.3. attēlā.



Att. 2.3. Logrīks deskriptoru saglabāšanai *csv* formātā.

Deskriptoru matricu pēcapstrāde Vispirms *csv* failus atver kādā darba grāmatu apstrādes programmā un izdzēš kolonas *GCount* un *BCount*. Pēc tam, kad *MatLab* ielasīti *csv* faili *table* formātā, ielasītās deskriptoru tabulas pārvērš matricas formātā, izmantojot *MatLab* iebūvēto funkciju *table2array*. Pēc tam šo masīvu, kā parametru, padod funkcijai *removeDuplicates*, kas izdzēš tās rindiņas, kas manuālas datu atlases rezultātā veidojušas reģionu atgriezumus un, kuru deskriptori var būt sagrozītas vērtības mazo reģionu izmēru dēļ.

Vispirms funkcija *removeDuplicates* atsevišķā matricā atlasa tikai tās rindiņas ar reģioniem, kur reģiona izmērs ir lielāks par 5 pikseliem. Tad no šiem izmēra ziņā derīgajiem reģioniem nosaka unikālās reģionu identifikatoru vērtības. Ja iepriekš atlasītajā matricā, ciklā ejot cauri reģionu identifikatoriem, ir vairāk kā viena rindiņa ar vienu identifikatoru, tad šī reģiona identifikatoru pievieno matricai *duplicateIds*.

```

1 function result = removeDuplicates(deskriptori)

```

```

2      j=1;
3      for i=1:length(deskriptori)
4          if (deskriptori(i,2)>5)
5              tempDeskriptori(j,:)=deskriptori(i,:);
6              j=j+1;
7          end
8      end
9      uniqueIds=unique(tempDeskriptori(:,1));
10     j=1;
11     for i=1:length(uniqueIds)
12         if sum(tempDeskriptori(:,1)==uniqueIds(i))>1
13             duplicateIds(j)=uniqueIds(i);
14             j=j+1;
15         end
16     end

```

Pēc tam ciklā, ejot cauri to reģionu identifikatoriem, kas masīvā atkārtojas, atlasa tās rindiņas, kam ir viens reģiona identifikators un izveido pagaidu masīvu, kur uzglabā šo atkārtojošo rindiņu reģionu identifikatorus un izmērus. No tikko aizpildītā pagaidu masīva atrod to reģiona daļu, kam ir vislielākais izmērs un tā rindiņu ar visiem deskriptoriem ievieto gala rezultāta masīvā. Pašās funkcijas beigās no gala rezultāta masīva izdzēš kolonnu ar reģionu izmēriem.

```

1      k=1;
2      for i=1:length(duplicateIds)
3          duplicateRows=find(tempDeskriptori(:,1)==duplicateIds(i));
4          temp=zeros(length(duplicateRows),2);
5          for j=1:length(duplicateRows)
6              temp(j,1)=duplicateRows(j);
7              temp(j,2)=tempDeskriptori(duplicateRows(j),2);
8          end
9          maxCount=max(temp(:,2));
10         for j=1:length(duplicateRows)
11             if temp(j,2)==maxCount
12                 result(k,:)=tempDeskriptori(duplicateRows(j),:);
13                 k=k+1;
14             end
15         end
16     end
17     result(:,2)=[];

```

Iepriekš aprakstītās darbības izpilda visiem *csv* failiem, kas satur sevī deskriptorus parauga datiem un visiem reģioniem. Kad iegūtas visas tabulas bez dublikātu rindiņām vienam un tam pašam reģionam, parauga datu deskriptoru matricas apvieno, izmantojot *MatLab* iebūvēto funkciju *vertcat*, ievērojot secību. Secība ir būtiska, jo nepieciešams izveidot kolonnas vektoru ar klašu atzīmēm parauga datiem. Klašu atzīmes ir skaitļi, kas nosaka kāds

skaitlis atbilst katrai klasei. Pēc tam, kad klašu atzīmju vektors ir izveidots, var veikt pēdējās darbības gan parauga datu deskriptoriem, gan visu reģionu deskriptoru matricām - atbrīvoties no reģionu identifikatoru kolonnas un normalizēt deskriptorus, izmantojot *MatLab* iebūvēto funkciju *zscore*.

```

1 sampleDescr=vertcat( udensDescr , kokiDescr , zaliensDescr , lauksDescr , ...
   citsDescr ) ;
2
3 a=length( udensDescr ) ;
4 b=a+length( kokiDescr ) ;
5 c=b+length( zaliensDescr ) ;
6 d=c+length( lauksDescr ) ;
7
8 class_labels=1:5;
9 sampleClasses(1:a)=class_labels(1) ;
10 sampleClasses(a+1:b)=class_labels(2) ;
11 sampleClasses(b+1:c)=class_labels(3) ;
12 sampleClasses(c+1:d)=class_labels(4) ;
13 sampleClasses(d+1:length( sampleDescr ))=class_labels(5) ;
14 sampleClasses=sampleClasses' ;
15
16 sampleDescr (:,1) =[] ;
17 vissDescr (:,1) =[] ;
18
19 sampleDescr=zscore( sampleDescr ) ;
20 vissDescr=zscore( vissDescr ) ;

```

2.1. 2. JSEG un tā realizācija

JSEG metodes teorētiskais apskats

JSEG jeb J-segmentācija ir krāsu attēlu segmentācijas algoritms, kas reģionu atrašanas procesā izmanto gan krāsas, gan tekstūras informāciju [7]. Attēlu segmentācija datorrežē ir digitālā attēla sadalīšanas process vairākos segmentos jeb pikseļu kopās, savienojot kaimiņpikselus ar līdzīgām raksturīpašībām. Segmentācijas mērķis ir vienkāršot vai mainīt attēla reprezentāciju formā, kas ir daudz nozīmīgāka un vieglāk analizējama. [21], [5]

Pats algoritms sastāv no pieciem soļiem, kur katrs no tiem tiks detalizēti aprakstīts tālākajos punktos.

Krāsu attēlu kvantizācija Lai atšķirtu līdzīgos krāsu reģionus no citiem, vispirms attēlu nepieciešams kvantizēt jeb samazināt tajā esošo krāsu skaitu. Krāsu attēlu kvantizācija ir process, kurā tiek samazināts atšķirīgo toņu skaits attēlā, ar mērķi, lai jaunais attēls būtu maksimāli līdzīgs iepriekšējam attēlam. Šādā veidā netiek būtiski ietekmēta attēla vizuālā kvalitāte, kā arī netiek zaudēta vērtīgā informācija, bet attēls tiek pārveidots tādā formā, ka to reprezentē tikai pāris krāsu. Krāsu attēliem, kas satur dabas ainavas, parasti izvēlas 10-20 krāsas, kas salīdzinoši efektīvi spēj reprezentēt attēlā redzamo krāsu gammu.[7]

Pēc krāsu kvantizācijas katram attēla pikselim tiek piešķirts kvantizācijas rezultātā atrastās krāsas numurs. Krāsu klase ir visu to pikseļu kopa, kam piešķirta konkrētā kvantizētā krāsa. Pēc tam tiek izveidota krāsu tabula (LZA, angl. *color map*), kur katrs attēla pikselis tam atbilstošajās koordinātās (x, y) krāsu tabulā tiek aizstāts ar atzīmi, kurai krāsu klasei tas pieder.[7]

Pareiza krāsu kvantizācija ir ļoti nozīmīga segmentācijas sastāvdaļa, kas var ietekmēt galarezultātu.

Labi segmentēta attēla nosacījums Tieki ieviesta vērtība J , kuru definē tālāk apskatītās formulas. Vispirms pieņemam, ka Z ir N punktu kopa no krāsu kartes, kas sastāv no punktiem $z = (x, y)$, $z \in Z$ un, ko raksturo vidējā vērtība m ,

$$m = \frac{1}{N} \sum_{z \in Z} z \quad (2.5.)$$

Pieņem, ka Z sastāv no C klasēm Z_i , $i = 1, \dots, C$. Tādā gadījumā m_i ir N_i punktu vidējā vērtība klasē Z_i ,

$$m_i = \frac{1}{N_i} \sum_{z \in Z_i} z \quad (2.6.)$$

Ja

$$S_T = \sum_{z \in Z} \|z - m\|^2 \quad (2.7.)$$

un

$$S_W = \sum_{i=1}^C S_i = \sum_{i=1}^C \sum_{z \in Z_i} \|z - m_i\|^2 \quad (2.8.)$$

tad J definē kā

$$J = \frac{S_B}{S_W} = \frac{(S_T - S_W)}{S_W} \quad (2.9.)$$

Šis mērs nosaka, cik labi atšķirami ir dažādi krāsu klašu reģioni attēlā - jo lielāka J vērtība, jo tā krāsu klasses ir labāk atšķiramas, homogēnas un krāsu klasei piederīgie pikseļi tuvāki viens otram vērtību ziņā.[7]

Lai noteiktu, kāds ir labs segmentācijas rezultāts, ieviešam mainīgo \bar{J} , kas tiek aprēķināts kā vidējā vērtība visiem segmentētajiem reģioniem tā vietā, lai to aprēķinātu uzreiz visam attēlam. Šāda mainīgā vērtību aprēķina

$$\bar{J} = \frac{1}{N} \sum_k M_k J_k, \quad (2.10.)$$

kur J_k ir reģiona k J vērtība, M_k punktu skaits reģionā k un N kopējais punktu skaits klasē kartē, bet summēšana notiek pa visiem reģioniem k . Šādi ieviests mainīgais ievieš jaunu kritēriju segmentācijas kvalitātes noteikšanai, kur labāko segmentācijas kvalitāti izsaka mazākā iespējamā \bar{J} vērtība pār visiem reģioniem k . [7]

J-attēla izveide Tomēr iepriekš aprakstītā kritērija pielietošana nav praktiska. Tādēļ tiek ieviests tāds jēdziens, kā J-attēls. J-attēls ir melnbalts attēls, kura pikselu vērtības veido J vērtība, kas aprēķināta interesējošā pikseļa apkārtnē ar izvēlētu rādiusu. Šādam attēlam raksturīga kalniem un ieļām līdzīga struktūra, kur tumšākās vērtības norāda tos pikselus, kas vairāk koncentrēti kāda krāsu reģiona centrā, bet gaišākās vērtības pretēji attiecināmas uz krāsu reģiona robežu.[8]

Interesējošā pikseļa apkārtnes izmērs tiek izvēlēts balstoties uz nepieciešamo atšķiramo reģionu izmēriem, kā arī reģionu robežas detaļu smalkuma. Attiecīgi izmantojot mazāku loga izmēru iespējams izšķirt smalkākas detaļas un reģionus, kurus sastāda atsevišķas detaļas, bet liels loga izmērs ir izdevīgs tajos gadījumos, kad nepieciešams izšķirt reģionus, ko raksturo specifiska tekstrūra. [7]

Lai noteiktu kāda pikseļa piederību konkrētajam reģionam, bet, lai nerastos novirzes taisnstūra formai līdzīgu reģionu dēļ, loga forma ir izvēlēta līdzīga aplim vai kvadrātam ar noapaļotiem stūriem. Apļi ir pieejami dažādos mērogos, kur katrs nākamais ir ar divas reizes lielāku rādiusu kā iepriekšējais. [7]

Reģionu audzēšana J-attēla raksturīgās īpašības ļauj izmantot reģionu audzēšanas algoritmus attēla segmentācijā. Ja pieņem, ka sākumā viss attēls uzskatāms par veselu reģionu, tad kā redzams tabulā 2.1., algoritms jāsāk tādā mērogā, kāds ir nosakāms pēc reģiona izmēra. Piemēram, gadījumā, ja attēls nepārsniedz 512 pikselus platumā un/vai garumā, bet ir lielāks par 256 pikseliem, algoritms jāsāk no 3. mēroga soļa. [8]

Tabula 2.1. Logu izmēri pie dažādiem mērogiem.

Mērogs	Loga izmērs (pikseļos)	Reģiona izmērs (pikseļos)	Min. ieļejas izmērs (pikseļos)
1	9 x 9	64 x 64	32
2	17 x 17	128 x 128	128
3	33 x 33	256 x 256	512
4	65 x 65	512 x 512	2048

Pirmais solis ir noteikt, kuri pikseļi veidos tā saucamo ieļejas sēklas pikseļu kopu. Viens variants ir aprēķināt katras reģiona vidējo vērtību un standartnovirzi, apzīmējot tās ar μ_J un σ_J un ieviest sliekšņa vērtību T_J .

$$T_j = \mu_J + a\sigma_J \quad (2.11.)$$

Šajā gadījumā a ir kāda vērtība no skaitļu masīva $[-0.6, -0.4, -0.2, 0, 0.2, 0.4]$. Tas, kura vērtība no masīva tiek izmantota, tiek noteikts eksperimentālā veidā, balstoties uz to, kura vērtība garantē vislielāko ieļejas punktu skaitu. Iepriekš ieviestā sliekšņa vērtība tiek pielāgota visām lokālajām J vērtībām un tās, kuras ir mazākas par šo vērtību, tiek sauktas par ieļejas punktu kandidātiem. Ieļejas kandidātu punktus savieno izmantojot 4-savienojamību

un tā tiek iegūti ieleju kandidāti. Ja šādas ielejas izmērs ir lielāks nekā norādīts 2.1. tabulā, tad tādu kandidātu nosaka par ieleju. [7]

Tā kā audzēt reģionu pikseli pa pikselim ir ļoti lēni, tiek izmantota citādāka pīeja. Vispirms ielejās aizpilda caurumus un atrod tās J vērtības, kuras ir zemākas par visu nesegmentēto pikseļu vidējo J vērtību. J vērtības, kas ir zemākas par vidējām vērtībām, savā starpā savieno un, ja tās atrodas blakus vienai vienīgai ielejai, tad pievieno tai. Tālāk pārvietojas uz nākamo mazāko mērogu un atkal atkārto iepriekš aprakstītās darbības ar to J vērtību atrašanu, kuru vērtība mazāka nu jau par nākamā mēroga vidējo J vērtību un to pievienošanu ielejai. Šādi turpina līdz sasniegts mazākais mērogs, kad pikseļu pievienošana jau notiek pa vienam. Pievienošanu sāk ar to, kam mazākā J vērtība, un tajā brīdī nepievienotie pikseļi tiek uzglabāti buferatmiņā (LZIA, angl. *buffer*) katru reizi informāciju par nepievienotajiem pikseliem atjaunojot. [7]

Reģionu apvienošana Standarta JSEG algoritmā reģionu apvienošana tiek veikta kā pēdējais solis. Reģionus apvieno izmantojot to, ka kvantizēto krāsu attēlā katram reģionam iespējams izveidot histogrammu, uz kuru balstoties tiek noteikta reģionu līdzība. Vispirms tabulā tiek aprēķinātas katru 2 blakus esošo kaimiņreģionu krāsu distances un tad šie reģioni, ar mazāko krāsas distanci starp pārējām, tiek apvienoti. Apvienotajam reģionam atkal tiek aprēķināta krāsas distance ar kaimiņreģioniem un informācija par distancēm tabulā tiek atjaunota. Process tiek atkārtots līdz tiek sasniegta iepriekš noteikta maksimāla sliekšņa vērtība. Iegūtais rezultāts arī ir attēla segmentācijas galarezultāts.[7]

Literatūras avotā [22], kā viens no JSEG metodes trūkumiem, tiek minēts tā segmentācijas rezultātu izteiktā sadrumstalotība salīdzinoši ar automātisko reģionu audzēšanas algoritmu no sēklas pikseliem. To iespējams labot pēc segmentēšanas iegūtos reģionus klasificējot, izmantojot labu klasifikācijas algoritmu. Kā minēts pētījumos [7], [8], [23] un [26] JSEG metodes priekšrocības, salīdzinoši ar citām attēlu segmentācijas metodēm, ir:

- Nav nepieciešams tekstūras modeļa parametru novērtējums;
- Algoritms ir piemērots automātiskai attēlu apstrādei;
- Algoritms necieš no trokšņa izraisītiem traucējumiem.

JSEG metodes realizācija krāsu attēlu segmentācijā

Tālāk apskatīsim JSEG implementāciju aerofotogrāfijas segmentācijai. Autore eksistējošu JSEG implementāciju programmatūrā *MatLab* optimizēja un pielāgoja pētījuma vajadzībām. Algoritmu iespējams izpildīt jebkurai aerofotogrāfijai, kas saglabāta TIFF formātā jeb tas ir 3 dimensiju attēls , kura vērtības ir 8 bitu veseli skaitļi bez zīmes (LZA, angl. *unsigned integer*) (*uint8*).

Kā pats pirmsais solis metodes implementācijā ir aerofotogrāfijas ielasīšana, kas tiks saglabāta 3 dimensiju matricā f . Tālāk šī matrica f tiek padota kā viens no parametriem metodei *mainJSEG*. Metode *mainJSEG* kā ievadparametru saņem veselu skaitli (*parts*), 3

dimensiju matricu f , ko jau iepriekš apskatījām un veselu skaitli $kmeansK$. Parametrs $parts$ raksturo to, cik daļas tiks sadalīts attēls gan horizontāli, gan vertikāli, bet $kmeansK$ nosaka kvantizācijai izmantotā pudurošanas pēc vidējiem (LZA, angl. *k-means clustering*) k parametru. Šī metode izvada reģionu matricu $regions$ un kvantizēto attēlu $ImgQ$. Tālāk detalizētāk apskatīsim pašu *mainJSEG* funkciju.

Vispirms izveidojam tukšu divdimensiju *region* matricu tādā pašā izmērā, kā aerofotogrāfija, aprēķinam aerofotogrāfijas daļas izmēru, cik lielai daļai tiks veikti aprēķini vienā cikla reizē un izveidojam direktoriju starprezultātu uzglabāšanai. JSEG segmentācija tiek veikta ejot cauri kā horizontāli, tā vertikāli visiem aerofotogrāfijas apakšapgabaliem un vispirms aprēķinot katru apakšapgabala koordinātas, jo pēdējiem apakšapgabaliem kā rindā, tā kolonnā, var nebūt standarta izmērs (vienāds ar pārējiem ar apakšapgabaliem). Tālākajā koda fragmentā redzams, kā tiek izvēlēts apakšapgabals $[i, j]$ un izmantojot pudurošanu pēc vidējiem, tam tiek veikta kvantizācija, kuras rezultāts tiek atgriezts kā karte (matrica *map*) ar tādu pašu izmēru, kā apakšapgabalam.

```

1 subIm=image(((i-1)*partSize)+1:endOfImV,((j-1)*partSize)+1:endOfImH,:);
2 %—————K-means—————%
3 [m,n,d] = size(subIm);
4 subIm=im2double(subIm);
5 X = reshape(subIm, m*n,d);
6 clusters = kmeans(X,kmeansK);
7 map = reshape(clusters, m, n);

```



(a) Klašu karte.

(b) Kvantizēts ortofotokartes fragments.

Att. 2.4. Pudurošanas pēc vidējiem gala rezultāts ortofotokartes fragmentam, ja $k = 8$.

Pudurošanas pēc vidējiem parametrs k ir viens no nozīmīgākajiem parametriem JSEG segmentācijā, kas nosaka to, cik daudz daļas tiks segmentēts attēls. 2.4. attēlā redzama klašu karte, ja pudurošanas pēc vidējiem parametrs $k = 8$ un klašu kartei atbilstošais kvantizētais attēls. Toties 2.5. attēlā redzama klašu karte un tai atbilstošais kvantizētais ortofotokartes fragments, ja pudurošanas pēc vidējiem parametrs $k = 16$. Salīdzinot 2.4. un 2.5. attēlus,



(a) Klašu karte.

(b) Kvantizēts ortofotokartes fragments.

Att. 2.5. Pudurošanas pēc vidējiem gala rezultāts ortofotokartes fragmentam, ja $k = 16$.

redzams, ka visizteiktāk krāsu samazināšana ietekmējusi ūdens platības, zālāju un ceļu, kam raksturīga viendabīga tekstūra, bet vismazāk ietekmējusi koku vainagu pārsegumu. Protams, pudurošanas pēc vidējiem parametrs k ir jāpielāgo arī apskatāmā attēla izmēram un dabas daudzveidībai tajā. Maza parametra vērtība var sniegt labus rezultātus mazā attēlā, bet lielākos attēlos vai tādos, kuros sastopama plaša dabas daudzveidība, var tikt nevajadzīgi sapludināti dažādi zemes pārseguma tipi.

Tālāk izmantojot jau klašu karti, tiek veikta J-attēla izveide. J-attēla izveide tiek veikta četros mērogos, kas gala rezultātā veido četrus J-attēlus - $J11, J12, J13, J14$, kur $J11$ viissmalkākais, bet $J14$ attiecīgi izmantojot vislielāko mērogu. Tā kā visos mērogos J-attēla izveide ir vienāda, apskatīsim sīkāk tikai J-attēla izveidi smalkākajā mērogā ar loga izmēru 9. Šajā gadījumā funkcijai $JImage$ tiek padota klašu karte map (funkcijā parametrs I), loga izmērs (funkcijā parametrs ws), kas, kā jau minēts, vienāds ar 9 un minimālā loga izmērs (funkcijā parametrs $minWindow$), kas šajā gadījumā sakrīt ar loga izmēru. Izvadparametrs ir J-attēla matrica $J11$.

Funkciju sākam ar klašu kartes izmēru m un n iegūšanu, kā arī aprēķinam parametru d , kas noteiks, ik pa cik pikseļiem, tiks veidots attēls J-attēla izveidošanai un loga diametru $wswidth$. Līdzīgi, kā $mainJSEG$ funkcijā, izveidojam tukšu matricu $J1$. Tālāk katram punktam $[i, j]$ no klašu kartes I tiek aprēķināta tā apkārtnes robežu punkti un izmēri. Gadījumā, ja robežu vērtības pārsniedz klašu kartes izmērus vai tās ir negatīvas, tās tiek aizstātas ar minimālajām vai maksimālajām iespējamām vērtībām.

```

1 x1 = i-wswidth;
2 x2 = i+wswidth;
3 y1 = j-wswidth;
4 y2 = j+wswidth;
5 if x1<1; x1 = 1; end
6 if x2>m; x2 = m; end
7 if y1<1; y1 = 1; end

```

```

8 if y2>n; y2 = n; end
9 wid = x2-x1+1;
10 hei = y2-y1+1;

```

Pēc tam tiek aprēķināta standartnovirze izmantojot aprēķinātos loga izmērus un atrašanās vietu un šī standartnovirze tiek izmantota, kā viens no parametriem J vērtības aprēķināšanai funkcijā $J\text{Calculation}$, kas bez standartnovirzes vēl kā ievadparametru saņem katru d -to pikseli no klašu kartes I diapozonā no $x1$ līdz $x2$ un no $y1$ līdz $y2$. Sīkāk šīs funkcijas aprakstīšanā neiedziļināsimies, tikai ir vērts piebilst, ka šī funkcija atgriež J vērtību punktam $[i, j]$, kas arī tiek saglabāta iepriekš sagatavotajā, tukšajā matricā JI . Ar to arī $J\text{Image}$ funkcijas apskatu var pabeigt.

```

1 if wid == ws && hei == ws
2     St = 1080;
3 else
4     wid=length(1:d:wid);
5     hei=length(1:d:hei);
6     reg=ones(wid,hei);
7     M = [mean(1:wid), mean(1:hei)];
8     [z1, z2] = find(reg);
9     z = [z1, z2];
10    St = sum(sqrt(sum((z - M).^2)));
11 end
12 JI(i,j) = J\text{Calculation}(I(x1:d:x2, y1:d:y2), St);

```

Atgriežoties pie mainJSEG apskata, nākamais solis ir cilvēka acij patīkama kvantizētā attēla sagatavošana apakšapgabalam $subIm$. Lai to izdarītu, funkcijai $class2Img$ tiek padota attiecīgā apakšapgabala klašu karte map (funkcijā $class_map$) un paša apakšapgabala matrica $subIm$ (funkcijā OrI). Kā izvadparametrs, tiek veidota matrica $partImgQ$ (funkcijā Img). Kā jau pārējās funkcijās, arī šajā vispirms tiek noteikti apakšapgabala izmēri (m, n, d) un izveidota tāda paša izmēra matrica $partImgQ$ (Img) uzglabāšanai. Tālākajā koda fragmentā katrai klasei no klašu kartes tiek atbilstoši pa attēla slāņiem aprēķināta vidējā vērtība un visu klasei piederošo pikselu vērtības tiek ar to aizstātas.

```

1 for i = 1:max(class_map(:))
2     sq = find(class_map == i);
3     for j = 1:d
4         Channel = OrI(:,:,j);
5         u = mean(Channel(sq));
6         Channel = zeros(m,n);
7         Channel(sq) = u;
8         Img(:,:,:,j) = Img(:,:,:,:,j) + Channel;
9     end
10 end

```

Pēc cilvēka acij tīkama, kvantizēta attēla iegūšanas nepieciešams veikt ieļeju identifikāciju. Ieļeju identifikāciju veic funkcija *ValleyD*. Funkcijai, kā parametri, tiek padota J-attēla matrica *J14* (funkcijā parametrs *J1*) vislielākajā mērogā, pats mērogs (funkcijā parametrs *scale*), kas šajā gadījumā ir 4, J-attēla vidējā vērtība *uj* un standartnovirze *sj*. Vispirms tiek atrasti matricas izmēri *m*, *n* un *d* un izveidota tukša matrica *ValleyI* rezultāta uzglabāšanai. Tad tiek ieviests specifisks masīvs *a*, kas satur $[-0.6, -0.4, -0.2, 0, 0.2, 0.4]$, noteikts minimālais ieļejas izmērs *scale*, balstoties uz mērogu, un pagaidām atrastais maksimālais ieļejas izmērs uzstādīts vienāds ar nulli. Zemāk esošajā koda fragmentā, izmantojot katru masīva *a* elementu, tiek aprēķina sliekšņa vērtība *TJ*, kuru izmantojot tiek atrasti kandidātu ieļeju punkti *VP*. Izmantojot 4-savienojamību un minimālā ieļejas izmēra vērtību *scale*, tiek atrastas ieļejas. Ja šo atrasto ieļeju izmērs pie konkrētās *a* vērtības lielāks, kā atrastais maksimālais ieļejas izmērs, tad tās tiek saglabātas izvadparametrā *ValleyI* un tagadējais maksimālās ieļejas izmērs *MaxVSize* tiek atjaunots.

```

1  for i = 1:length(a)
2      TJ = uj + a(i)*sj;
3      VP = false(m,n);
4      VP(JI <= TJ) = 1;
5      VP_lab = bwlabel(VP,4);
6      VPlab_hist = histc(VP_lab(:),1:max(VP_lab(:)));
7      sq = find(VPlab_hist >= scale);
8      VSize = length(sq);
9      if length(sq)>MaxVSize
10         ValleyI = zeros(m, n);
11         for k = 1:length(sq)
12             ValleyI(VP_lab==sq(k))=k;
13         end
14         MaxVSize = VSize;
15     end
16 end

```



(a) RGB attēls. (b) Klašu karte ($k = 12$). (c) Identificētās ieļejas.

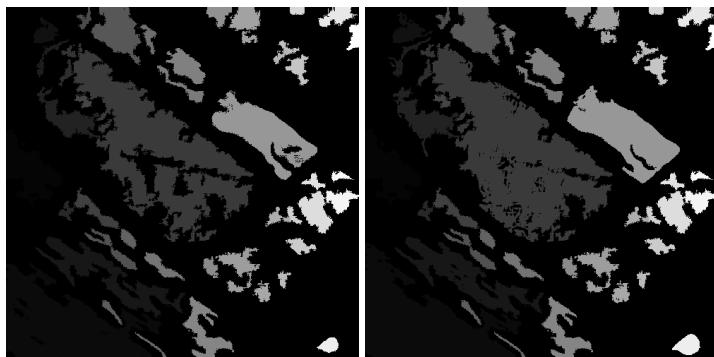
Att. 2.6. Ortofotokartes fragments ar atbilstoši identificētajām ieļejām.

Kad ir atrastas ielejas (redzamas 2.6.c attēlā), iespējams veikt audzēšanu lielākajā mērogā. Ieleju audzēšana pēc funkcijas *ValleyG1* noris vēl arī 2. un 3. mērogā. Ievadparametri *ValleyG1* funkcijai ir J-attēls *J1* konkrētajā mērogā (šajā gadījumā apskatīsim *J14*) un iepriekšējā solī sagatavotā reģionu matrica apakšapgabalam *subRegion* (funkcijā parametrs *ValleyI*). Izvadparametrs ir arī reģionu matrica (funkcijā parametrs *SegI*) tam pašam apakšapgabalam. Šīs funkcijas darbība, galvenokārt, paredzēta ieleju audzēšanai. To iespējams panākt vispirms aizpildot caurumus, kas radušies iepriekšējā fāzē. Tad atrod tos punktus no segmentētajiem, kuru vērtība mazāka par nesegmentēto punktu vidējo vērtību, bet visam attēlam no iepriekšējās fāzes *ValleyI* veic atvēšanu (angl. *dilation*). Pēc tam iepriekš atrastajiem punktiem, kuru vērtība mazāka par vidējo, līdzīgi, kā *ValleyD* funkcijā, veic operāciju, izmantojot 4-savienojamību. Balstoties uz iegūtajiem rezultātiem pēc 4-savienojamības pielietošanas, veic ieleju audzēšanu un rezultātu pievieno *SegI* matricai. Gala rezultātā atkal tiek iegūta reģionu matrica (redzama 2.7.a).

```

1 subRegion = ValleyD( JI4 ,   4, mean( JI4(:)), std( JI4(:)));
2 subRegion = ValleyG1( JI4 , subRegion );

```



(a) Reģioni pēc reģionu audzēšanas 4. mērogā izmantojot dzēšanas 4. mērogā izmantojot *J14* matricu. (b) Reģioni pēc reģionu audzēšanas 4. mērogā izmantojot dzēšanas 4. mērogā izmantojot *J13* matricu.

Att. 2.7. Reģionu attēli pēc audzēšanas, izmantojot funkciju *ValleyG1* 4. mērogā.

Pēc tam, kad iegūta atjaunota reģionu matrica pēc reģionu audzēšanas lielākajā mērogā (izaudzētie reģioni redzami 2.7.b attēlā), tai tiek veikta reģionu audzēšana izmantojot nākamo lielāko mērogu jeb mērogu 3 ar to pašu funkciju *ValleyG1*. Kā pēdējais solis pie mēroga 4, tiek veikta ieleju audzēšana izmantojot funkciju *ValleyG2*, kur atlikušie nesegmentētie punkti tiek tik daudzas reizes pārdalīti, izmantojot paša mazākā mēroga J-attēlu, līdz visi ir pieaudzēti pie kādas no ielejām.

```

1 subRegion = ValleyG1( JI3 , subRegion );
2 subRegion = ValleyG2( JI1 , subRegion );

```

Tālāk līdzīgas darbības tiek veiktas arī nākamajos mērogos līdz galarezultātā tiek iegūta segmentēto reģionu matrica *subRegion* apakšapgabalam.

```

1 % scale 3 %
2 subRegion = SpatialSeg( JI3 , subRegion , 3 );
3 subRegion = ValleyG1( JI2 , subRegion );
4 subRegion = ValleyG2( JI1 , subRegion );
5 % scale 2 %
6 subRegion = SpatialSeg( JI2 , subRegion , 2 );
7 subRegion = ValleyG1( JI1 , subRegion );
8 subRegion = ValleyG2( JI1 , subRegion );
9 % scale 1 %
10 subRegion = SpatialSeg( JI1 , subRegion , 1 );
11 subRegion = ValleyG2( JI1 , subRegion );

```

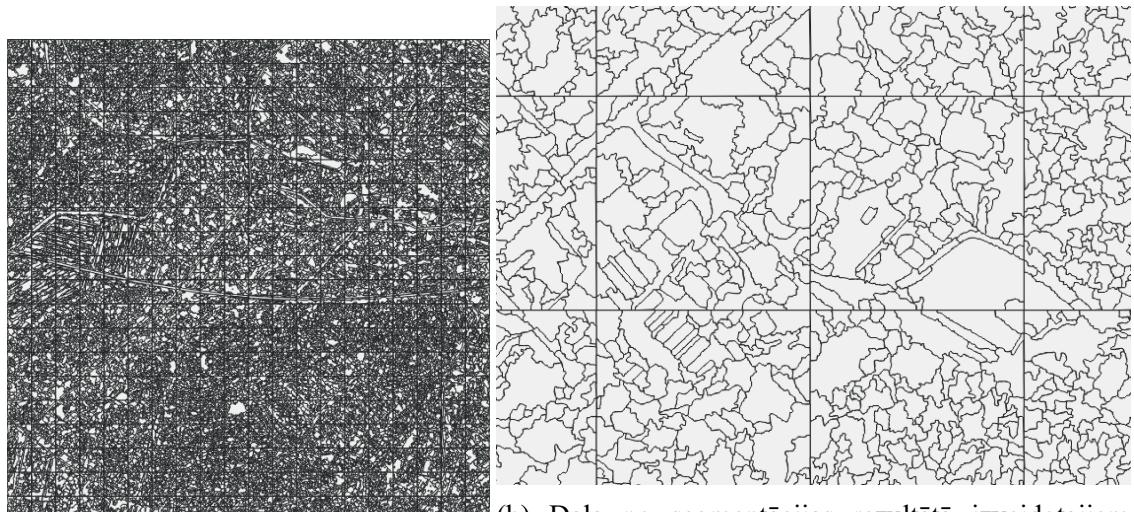
Tā kā katru reizi *subRegion* matrica savus reģionus sāk skaitīt sākot ar 1, apakšapgabala reģionu matricas katrs reģiona numurs tiek pārvērts tam pieskaitot šobrīd maksimālo reģiona numuru lielajā matricā *region*, kurā pēc tam tāpat kā *ImgQ* tiek ievietots starprezultāts apakšapgabalam.

```

1 ImgQ(((i-1)*partSize)+1:endOfImV,((j-1)*partSize)+1:endOfImH, ...
      :)=partImgQ;
2 region(((i-1)*partSize)+1:endOfImV,((j-1)*partSize)+1:endOfImH) = ...
      subRegion;

```

Funkcijas *mainJSEG* nobeigumā vēl tiek saglabāts *subRegion* starprezultāts iepriekš izveidotajā direktorijā un cikls sāk apstrādāt nākamo apakšapgabalu. Gala rezultātā tiek atgriezta reģionu matrica *region* (*Region*), kas redzama attēlā 2.8.a un kvantizētais attēls *ImgQ* vēlākai rezultātu attēlošanai.



(a) Attēls ar segmentācijas rezultātā izveidoti reģioniem.
 (b) Daļa no segmentācijas rezultātā izveidotajiem

Att. 2.8. Segmentācijas rezultāts

2.1. 3. K-tuvāko kaimiņu metode un tās realizācija

K-tuvāko kaimiņu metodes teorētisks apskats

K-tuvāko kaimiņu metode ir klasifikācijas metode, kas pieder pie vadītās mašīnmācīšanās metožu grupas. Klasifikācija ir sistemātiska vienumu sadalīšana grupās, kategorijās. Metodes pamatideja - pikselā piederība kādai klasei tiek izsvērta no tā, kādai klasei pieder tā tuvākais kaimiņš. Šāds skaidrojums izmantojams gadījumā, kad $k = 1$. Gadījumā, kad $k \geq 2$, piederība konkrētai klasei tiek izsvērta no lielākās daļas tuvāko kaimiņu piederības. Piemēram, gadījumā, ja $k = 3$ un tiek apskatīti kāda punkta 3 tuvākie kaimiņi, kur viens no tiem pieder 1. klasei, bet divi atlikušie 2. klasei, tad apskatītais punkts pieder 2. klasei.[6]

K-tuvāko kaimiņu algoritms sastāv no diviem soliem. Pirmais solis ir tuvāko kaimiņu atrašana, bet otrs, klases piešķiršana, izmantojot šos kaimiņus.[6]

Lai atrastu tuvākos kaimiņus, vispirms vērts definēt jēdzienu distance. Tātad ar vārdu distance izsakām kādu mēru, kas raksturo viena pikselā vai reģiona deskriptoru vektora attālumu līdz otram deskriptoru telpā. Iespējami vairāki veidi kā aprēķināt distances. Bieži lietota ir Eiklīda distance, kuru aprēķina

$$d(x, y) = \sqrt{\sum_{i=1}^z (x_i - y_i)^2} \quad (2.12.)$$

Šajā gadījumā x un y ir pikselu vai reģionu deskriptoru vektori z dimensionālā deskriptoru telpā, starp kuriem tiek aprēķināta distance. Alternatīvs un skaitlošanas ziņā ātrāks veids ir Manhetena distance. Manhetena distanci aprēķina pēc

$$D = \sum_{i=1}^z |x_i - y_i| \quad (2.13.)$$

Arī šeit x un y ir divi deskriptoru vektori z dimensionālā deskriptoru telpā.[27]

Protams, veiksmīgi distanci iespējams aprēķināt tikai, ja deskriptoru vērtības ir normalizētas jeb visas vērtības atrodas vienā diapazonā [6].

Otrs solis - klases piešķiršana, izmantojot atrastos kaimiņus ir viegli un intuitīvi saprotama gadījumā, ja $k = 1$ vai $k > 2$ un tuvāko kaimiņu klases nav vienādi sadalītas. Tādā gadījumā apskatītajam punktam tiek piešķirta tā klase, kāda ir visbiežāk sastopama starp kaimiņiem. Problemātiska situācija var rasties, ja $k = 2$ vai klašu sadalījums starp kaimiņiem ir vienāds. Tādā gadījumā klases piešķiršana apskatītajam punktam notiek ņemot vērā kādus papildus faktorus, kā piemēram, nozīmīgāko pazīmi, ja punkts atrodas vairāku dimensiju telpā (to raksturo vairākas pazīmes), vai piešķir tieši tuvākā kaimiņa klasi. [6]

K-tuvāko kaimiņu metodes pielietojums segmentētu reģionu klasifikācijā

Kad visi nepieciešamie dati sagatavoti, varam sākt apskatīt funkciju *classifierKNN*. Funkcija *classifierKNN*, kā parametrus saņem reģionu matricu *Region*, 3-dimensiju matricu

attēlam *image_org*, k-tuvāko kaimiņu metodes parametru *k*, kas nosaka izmantoto tuvāko kaimiņu skaitu, deskriptoru matricu parauga datiem *sampleDescr*, deskriptoru matricu visam attēlam *vissDescr* un klašu vektoru parauga datiem *sampleClasses*. Funkcijas izvadītais rezultāts ir segmentējamā attēla izmēriem atbilstoša maska, kur katrā pikselā vērtība aizstāta ar klasi, kam tas pieder.

Funkcija *classifierKNN* izsauc funkciju *mansKNN*, kas tika izstrādāta studiju kursa ”*Digitālo attēlu apstrāde*” ietvaros un rezultātā tiek iegūts vektors, kur katram segmentētajam reģionam atbilst tā klase.

Pēc tam šis vektors tiek pārvērts klašu maskā, kas tiek atgriezta no *classifierKNN* funkcijas.

```

1 function mask = classifierKNN(region , rgb_image , k , sampleDescr , ...
2   vissDescr , sample_classes)
3   classified_image=mansKNN(vissDescr , k , sampleDescr , ...
4     sample_classes);
5   sample_classes=unique(sample_classes);
6   [ r k d]=size(rgb_image);
7   mask=zeros(r,k);
8   for i=1:length(classified_image)
9     for j=1:length(sample_classes)
10       if classified_image(i)==sample_classes(j)
11         mask(region==i)=j;
12       end
13     end
14   end
15 
```

Detalizētāk apskatīsim funkciju *mansKNN*, kas klasificē segmentētos reģionus. Funkcija *mansKNN*, kā parametrus saņem klasificējamo attēlu *im*, kas šajā gadījumā būs deskriptoru matrica visam attēlam ar deskriptoriem katram attēla segmentētajam reģionam, k-tuvāko kaimiņu metodes parametru *k*, kas nosaka izmantoto tuvāko kaimiņu skaitu, parauga datus *sample*, kas šajā gadījumā saturēs matricu *sampleDescr* ar deskriptoru vērtībām parauga datu reģioniem un klašu vektoru parauga datiem *sample_classes*. Rezultāti tiks uzglabāti matricā *classIm*, kas saturēs tikpat rindiņu, cik deskriptoru matrica visiem reģioniem *vissDescr*.

Pati funkcija tikai izveido matricu rezultāta uzglabāšanai un ciklā katram reģionam nosaka tā piederību konkrētai klasei izmantojot funkciju *knnProcedura*.

```

1 function classIm=mansKNN(im , k , sample , sample_classes)
2   [ r , ko , d]=size(im);
3   classIm=zeros(r,1);
4   for i=1:r
5     x=im(i,:);
6     classIm(i)=knnProcedura(x , sample , sample_classes , k);
7   end
8 
```

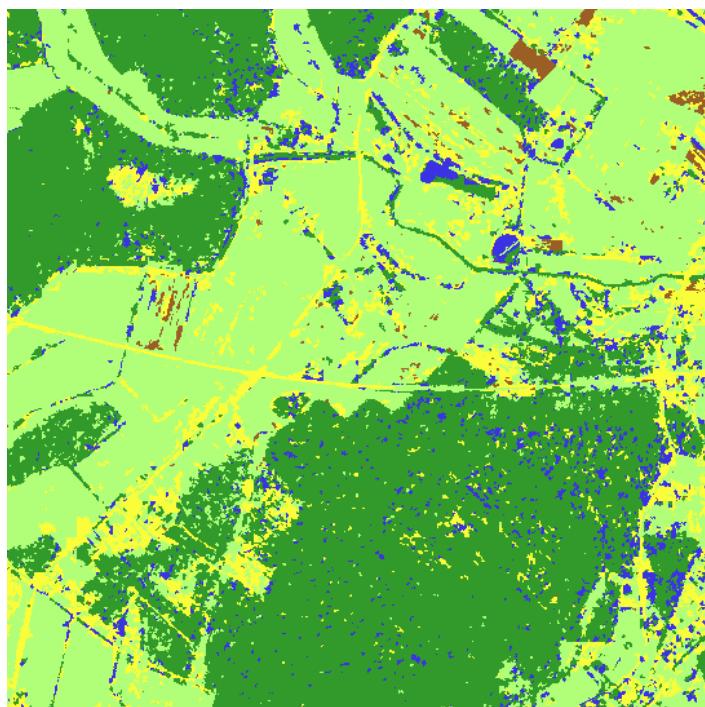
Funkcija *knnProcedura*, kā parametrus saņem reģiona deskriptoru vektoru *x*, deskriptoru matricu parauga datiem *sample*, parauga datu kļaušu vektoru *sample_classes* un k-tuvāko kaimiņu metodes parametru *k*. Vispirms tiek noteikt parauga datu skaits un izveidots vektors *d*, kas sevī saturēs distances no reģiona, kam tiek meklēta klasses piederība līdz visiem pārējiem parauga datu reģioniem.

```

1 function knnResponse=knnProcedura (x , sample , sample_classes , k)
2     sampleN=length ( sample_classes );
3     d=zeros ( sampleN , 1 );
4     for z=1:sampleN
5         d ( z ) =sqrt ( sum ( ( x -sample ( z , : ) ) . ^2 ) );
6     end

```

Kad atrastas visas distances nepieciešams noteikt *k* īsākās. To realizē ciklā aizpildot *k* garu rindas vektoru ar tām distancēm atbilstošo reģionu klasēm, kas ir vismazākās un šo distanču vērtības aizvietojot ar bezgalību, lai visās *k* reizēs netiktu atrasta viena vērtība. Pēc tam no šīm *k* klasēm tiek noteikta vektora moda jeb vērtība, kas atkārtojas visbiežāk, kas arī tiek atgriezta, kā rezultāts funkcijai *knnProcedura*.



Att. 2.9. Klašu maska klasificētajiem JSEG segmentācijas rezultātiem, izmantojot k-tuvāko kaimiņu metodi.

```

1 typesForKNb=zeros ( 1 , k );
2 for z=1:k
3     [m, I]=min ( d );
4     typesForKNb ( z ) =sample_classes ( I );
5     d ( I ) =Inf ;

```

```

6   end
7   knnResponse=mode( typesForKNb );

```

2.9. attēlā redzama klašu maska pēc reģionu klasifikācijas, izmantojot k-tuvāko kaimiņu metodi. Zilā krāsā attēloas ūdens platības, tumši zaļā - koku pārsegums, gaiši zaļā - zālāji, brūnā - lauki un dzeltenā krāsā platības, kas atbilst citam pārseguma tipam.

2.2. Neironu tīklu pielietojums

2.2. 1. Apmācības datu sagatavošana

Apmācības datu sagatavošanai tiek izmantoti tie paši attēli mazākā apjomā kā k-tuvāko kaimiņu metodes gadījumā. Ja k-tuvāko kaimiņu metodes gadījumā, lai aprēķinātu deskriptorus, bija iespējams izmantot reģionu attēlus katram pārseguma tipam tos iepriekš nekā neapstrādājot, tad neironu tīkla pielietošanas gadījumā situācija ir citādāka. Šajā gadījumā reģionu attēlus nepieciešams sadalīt smalkāk jeb tā, ka katrs regions ir atsevišķs attēls.

Lai sasniegtu šādu rezultātu un izvairītos no nevēlamu artifaktu parādīšanās attēlos, ar programmatūras *MatLab* palīdzību tika izveidota funkcija *saveTestDataByRegion*. Šī funkcija, kā ievadparametru saņem matricu *testRegion* ar tā pārseguma tipa parauga datu reģioniem, ko nepieciešams sadalīt attēlos, oriģinālā attēla matricu *I* ciparu formātā *double* formā $M \times N \times 3$, kur *M* apzīmē rindiņu skaitu, bet *N* - kolonnu skaitu un pārseguma tipa kārtas numuru *folder* direktorijas izveidošanai, kur tiks saglabāti izveidotie attēli. Vispirms tiek atrasti to reģionu numuri, kurus satur matrica *testRegion* un izveidota direktorija izveidoto attēlu saglabāšanai.

```

1 function saveTestDataByRegion( testRegion , I , folder )
2     uniqueRegions=unique( testRegion );
3     mkdir( in2str( folder ) );

```

Tālāk ciklā, ejot cauri unikālajiem reģionu numuriem sākot no otrā, jo pirms vienmēr apzīmēs visus tos reģionus, kas nepieder izvēlētajiem parauga datiem, izveido matricu *region*, kas saturēs tajā brīdī apskatāmo reģionu. Izmantojot *MatLab* iebūvēto funkciju *regionprops*, tiek atrasti mazākie taisnstūri, kas iekļauj apskatītajā reģionā savienotus objektus un informācija par šiem taisnstūriem tiek saglabāta mainīgajā *stats*. Pēc tam, ja mainīgais *stats* sevī ietver vairāk kā vienu savienoto komponenti ar to aptverošo taisnstūri, tiek izveidots rindas vektors *sizes*. Izmantojot ciklu, šis rindas vektors tiek aizpildīts ar komponenti aptverošā taisnstūra rindu un kolonnu skaits summu. Balstoties uz šo aptverošo taisnstūru izmēriem, tiek paturēta lielākā komponente, kuru aptverošā taisnstūra augšējā kreisā stūra koordinātas un rindas un kolonnu skaits tiek padots kā vektors tālākai apstrādei. Ja eksistē tikai viena savienotā komponente, tās aptverošā taisnstūra parametri tiek padoti tālāk.

```

1 for i=2:length( uniqueRegions )
2     region=(testRegion==uniqueRegions( i )) ;
3     stats=regionprops( region , 'BoundingBox' );

```

```

4      if ( length( stats )>1)
5          sizes=zeros( 1, length( stats ) );
6          for j=1:length( stats )
7              bb=ceil( stats( j ).BoundingBox );
8              sizes( j )=bb( 3 )+bb( 4 );
9          end
10         [M idx]=max( sizes );
11         bb=ceil( stats( idx ).BoundingBox );
12     else
13         bb=ceil( stats( 1 ).BoundingBox );
14     end

```

Lai izvairītos no smalku un līdz ar to nederīgu reģionu saglabāšanas, pārbauda, vai aptverošā taisnstūra laukums ir lielāks par 80 pikseļiem. Ja aptverošā taisnstūra laukums nesasniedz 80 pikseļus, cikls šajā momentā tiek pārtraukts un attiecīgais reģions kā attēls saglabāts netiek. Šādos smalkos reģionos parādās nekorektas un realitātei neatbilstošas pikseļu un reģionu raksturojošo parametru (vidējās vērtības, mediānas, standartnovirzes) vērtības, kas var negatīvi ietekmēt apmācības gaitu.

```

1      if ( bb( 3 ) *bb( 4 )<80)
2          continue ;
3      end

```

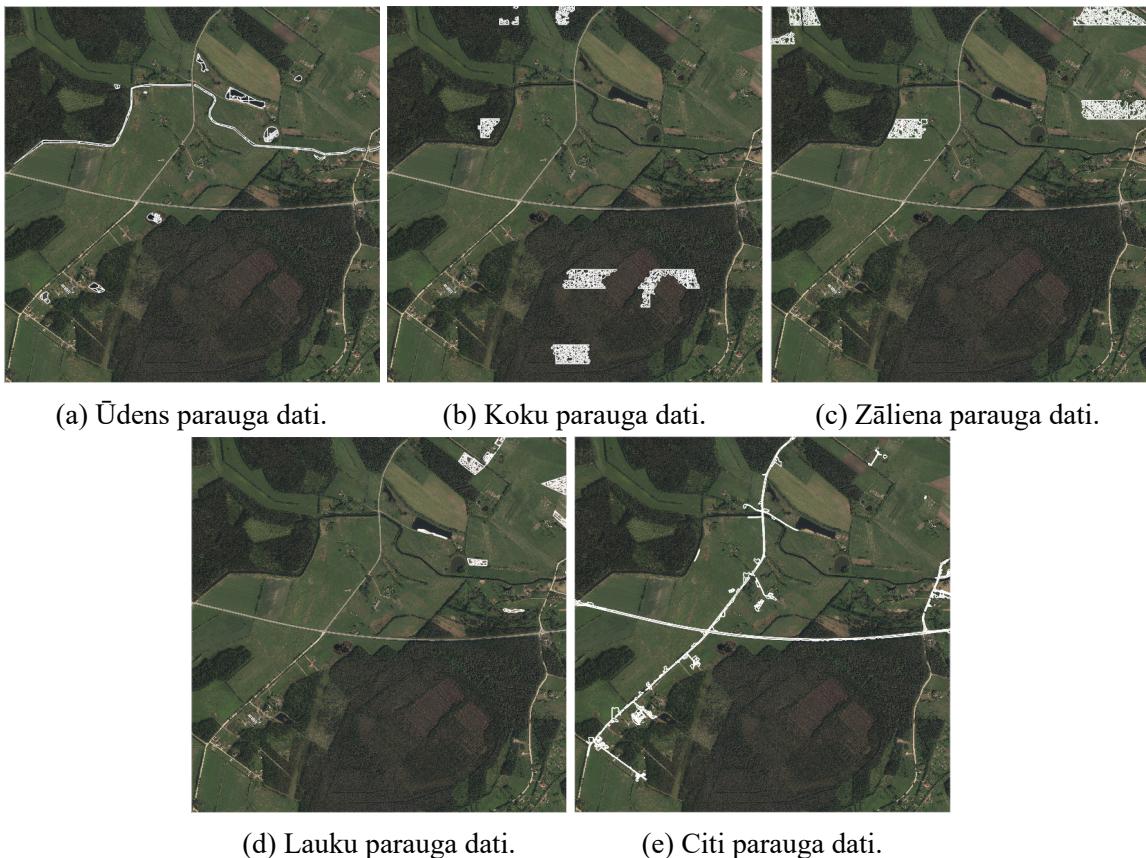
Kad iegūtas derīga reģiona kreisā augšējā stūra koordinātas un izmēri, tos izmantojot no reģionu un originālā attēla, tiek atlasītas matricas, kuru reizinājums tiek saglabāts, kā attēls *JPEG* formātā iepriekš noteiktā direktorijā.

```

1      croppedRegion=region( bb( 2 ) :bb( 2 )+bb( 4 )-1,bb( 1 ) :bb( 1 )+bb( 3 )-1 );
2      croppedIm=I( bb( 2 ) :bb( 2 )+bb( 4 )-1,bb( 1 ) :bb( 1 )+bb( 3 )-1,:) ;
3      result=croppedRegion.*croppedIm ;
4      imwrite( result , [ int2str( folder ) ' / ' int2str( uniqueRegions( i ) ) ...
5          '.jpg ' ] , ' jpg ' );
end

```

Izpildot šo funkciju katra zemes pārseguma tipa parauga datu reģionu failam, gala rezultātā tiek iegūtas 5 direktorijas ar attiecīgajiem parauga datiem tajās. Lai vienlīdz labi apmācītu neironu tīklu atpazīt visas zemes pārseguma tipu klases, "zāliena" un "koku" klasei apmācības datu skaits tiek samazināts līdz 1000 attēliem. Bez "zāliena" (attēls 2.10.c) un "koku" (attēls 2.10.b) parauga datiem neironu tīkla apmācībai tiek izmantoti 284 attēli ar "ūdens" (attēls 2.10.a) parauga datu reģioniem, 336 attēli ar "lauku" (attēls 2.10.d) parauga datu reģioniem un 365 attēli ar "cita" parauga datu reģioniem.



Att. 2.10. Parauga dati neironu tīkla apmācībai.

2.2. 2. Neironu tīkla teorētisks apskats

Bioloģiskie un mākslīgie neironi un tīkli

Mākslīgie neironu tīkli (LZA, angl. *artificial neural networks*) ir pētīti jau vairākus desmitus gadu, lai spētu iegūt līdzīgus rezultātus balss un attēlu atpazīšanā kā izmantojot cilvēka manu orgānus un smadzenes. Lai arī datoru skaitļošanas jaudas šobrīd ir kļuvušas spēcīgākas kā vēl nekad, joprojām cilvēka smadzenēm vieglāk padodas tādu uzdevumu atrisināšana, kā veselo augļu atšķiršana no bojātajiem un negataviem vai arī seju atpazīšana.[19],[20],[28]

Mākslīgie neironu tīkli ir informācijas apstrādes sistēmas, kas sastāv no liela skaita skaitļošanas vienību (neironu). To izveide ir balstīta uz pētījumiem par cilvēka smadzeņu un nervu sistēmas darbību. Mākslīgo neironu tīklu mērķis ir izveidot skaitļošanas sistēmu, kas būtu līdzīga cilvēka smadzenēm - kompleksa, paralēla un nelineāra. [16]

Neironu tīklu veido liels skaits savā starpā cieši saistītu informācijas apstrādes elementu jeb neironu, kas kopā veic konkrētu uzdevumu vai atrisina konkrēto problēmu. Mākslīgo neironu tīklu var arī saukt par vispārīgu cilvēka smadzeņu modeli, lai arī tas no cilvēka smadzenēm spējis adaptēt tikai divas īpašības:

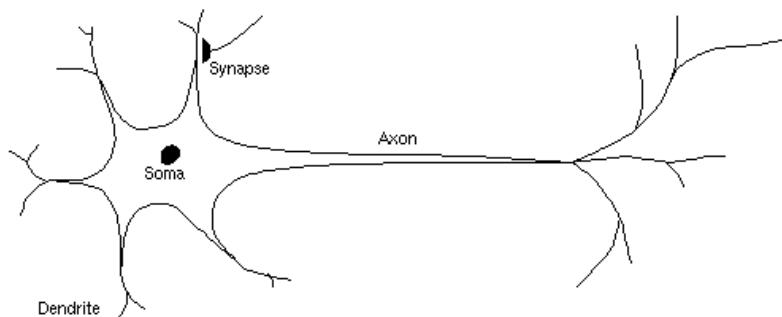
- Spēju tikt apmācītam jeb veids, kā neironu tīkls iegūst informāciju un zināšanas.
- Savienojamību jeb tas, cik ļoti stipras saites pastāv starp neironiem nosaka to, kā tīklā tiek uzglabātas zināšanas.[28]

Mākslīgā neironu tīkla iespējas slēpjās tā neironos un to saitēs. Lai arī viens pats mākslīgais nerons ir tikai salīdzinoši vienkārša skaitlošanas vienība, liels to daudzums un veidi, kā tie ir savienoti palīdz atrisināt dažādus uzdevumus. Salīdzinoši ar bioloģiskajiem neironiem, kas atrodami cilvēka smadzenēs, mākslīgo neironu takts frekvence ir daudz augstāka, bet toties to skaits ir daudz mazāks, kas rada bažas vai jebkad mākslīgie neironu tīkli spēs risināt tādus pašus uzdevumus, kā cilvēka smadzenes.[28]

Nemot vērā, ka mākslīgo neironu tīkla ideja ir aizgūta no bioloģiskajiem neironu tīkliem, vērts apskatīt bioloģiskā neirona uzbūvi. Cilvēka smadzenes sastāv no aptuveni 10 miljardiem neironu un katrs no tiem ir savienots ar 10 tūkstoš citu neironu. Neirons jeb nervu šūna sastāv no:

- ķermenē jeb somas (angl. *soma*);
- tieviem, mugurkaulam līdzīgiem ar izteiktiem sazarojumiem izaugumiem - dendrītiem (angl. *dendrites*);
- resnāka izauguma, kas savienojas ar citiem neironiem - aksona (angl. *axon*).

Neirons informāciju iegūst, izmantojot sinapses (angl. *synapses*) pirms tā tiek novadīta aksonam. [9]



Att. 2.11. Bioloģiskā neirona shematisks zīmējums [9].

Mākslīgā un bioloģiskā neirona līdzība saskatāma jau tajā, ka mākslīgais nerons no bioloģiskā ir aizguvis vairākas īpašības, kā piemēram:

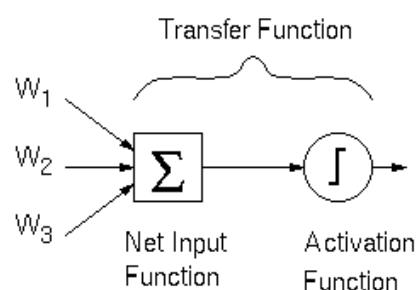
- neironā norisinās ienākošo vērtību svērtā summēšana;
- neironā ienāk vairāki signāli;
- uztverošās sinapses svars var veikt neironā ienākošās vērtības izmaiņas;
- aksonam raksturīga sazarošanās, tātad neirona izejas vērtība var nonākt vairākos citos neironos;
- gadījumā, kad ievads ir pietiekams, neirons izejā var dot tikai vienu signālu.

Papildus šīm īpašībām mākslīgajiem neironiem ir arī īpašības, kas aizgūtas balstoties uz bioloģiskajiem neironiem, kā piemēram:

- neironā atmiņa ir sadalīta;
- īslaicīgā atmiņa ir signāli, kurus izsūta neironi;
- ilglaicīgā atmiņa atrodas neironu sinapsēs;
- svarus sinapsēs var mainīt balstoties uz pieredzi;
- sinapsēm var būt gan pastiprinoša, gan pavājinoša iedarbība;
- informācija tiek apstrādāta lokāli. [10]

Tā kā vispārīgā gadījumā neironu tīkls sastāv no liela skaita neironu, var uzskatīt, ka neirons ir salīdzinoši vienkārša neironu tīkla sastāvdaļa. Tā kā neironu tīklā galvenā loma ir nevis konkrētām tā sastāvdaļām, bet gan neironu savstarpējam novietojumam, neironu tīklu ir grūti vispārīgi modelēt, testēt un lietot. Tieši šī īpašība neironu tīklu nepadara par unikālu visu problēmu atrisinājumu. Lielā daļā problēmu joprojām labākus rezultātus sasniedz standarta algoritmi. Neironu tīklu kā risinājumu problēmai tā sarežģītās struktūras dēļ vērts izmantot problēmu risināšanai, kur parastā standarta pieeja nesniedz gaidīto rezultātu jeb ir neefektīva. Otrs variants, kad vērts apsvērt neironu tīkla izmantošanu, ir gadījumos, kad informācija par problēmu nav pilnīga. Par spīti tam, ka par neironu tīklu var spriest tikai, ja tam ir daudz elementu, ir vērts aplūkot arī atsevišķa neirona uzbūvi, jo tā var būtiski ietekmēt skaitlošanas jaudu. [28]

Mākslīgais nerons sastāv no svariem, summēšanas jeb izplatīšanas funkcijas un aktivitātes funkcijas. Tā shēma un galvenās sastāvdaļas redzamas 2.12. attēlā, bet katru no sastāvdaļām apskatīsim smalkāk.



Att. 2.12. Mākslīgā neirona shematisks zīmējums [9].

Svari, saukti arī par sinaptiskiem svariem (angl. *synaptic weights*) ir vērtības, ar kuru palīdzību neironā ienākošais signāls tiek izmainīts. Svari lielā mērā nosaka to, cik ļoti neirons atšķiras no neirona, jo visam neironu tīklam vai atsevišķai neironu grupai izplatīšanās un aktivitātes funkcijas parasti ir vienādas. Tāpat arī svari ir atbildīgi par neironu tīkla mācīšanās spēju, jo tos mainot iespējams pielāgot neironu tīklu atrisināt konkrētu problēmu. Parasti svaru skaits atbilst ieeju skaitam, bet var būt arī viens papildus svars. Svarus no angļu valodas vārda *weight* apzīmē ar burtu w un indeksu, kas parāda ieejas kārtas numuru. Visu svaru apzīmēšanai izmanto matricu pierakstu (w vai W).[28]

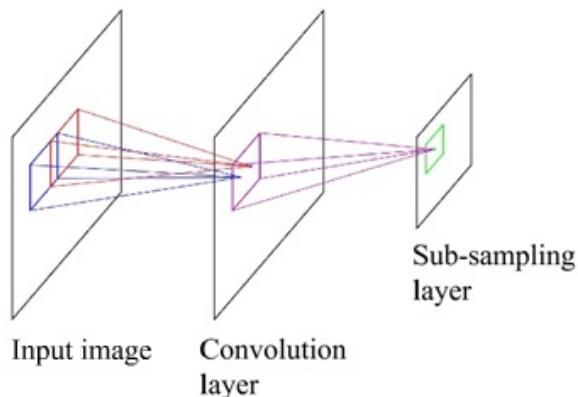
Summēšanas funkcija (angl. *propagation function*) ir atbildīga par visu ienākošo signālu kombinēšanu ar to svariem un kopējās vērtības padošanu tālāk aktivitātes funkcijai. Parasti izmanto signālu un to atbilstošo svaru reizinājumu summu kā summēšanas funkciju. Šīs funkcijas aprēķināto vērtību parasti apzīmē ar burtu virkni *NET*.[28]

Aktivitātes funkcija ir svarīgākā neirona darbības raksturotāja. Aktivitātes funkcija nosaka to, kāda tipa problēmu risinās nerons. Pati funkcija, izmantojot summēšanas funkcijas galarezultāta vērtību *NET*, aprēķina izejas vērtību.[28]

Konvolucionālie neironu tīkli

Konvolucionālais neironu tīkls (angl. *convolutional neural network*) ir īpašs mākslīgo neironu tīkla veids, ko bieži izmanto plašā spektrā tēlu pazīšanas problēmu, kā piemēram, datorredzē (angl. *computer vision*), runas pazīšanā (LZA, angl. *speech recognition*) un cītur. Turpmākajā aprakstā apskatīsim tikai divdimensionālus konvolucionālos neironu tīklus. Konvolucionālā neironu tīkla galvenā ideja ir veidot invariantas īpašības neironu tīkliem, veidojot tādus modeļus, kas arī ir invarianti noteiktiem ievades datu pārveidojumiem. Ideja ir radusies no problēmas, kas ir bieži sastopama vienvirziena neironu tīklos (angl. *feed forward neural networks*) un it īpaši vairākslāņu vienvirziena neironu tīklos. Problēma ir tajā, ka vairākslāņu vienvirziena neirona tīklam slāņi savā starpā ir savienoti un tas noņem ievades telpisko informāciju, kas nepieciešama aprēķiniem.[12]

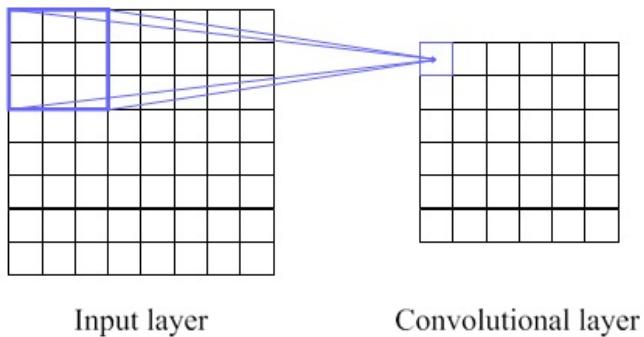
Atšķirībā no parastajiem neironu tīkliem, konvolucionālajiem neironu tīkliem ir īpaša arhitektūra. Arhitektūra parasti sastāv no konvolucionālā slāņa un apakšparaugas atlases slāņa (LZA, angl. *sub-sampling layer*) kā redzams attēlā 2.13.. Konvolucionālais slānis



Att. 2.13. Konvolucionālā neironu tīkla arhitektūra [24].

implementē konvolūcijas operāciju un apakšparaugas atlases operāciju norisinās apakšparaugas atlases slānī. Konvolucionālie neironu tīkli ir veidoti balstoties uz trīs galvenajām idejām - vietējiem uztverošajiem laukumiem, svaru dalīšanu un apakšparaugu atlasi jeb savstarpēju izmantošanu (LZA, angl. *pooling*). Katru ideju apskatīsim sīkāk. [24]

Vietējie uztverošie laukumi Vienvirziena neironu tīklā ievade ir cieši saistīta ar nākamo slēpto mezglu katram neironam. Turpretī konvolucionālā neironu tīklā ievade veido savieno-

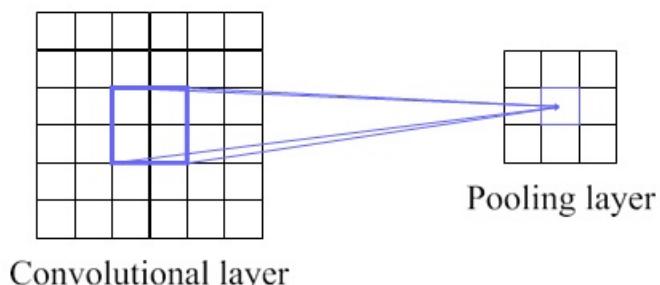


Att. 2.14. Vietējā uztverošā laukuma piemērs konvolucionālajā slānī [24].

jumus tikai mazā reģionā. Katrs nerons slēptajā slānī būs savienots tikai ar mazu laukumu no iepriekšējā slāņa, ko sauc par vietējo uztverošo laukumu. 2.14. attēlā ar violetu kvadrātu apzīmēts vietējais uztverošais laukums, bet ar violetajām līnijām, kā tas savienots ar neuronu. [24]

Svaru dalīšana Konvolūcjonālajā slānī neuroni ir izkārtoti vairākos paralēlos slēptajos slāņos jeb iezīmju kartēs (angl. *feature map*). Katrs nerons iezīmju kartē ir savienots ar vietējo uztverošo laukumu. Katrai iezīmju kartei visi neuroni dala vienu un to pašu svara parametru, kas ir pazīstams kā filtrs.[24]

Savstarpējā izmantošana Konvolucionālais neuronu tīkls satur ne tikai konvolucionālos slāņus, bet dažreiz arī savstarpējās izmantošanas slāņus (angl. *pooling layers*). Gadījumos, kad šis slānis eksistē, to parasti izmanto uzreiz pēc konvolucionālā slāņa. Tas nozīmē, ka konvolucionālā slāņa izvads ir ievads savstarpējās izmantošanas slānī. Šis slānis ir paredzēts, lai ġenerētu translācijas invarianta iezīmes, aprēķinot statistiku no konvolūciju aktivācijas mazam uztveršanas laukumam, kas atbilst iezīmju kartei. Šeit mazā uztverošā laukuma izmērs ir atkarīgs no savstarpējās izmantošanas izmēra. 2.15. attēlā redzams, kā savstarpējās izmantošanas slānis darbojas katrai iezīmju kartei.[24]



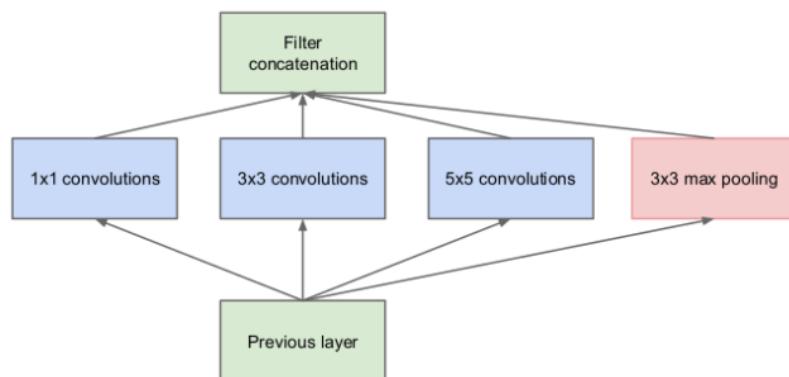
Att. 2.15. Savstarpējās izmantošanas slāņa piemērs iezīmju kartei [24].

Bibliotēka *Tensorflow* un *Inception* modelis

TensorFlow ir atvērtā koda mašīnmācīšanās metožu bibliotēka. Skaitļošanu, kas veikta izmantojot *TensorFlow*, iespējams izpildīt ar nelielām vai bez izmaiņām uz dažādām sistēmām, sākot ar mobilām ierīcēm līdz pat plaša mēroga dalītām sistēmām, kas sastāv no simtiem mašīnu un tūkstošiem skaitļošanas vienību. Fakts, ka eksistē viena sistēma, kas noklāj tik plašu platformu loku, ievērojami atvieglo reālās dzīves pielietojumu mašīnmācīšanās sistēmām. Sistēmu ir viegli pielāgot un to var izmantot, lai izteiku dažādus algoritmus, veiktu pētījumus un uzstādītu mašīnmācīšanās sistēmas produkcijas vidē vairāk kā desmit datorzīnātņu nozarēs, ieskaitot runas pazīšanu, datorredzi, robotiku, informācijas izguvi, dabīgo valodu apstrādi, ģeogrāfiskās informācijas izdalīšanu un citās. [1]

Google Brain projekts sākās 2011. gadā ar mērķi izpētīt ļoti plaša mēroga dzīlās apmācības neironu tīklu pielietojumu izpētē un, lai izmantotu *Google* produktos. Viens no pirmajiem produktiem projekta ietvaros tika izveidots *DistBelief* - pirmās paaudzes mērogjama, sadalīta apmācības un secinājumu sistēma. Balstoties uz pieredzi, kas gūta, strādājot ar *DistBelief* un lielāku sapratni par vēlamās sistēmas īpašībām un prasībām apmācībai un neironu tīkla izmantošanā, tika radīts *TensorFlow* - otrs paaudzes sistēma implementācijai un plaša mēroga mašīnmācīšanās modeļu uzstādīšanai. [1]

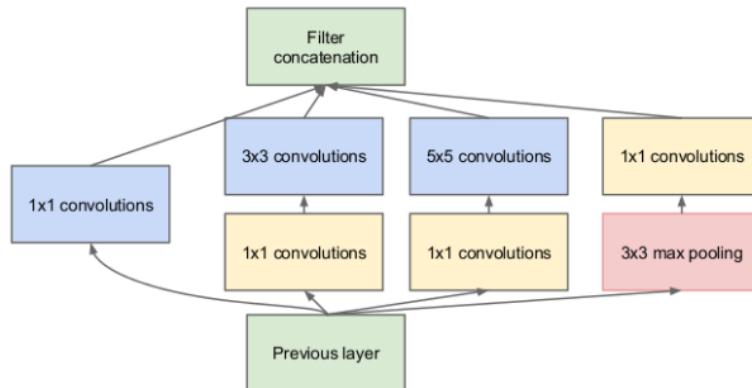
TensorFlow ir atvērtā koda programmatūras bibliotēka cipariskām skaitļošanām, izmantojot datu plūsmas grafus (LZA, angl. *data flow graph*). Mezgli grafo apzīmē matemātiskās operācijas, bet grafu malas daudzdimensiju datu masīvus (tenzorus), kas starp mezgliem plūst. [1]



Att. 2.16. *Inception* modulis [25].

Viens no *TensorFlow* iekļautajiem modeļiem ir *Inception* modelis. *Inception* modeļa arhitektūras galvenā pamatideja ir balstīta uz to retināta struktūra konvolucionālā redzamības tīklā var tikt tuvināti nosepta ar blīvām komponentēm. Viss, kas ir vajadzīgs, ir atrast optimālu vietējo konstrukciju un to telpiski atkārtot. Viens no risinājumiem ir konstrukcija slāni pa slānim, kur būtu jāanalizē korelācijas statistika pēdējam slānim un tad pudurot tos grupās ar augstu korelāciju. Šie puduri veidos nākamā slāņa vienības un būs savienoti ar iepriekšējo slāni. Varam pieņemt, ka katrā vienība no iepriekšējā slāņa atbilst kādam reģionam no ievadattēla un šīs vienības ir sagrupētas filtru bankās. Zemākajos slāņos jeb tajos,

kas tuvāki ievadei, korelētās vienības veidotu vietējos reģionus. Tas nozīmē, ka beigās tiktu iegūts daudz koncentrētu puduru vienā reģionā un tos iespējams noklāt ar 1×1 konvolūcijām nākamajā slānī.[25]



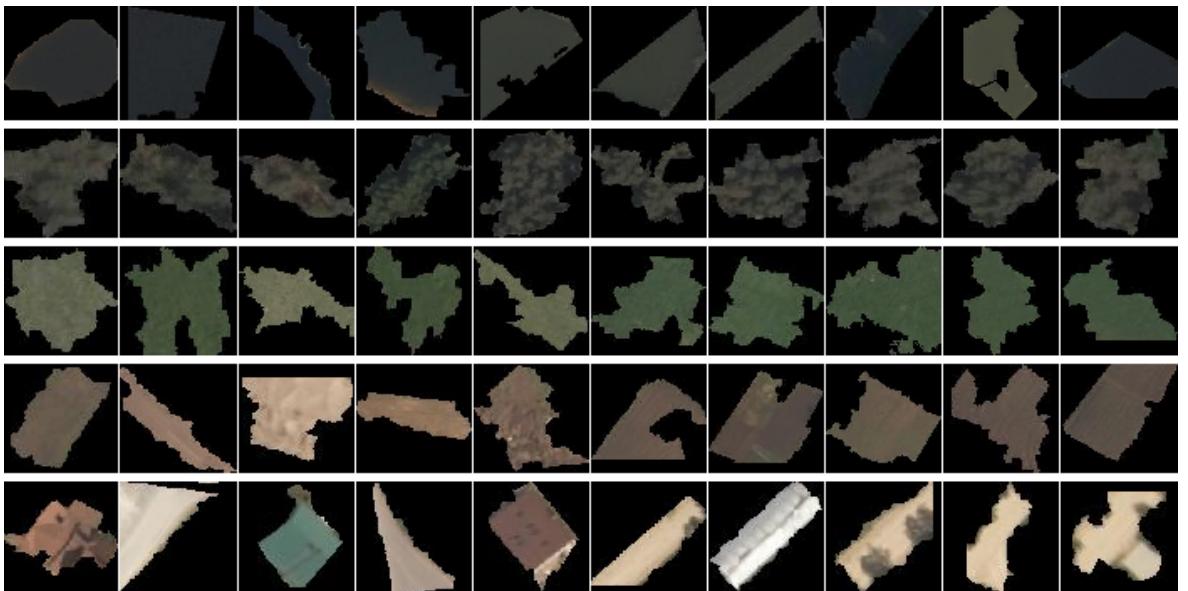
Att. 2.17. *Inception* modulis ar dimensiju skaita samazināšanu [25].

Lai izvairītos no problēmām, kas rodas apgabalu salāgošanas dēļ, tagadējās *Inception* arhitektūras iemiesojumi izmanto 1×1 , 3×3 un 5×5 filtru izmērus. Tas arī nozīmē, ka ieteiktā arhitektūra ir visu slāņu un to izvades filtru banku kombinācija apvienota vienā izvades vektorā, kas veido nākamā līmeņa ievadi. Visu apkopojot, *Inception* tīkls ir tīkls, kas sastāv no augstāk aprakstītā tipa moduļiem, izvietoti viens virs otra, ar laiku pa laikam esošiem maksimālas savstarpējās izmantošanas slāniem ar soli sākot no divi līdz pusei no tīkla izšķirtspējas. [25]

2.2. 3. Neironu tīkla izmantošana JSEG reģionu klasifikācijai

Nākamais solis aiz parauga datu sagatavošanas JSEG segmentācijas rezultātu klasifikācijā ir mākslīgā neironu tīkla apmācība. Kā pamats tiek ņemts repozitoriju uzglabāšanas platformā *GitHub*, *TensorFlow* projekta, *Tensorflow* repozitorijā esošais, brīvi pieejamais kods, kas paredzēts *Inception* modeļa pēdējā slāņa vairākkārtējai apmācībai attēlu klasifikācijā un klašu piešķiršanai klasificējamiem attēliem.

Inception modelis ir jau iepriekš apmācis attēlu klasificēšanā un katru reizi, apmācot tā pēdējo slāni, iespējams modelim iemācīt atpazīt jaunas attēlu klases. Šāda pieeja ne tikai ievērojami saīsina apmācības laiku no vairākām nedēļām līdz nepilnai stundai, bet arī sniedz pārsteidzoši labus rezultātus. Modelis ir pilnībā apmācis, izmantojot attēlus no *ImageNet* attēlu datu kopas. Lai atkārtoti apmācītu pēdējo modeļa slāni, nepieciešama direktorija, kas satur apakšdirektorijas ar parauga datiem. Īpaša uzmanība jāpievērš apakšdirektoriju nosaukumiem, jo tieši tie nosaka atpazīstamo klašu nosaukumus. 2.2. 1. apakšnodalā jau apskatījām, kā izveidot parauga datu attēlus ar atbilstošajām apakšdirektorijām. Jāpiebilst, ka parauga datu attēlu nosaukumi nav svarīgi, tādēļ šajā gadījumā attēlu nosaukumi atbilst to reģionu kārtas numuriem, kas radušies JSEG segmentācijas rezultātā. Attēlā 2.18. redzami daži no apmācībā izmantotiem attēliem - 1. rindā ūdens pārseguma tipam, 2. rindā koku pārseguma tipam, 3. rindā zāliena pārseguma tipam, 4. rindā lauku pārseguma tipam un 5. rindā citam pārseguma tipam.



Att. 2.18. Zemes pārseguma tipu parauga datu attēli.

Apmācībā un attēlu klasificēšanā nepieciešams uz darbsistēmas instalēt programmēšanas valodu *Python*. Lai *TensorFlow* bibliotēka sekmīgi tiktu importēta, nepieciešams instalēt vismaz *Python 3.5.2* versiju. Tāpat veiksmīgai apmācības un klasificēšanas skriptu izpildei nepieciešams papildus instalēt bibliotēkas *numpy+mkl* un *scipy*. Lai veiktu apmācību, no komandrindas, atrodoties direktorijā, kur pieejams brīvi pieejamais atvērtā koda *Python* skripts *retrain.py*, šo skriptu izpilda. Kā parametrs skripta izpildē tiek padots apakš-direktorijas nosaukums, kur saglabātas direktorijas ar parauga datiem.

```
1 python retrain.py --image_dir paraugaDati
```

Pēc veiksmīgas modeļa pēdējā slāņa otrreizējas apmācības iespējams veikt JSEG segmentācijas rezultātā radušos reģionu klasifikāciju. Līdzīgi, kā apmācības datu sagatavošanā, arī segmentēto ortofotokarti nepieciešams sadalīt atsevišķos attēlos katram reģionam. Lai to īstenotu, izmanto 2.2. 1. apakšnodaļā minēto funkciju *saveTestDataByRegion*, pirms tam to attiecīgi pielāgojot. Kā pirmais pielāgojums minams cikla sākšana ar pirmo elementu, jo atšķirībā no parauga datu reģionu attēlu sagatavošanas, segmentētās ortofotokartes reģionu attēlā nav nulles vērtības. Otrs pielāgojums saistīts ar nosacījuma bloka dzēšanu, kas atbild par mazo laukumu (mazāki par 80 pikseliem) neiekļaušanu parauga datos. Tā kā nepieciešams klasificēt visus reģionus, šis nosacījuma bloks tiek dzēsts. Pēc augstāk minētās funkcijas izpildes tiek izveidota direktorija, kas satur visus reģionu attēlus. Pēc tam šo direktoriju pārvieto uz to direktoriju, kur atrodas brīvi pieejamais atvērtā koda *Python* skripts *label_image.py*, lai direktorija un skripts atrastos vienādā līmenī. Šajā direktorijā izveido arī terminālī izpildāmu skriptu, kas ciklā apskata direktoriju ar reģionu attēliem un katram attēlam izsauc skriptu, kas nosaka attēla klasi.

Vispirms apskatīsim skriptu *runLabelScript.sh*, kas atbildīgs par *label_image.py* skripta izpildi ciklā katram attēlam no reģionu direktorijas. Skripta pirmajā rindiņā mainīgajā

FOLDER nodefinēts pilnais ceļš uz direktoriju ar reģionu attēliem. Pēc tam ciklā, izmantojot katru direktorijā esošā faila nosaukumu tiek izpildīts *Python* skripts *label_image.py*, kam kā parametrs tiek padots pilnais direktorijas ceļš ar apskatītā faila nosaukumu.

```

1 FOLDER="C:\\image_retraining\\regions\\"
2 for file in $(ls $FOLDER); do
3     python label_image.py "${FOLDER}${file}"
4 done

```

Tālāk apskatīsim *Python* skriptu *label_image.py*, ko izpilda katram attēlam iepriekš apskatītais skripts *runLabelScript.sh*. Lai varētu veikt reģionu attēlu klasifikāciju, pirms *runLabelScript.sh* izpildes jāpārliecinās par atsevišķu failu atrašanās vietām. Pēc sekmīgas *Inception* modeļa pēdējā slāņa otrreizējas apmācības skripts *retrain.py* izveido divus failus C:\\tmp direktorijā - *output_labels.txt* un *output_graph.pb*. Šādas vērtības definētas arī *label_image.py* failā, ko nepieciešams pielāgot, ja šie divi faili atrodami citā vietā. *output_labels.txt* satur tos klašu jeb direktoriju nosaukumus, kas saturēja parauga datus, kas tika izmantoti apmācībai. *output_graph.pb* fails satur *TensorFlow* grafus.

Jau iepriekš instalējām bibliotēkas, kas nepieciešamas skriptā *label_image.py*. Bez to importēšanas un padotā parametra vērtības nolasīšanas vēl faila sākumā tiek izteikts un izvadīts reģiona numurs, ko iegūst no attiecīgi padotā attēla faila nosaukuma.

```

1 import tensorflow as tf
2 import sys, os, csv
3 from scipy import misc
4
5 image_path = sys.argv[1]
6
7 regionNr = ...
    os.path.splitext(os.path.basename(image_path))[0].lstrip("0")
8 print(regionNr)

```

Pēc tam tiek ielasīts attēls, kuru nepieciešams klasificēt un tiek izveidots vietturis (LZA, angl. *placeholder*), kas vēlāk tiks izmantots, lai caur to grafos ievadītu datus. No faila *output_labels.txt* tiek izveidots masīvs *label_lines*, kas satur piešķiramo klašu nosaukumus.

```

1 images_placeholder = tf.placeholder(tf.int32)
2 label_lines = [line.rstrip() for line
3                 in tf.gfile.GFile("C:\\tmp\\output_labels.txt")]

```

Izmantojot iepriekš ar *retraining.py* palīdzību izveidoto failu *output_graph.pb*, tiek definēts grafu objekts mainīgajā *graph_def*.

```

1 with tf.gfile.FastGFile("C:\\tmp\\output_graph.pb", 'rb') as f:

```

```

2     graph_def = tf.GraphDef()
3     graph_def.ParseFromString(f.read())
4     _ = tf.import_graph_def(graph_def, name='')
```

Kad visi nepieciešamie resursi ir sagatavoti, iespējams noteikt konkrētā reģiona pie-derību kādai klasei. Lai to izdarītu, tiek atvērta *TensorFlow* sesija, kurā nosaka to zemes pārseguma tipu, kam ir procentuāli vislielākā atbilstība klasificējamam reģionam. Vispirms tiek iegūts saraksts ar visām zemes pārseguma tipu klasēm un atbilstošās procentuālās iespējamības, ka noteiktais reģions pieder noteiktai klasei. Pēc tam klases tiek sarindotas sākot ar to, kam ir visaugstākā procentuālā iespējamība.

```

1 with tf.Session() as sess:
2     softmax_tensor = sess.graph.get_tensor_by_name('final_result:0')
3     predictions = sess.run(softmax_tensor, { 'DecodeJpeg:0': im})
4     top_k = predictions[0].argsort()[-len(predictions[0]):][::-1]
```

Mainīgajā *cover_type* saglabā zemes pārseguma tipa numuru, kam ir procentuāli vislielākā atbilstība klasificējamam reģionam. Pēc tam *csv* failā tiek pievienota rinda ar reģiona numuru un kārtas numuru atbilstošajam zemes tipam.

```

1 cover_type = label_lines[top_k[0]]
2
3 with open('classified_image.csv', 'a', newline='') as csvfile:
4     writer = csv.writer(csvfile, delimiter=',', quotechar='|', ...
5                         quoting=csv.QUOTE_MINIMAL)
6     writer.writerow([regionNr, cover_type])
```

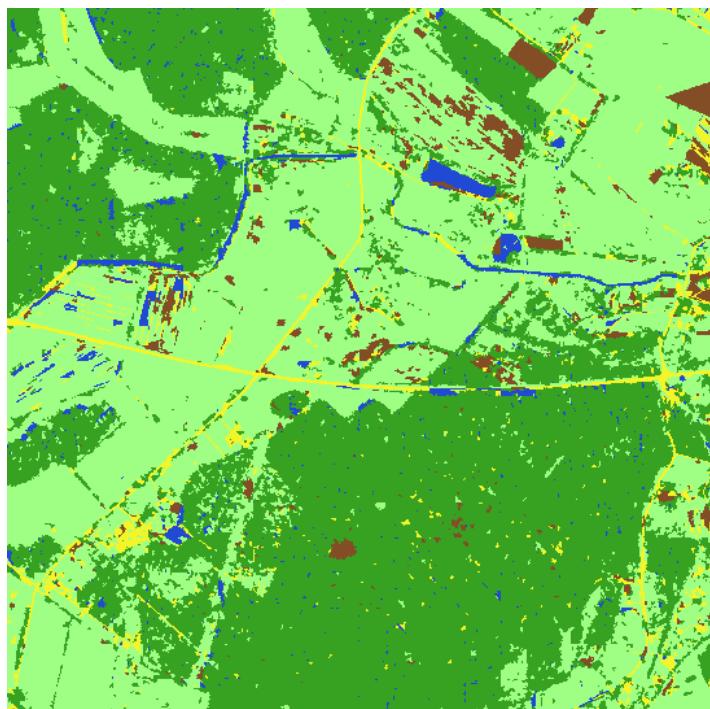
Kad *Python* skripts *label_image.py* atradis katram reģiona attēlam atbilstošo klases vērtību, *csv* failu ielasa programmatūrā *MatLab*, kā ciparu masīvu un, izmantojot iebūvēto *MatLab* funkciju *sortrows*, sakārto rindiņas augošā secībā pēc reģiona kārtas numura. Tad no šī divu kolonnu masīva, izmantojot *MatLab* skriptu, izveido ortofotokartes izmēra attēlu, kur pikseļu vērtība atbilst klasei, kurai tie pieder.

Tālāk apskatīsim šo skriptu, ar kura palīdzību tiek izveidota klašu maska. Iepriekš nepieciešams ielasīt ortofotokartes attēlu un to saglabāt mainīgajā *rgb_image*. Tāpat arī nepieciešams izveidot rindas vektoru ar pārseguma tipu numuriem augošā secībā un ielasīt reģionu attēlu. Kad iegūti ortofotokartes izmēri, izveido klašu maskas matricu, kas sastāv no nullēm. Tad ciklā katram klasificētam reģionam ciklā atrod klases numuru, kam reģions pieder. Beigās tajos klašu maskas punktos, kas atbilst apskatītā reģiona numuram, nulles aizstāj ar piešķirtās klases numuru. Gala rezultāts pēc klasifikācijas apskatāms 2.19. attēlā.

```

1 [r k d]=size(rgb_image);
2 mask=zeros(r,k);
3 for i=1:length(classified_image)
```

```
4 for j=1:length(sample_classes)
5     if classified_image(i,2)==sample_classes(j)
6         i
7         mask(region==classified_image(i,1))=j;
8     end
9 end
10 end
```



Att. 2.19. Klasifikācijas, izmantojot neironu tīklus, rezultāts.

3. Validācija un rezultāti

Pētījumā tika apskatīta krāsu attēlu segmentācijas metode JSEG un citas attēlu segmentācijas metodes, lai noteiktu JSEG metodes priekšrocības un trūkumus salīdzinoši ar citiem krāsu attēlu segmentēšanas algoritmiem. Balstoties uz JSEG metodē iekļautā reģionu apvienošanas algoritma jēgpilnās reģionu grupās vājajiem rezultātiem, tika apskatīta k-tuvāko kaimiņu metode un dziļās apmācības neuronu tīkli segmentācijas reģionu klasifikācijai. Lai veiktu zemes pārseguma tipu noteikšanu augstas telpiskās izšķirtspējas tālizpētes datos, tika izstrādāta darbplūsmas abām reģionu klasifikācijas metodēm.

Lai pārliecinātos par metodes iespējām un rezultāta pareizību, kā arī, lai veiktu nepieciešamos secinājumus, nepieciešams veikt metodes validāciju. Lai veiktu metodes validāciju, izmantojot programmatūru *QGIS*, tiek izveidots validācijas komplekts. Validācijas komplekts $10\ 000 \times 10\ 000$ pikseļu lielam attēlam sastāv no 729 punktiem, kas viens no otra izvietoti 375 pikseļu attālumā režģa formā, kas dabā rezultējas 150 metros. Tabulā 3.1. redzama katras klases izplatība un kā jau redzams, visvairāk sastopami tiesi koku un zālienу pārseguma tipi, kas izskaidrojams ar to plašo izplatību dabā.

Tabula 3.1. Validācijas punktu sadalījums pa klasēm

Klases numurs	Klases nosaukums	Skaits
1	Ūdens	9
2	Koki	365
3	Zāliens	308
4	Lauki	8
5	Cits	39



Att. 3.1. Validācijas komplekts.

Attēlā 3.1. redzams validācijas komplekts un katra punkta atrašanās vieta. Ar tumši zaļo krāsu attēlā apzīmēti punkti, kas pieder klasei "Koki", ar gaiši zaļo krāsu - punkti, kas

pieder klasei "Zāliens", ar dzelteno krāsu - punkti, kas pieder klasei "Cits", ar zilo krāsu - punkti, kas pieder klasei "Ūdens", bet ar brūno krāsu - punkti, kas pieder klasei "Lauki". Katram punktam klase ir manuāli piešķirta.

Validācija tiek veikta ielasot no faila vērtības, kur vienā kolonnā atrodama punkta "x" koordināta, nākamajā kolonnā punkta "y" koordināta, trešajā kolonnā punkta identifikācijas numurs, kas netiek izmantots, bet pēdējā, punktam piešķirtās klases numurs. Tad interesējošam reģionam tiek izveidota matrica, kur rindu skaits vienāds ar klašu skaitu, bet kolonnu skaits vienāds ar punktu skaitu un, kur vērtības binārā formā reprezentē, vai konkrētais punkts pieder konkrētajai klasei. Tāda pati matrica tiek izveidota arī klasificētajiem reģioniem un pēc tam, izmantojot *MatLab* iebūvēto funkciju *plotconfusion*, tiek iegūta kļūdu matrica, kas parāda, kuri punkti klasificēti pareizi, bet kuri ne.

3.1. JSEG segmentācijas rezultātu klasifikācijas ar k-tuvāko kaimiņu metodi validācijas rezultāti

Veicot validāciju, izmantojot programmatūru *MatLab*, tās iebūvēto funkciju *plotconfusion* un iepriekš sagatavoto validācijas komplektu, tika iegūta 3.2. attēlā redzamā kļūdu matrica.

Confusion Matrix						
Output Class	1	2	3	4	5	
	3 0.4%	29 4.0%	2 0.3%	0 0.0%	2 0.3%	8.3% 91.7%
	4 0.5%	285 39.1%	6 0.8%	0 0.0%	1 0.1%	96.3% 3.7%
	2 0.3%	34 4.7%	258 35.4%	2 0.3%	9 1.2%	84.6% 15.4%
	0 0.0%	0 0.0%	2 0.3%	4 0.5%	0 0.0%	66.7% 33.3%
	0 0.0%	17 2.3%	40 5.5%	2 0.3%	27 3.7%	31.4% 68.6%
	33.3% 66.7%	78.1% 21.9%	83.8% 16.2%	50.0% 50.0%	69.2% 30.8%	79.1% 20.9%
Target Class						

Att. 3.2. Kļūdu matrica klasifikācijai ar k-tuvāko kaimiņu metodi.

Metodes precizitāte ir 79,1%. Visaugstākā precizitāte - 83,8% ir "zāliena" pārseguma tipa klasei, bet viszemākā precizitāte ir, nosakot reģionu piederību "ūdens" pārseguma tipam - 33,3%. "Ūdens" pārseguma tips vienlīdz bieži tika sajaukts kā ar "koku" pārseguma tipu, tā "zāliena" pārseguma tipu.

”Koku” pārseguma tipam piederošie punkti vienlīdz bieži tika kļūdaini uzskatīti par ”ūdens” un ”zāliena” klasēm piederošiem punktiem. Tas, ka punkti tika uzskatīti par piederošiem ”zāliena” klasei skaidrojams ar plašajām purvainajām teritorijām un jaunaudzēm. Kļūdaini klasificētās ”ūdens” teritorijas ”koku” klasei piederošiem punktiem skaidrojamas ar vizuālo līdzību starp ūdens teritorijām un ēnām, ko veido koku vainagi.

”Zāliena” pārseguma tipam piederošie ortofotokartes punkti 40 gadījumos no 308 tika uzskatīti, ka pieder ”citam” zemes pārseguma tipam. Tā kā ”cita” pārseguma tipa apmācības datos dominēja reģioni ar ceļa seguma fragmentiem, iespējams, daudzas iebrauktas joslas zālājos sekmēja šādu kļūdaiņu klasifikāciju.

”Lauku” pārseguma tipa precizitāte ir 50% un tā vienlīdz bieži tika sajaukti ar ”zāliena” un ”citu” pārseguma tipu.

”Citam” pārseguma tipam atbilstošie punkti visbiežāk kļūdaini tika klasificēti kā piederoši ”zāliena” pārseguma tipam, bet tā precizitāte ir 69,2%

Vērojams, ka augstāka precizitāte novērojama tām zemes pārseguma tipa klasēm, kuru apmācībai ne tikai tika izmantots vairāk apmācības datu, bet arī ir vairāk validācijas datu jeb zemes pārseguma tips ir izplatītāks.

3.2. JSEG segmentācijas rezultātu klasifikācijas, izmantojot neuronu tīklus, validācijas rezultāti

Veicot validāciju, izmantojot programmatūru *MatLab*, tās iebūvēto funkciju *plotconfusion* un iepriekš sagatavoto validācijas komplektu, tika iegūta 3.3. attēlā redzamā kļūdu matrica.

Confusion Matrix											
Output Class	1	2	3	4	5						
	1	7 1.0%	9 1.2%	2 0.3%	0 0.0%	1 0.1%	36.8% 63.2%				
	2	2 0.3%	296 40.6%	15 2.1%	3 0.4%	4 0.5%	92.5% 7.5%				
	3	0 0.0%	52 7.1%	276 37.9%	0 0.0%	12 1.6%	81.2% 18.8%				
	4	0 0.0%	2 0.3%	12 1.6%	5 0.7%	2 0.3%	23.8% 76.2%				
	5	0 0.0%	6 0.8%	3 0.4%	0 0.0%	20 2.7%	69.0% 31.0%				
						77.8% 22.2%	81.1% 18.9%	89.6% 10.4%	62.5% 37.5%	51.3% 48.7%	82.9% 17.1%
Target Class						1	2	3	4	5	

Att. 3.3. Kļūdu matrica klasifikācijai, izmantojot neuronu tīklus.

Klasificējot reģionus, izmantojot neironu tīklus, tika sasniegta 82,9% precizitāte. Līdzīgi, kā klasifikācijā, izmantojot k-tuvāko kaimiņu metodi, arī šajā metodē visaugstāko precizitāti - 89,6% sasniedza to reģionu klasifikācijā, kas atbilst "zāliena" pārseguma tipam. Viszemāko precizitāti - 51,3 % iespējams novērot nosakot piederību "citam" zemes pārseguma tipam.

"Ūdens" pārseguma tipam atbilstošie validācijas punkti tika klūdaini klasificēti divas reizes kā piederoši "koku" pārseguma tipam, kas gala rezultātā rezultējās 77,8% precizitātē.

"Koku" pārseguma tipam piederošie punkti visbiežāk - 52 reizes tika klūdaini uzskaņoti par piederošiem "zāliena" pārseguma tipam. Šādu klūdu iespējams skaidrot ar plašajām purvainajām teritorijām un jaunaudzēm, kas attiecīgajā apgabalā ir plaši izplatītas.

"Zāliena" klasei piederošie punkti gandrīz vienādi bieži tika jaukti ar "koku" un "lauku" pārseguma tiem. "Koku" pārseguma tipa gadījumā klūda skaidrojama ar ēnām, ko mežu masīvi atstāj zālienā, bet "lauku" pārseguma tipa gadījumā klūdu iespējams skaidrot ar vizuālo līdzību starp pamestām pļavām un tikko apsētām arāzemēm.

"Lauku" pārseguma tipa klasei atbilstošie trīs validācijas punkti no astoņiem klūdaini tika klasificēti kā piederoši "koku" pārseguma tipam. Šāda klūda rezultējās 62,5% precizitātē.

"Citam" pārseguma tipam, kam raksturīga viszemākā precizitāte, visbiežāk piederošie punkti tika klūdaini klasificēti, kā piederoši "zāliena" pārseguma tipam.

Līdzīgi, kā klasificējot JSEG segmentācijas rezultātus, izmantojot k-tuvāko kaimiņu metodi, vislabākie rezultāti tika sasniegti tām zemes pārseguma tipu klasēm, kur bija lielākais apmācības datu un validācijas punktu skaits.

Secinājumi un priekšlikumi

Apkopojot iegūtos rezultātus tiek secināts, ka darba mērķis un tā veikšanai plānotie uzdevumi ir izpildīti. Tāpat autore secina, ka:

1. JSEG segmentēšanā svarīga loma ir tādiem parametriem, kā attēla kvantizācijā izmanto krāsu skaitam un attēla apakšapgabala izmēriem, kam tiek pielietots segmentēšanas algoritms. Šo parametru savstarpējā attiecība ietekmē to, cik smalki tiks segmentēts apskatītais attēls. Testējot parametrus, tika secināts, ka labāko rezultātu iespējams iegūt 500×500 pikselu lielam apakšapgabalam, izmantojot 12 krāsas attēla kvantizāciju.
2. JSEG segmentēšanas algoritma izpildes laiku iespējams saīsināt, samazinot attēla apakšapgabala izmērus. Viena un tā paša izmēra attēlam, izvēloties 4 reizes vairāk apakšapgabalu izpildes laiks saīsinājās par 17%.
3. Izvēloties ortofotokartes apakšapgabalu skaitu, jārēķinās, ka lielāks apakšapgabalu skaits garantē arī to, ka ortofotokarte tiks segmentēta vairāk un smalkākos reģionos.
4. Segmentēto reģionu klasifikācijā svarīga loma ir ne tikai tajā, cik labi parauga dati atbilst klasificējamam reģionam, bet arī to skaitam un zemes pārseguma tipa dažādo īpašību reprezentēšanai.
5. Segmentēto reģionu klasifikācija, izmantojot neironu tīklus, uzrādīja par 3,8 % augstāku precizitāti nekā šo pašu reģionu klasifikācija, izmantojot k-tuvāko kaimiņu metodi.
6. Reģionu deskriptoru vektoru izveidošana, izmantojot reģionu attēlu poligonizāciju programmatūrā *QGIS* un deskriptoru aprēķināšana, izmantojot *Python* skriptu aizņem 100 reizes mazāk laika, kā šī paša rezultāta sasniegšana, izmantojot programmatūru *MatLab*.
7. Reģionu klasifikācija, izmantojot k-tuvāko kaimiņu metodi, aizņem 40 reizes mazāk laika, kā tāda paša apjoma reģionu klasifikācija, izmantojot neironu tīklus.
8. Pamatojoties uz klasifikācijas metodes izpildes laiku un tā precizitāti, k-tuvāko kaimiņu metode sniedz labākus rezultātus kā reģionu klasificēšana, izmantojot neironu tīklus.

Pētījuma gaitā autore izvirza sekojošus priekšlikumus:

1. Abu pētījumā apskatīto klasifikācijas metožu precizitāti iespējams paaugstināt izvēloties vairāk parauga datu. Tāpat nepieciešams pievērst vairāk uzmanības parauga datu kvalitātei un tam, cik labi tie reprezentē konkrētā pārseguma tipa raksturīgākās īpašības.
2. Veicot apmācību un sagatavojot parauga datus, parauga datiem vienmērīgi jānoklāj visi zemes pārseguma tipi.

3. Balstoties uz pētījuma apgabalu, vērts ieviest papildus zemes pārseguma tipus, kā piemēram, smilšu segums (lauku ceļi ar grants segumu, karjeri, ūdenstilpju krasti), apdzīvota teritorija (ēkas, viensētas), purvs.
4. Lai uzlabotu skaitlošanas laiku gan segmentācijā, gan klasifikācijā, izmantojot neironu tīklus, nepieciešams palielināt skaitlošanas resursus, izmantojot skaitlošanas klasterus.

GALVOJUMS

Ar šo es, Katrīna Zvaigzne, galvoju, ka maģistra darbs ir izpildīts patstāvīgi, konsultējoties ar darba vadītāju. No svešiem pirmavotiem ņemtā informācija ir norādīta ar atsaucēm, dati un definējumi ir uzrādīti darbā. Šis darbs tādā vai citādā veidā nav nekad iesniegts nevienai citai pārbaudījumu komisijai.

2017.gada _____._____

(paraksts)