Glioomien luokittelu Raman-spektroskopian ja neuroverkkojen avulla

Mikael Mäkelä

Pro gradu -tutkielma
Lokakuu 2025

MATEMATIIKAN JA TILASTOTIETEEN LAITOS

Tarkastajat:
Prof. Ion Petre
Prof. Vesa Halava

TURUN YLIOPISTO, Matematiikan ja tilastotieteen laitos

Pro gradu -tutkielma
Pääaine: Matematiikka
Tekijä: Mikael Mäkelä
Otsikko: Glioomien luokittelu Raman-spektroskopian ja neuroverkkojen avulla
Ohjaaja: Prof. Ion Petre
Sivumäärä: 37 sivua
Aika: Lokakuu 2025

Glioma on ihmisen aivoihin muodostuva syöpäkasvain. Glioman luokittelu on tärkeää oikean hoitosuunnitelman laatimiseksi. APOLLO-prosessissa [18] on tutkittu luokittelun toteuttamista koneoppimisen avulla. Prosessissa käytetään Raman-spektroskopiadataa, jota on kerätty glioomapotilaiden kasvainnäytteistä. Prosessi sisältää datan esikäsittelyn, klusteroinnin ja luokittelun.

Tässä tutkielmassa koulutetaan viisi konvoluutioneuroverkkoa samaan luokitteluongelmaan. Käytössä on sama data ja datan esikäsittely kuin prosessissa, ja tavoitteena oli saavuttaa paremmat luokittelutulokset. Tutkielmassa koulutetuilla neuroverkoilla päästiin lähes yhtä hyviin tuloksiin. Lisäksi tutkielmassa esitellään koneoppimisen ja neuroverkkojen perusteoriaa.

Mallien kouluttamiseeen käytettiin Python-ohjelmointikieltä yhdessä TensorFlow- ja Keras-kirjastojen kanssa.

Asiasanat: glioma, koneoppiminen, neuroverkot, konvoluutioneuroverkot

Sisällys

1 Johdanto

Glioma on yleisin primaarinen aivokasvain, ja eliniänennuste on erityisesti aggressiivisimmissa glioomamuodoissa hyvin heikko. Glioomat voidaan luokitella eri tyyppeihin, ja hoitomuodot määräytyvät luokituksen perusteella [14][24]. Artikkelissa [18] esitetty APOLLO-prosessi on kehitetty hyödyntämään koneoppimista glioman luokittelussa. Glioomien luokittelussa käytetään Raman-spektroskopiadataa, joka on kerätty glioomapotilailta. Prosessi sisältää datan esikäsittelyn, klusteroinnin kasvainsolujen ja ei-kasvainsolujen välillä sekä glioomanäytteiden luokittelun. Luokitteluja tehtiin sekä binäärisenä luokitteluna että kuuden luokan luokitteluna.

APOLLO-prosessissa parhaimmaksi malliksi osoittautui tukivektorikoneen ja satunnaismetsän yhdistelmämalli. Muita testattuja malleja olivat satunnaismetsät ja tukivektorikoneet itsenäisesti, vahvistetut päätöspuut ja konvoluutioneuroverkot. Tämän työn tarkoituksena on kartoittaa tarkemmin konvoluutioneuroverkkojen suoriutumiskykyä luokittelutehtävästä. Konvoluutioneuroverkkojen hyvälle toiminnalle antoivat pohjan artikkelit [20] [8], joissa esitellään konvoluutioneuroverkoilla saatuja hyviä tuloksia Raman-spektroskopiadatalla.

Tutkielman ensimmäisessä luvussa esitellään hyvin yleisellä tasolla glioma, sen luokittelu, yleisyys ja eliniänennusteet. Toisessa luvussa esitellään myös hyvin yleisellä tasolla Raman-spektroskopia ja sen avulla kerätty spektridata. Kolmannessa luvussa siirrytään koneoppimisen puolelle, ja siinä käydään läpi koneoppimisen perusteita sekä mallien arviointia. Luvussa viisi esitellään neuroverkot alkaen yksinkertaisen neuronin rakenteesta ja kouluttamisesta. Luvussa perehdytään myös syvempien neuroverkkojen ja konvoluutioneuroverkkojen toimintaan. Kuudennessa luvussa esitellään klusterointialgoritmi DBSCAN, jota käytettiin APOLLO-prosessissa. Seitsemännessä luvussa esitellään tarkemmin APOLLO-porsessin vaiheet ja tulokset. Kahdeksannessa luvussa on esitelty tutkielman omat mallit ja yhdeksäs luku on yhteenveto. Tämän tutkielman kirjoituksessa on hyödynnetty ChatGPT-tekoälytyökalua kielenhuoltoon ja lähteiden etsimiseen.

2 Glioma

Tässä luvussa esitellään hyvin yleisellä tasolla glioma, joka on primaarisista aivokasvaimista yleisin. [14] Jopa 80% pahanlaatuisista aivokasvaimista on diffuuseja glioomia [4]. Glioma on ihmisen aivojen tukisoluihin muodostuva syöpäkasvain, ja ne kasvavat epätarkkarajaisesti terveen aivokudoksen seassa. Glioomista kaikkein aggressiivisinta tyyppiä kutsutaan glioblastoomaksi [34].

Yhdysvalloissa todetaan vuosittain noin 80000 - 90000 uutta primaarista aivokasvaintapausta, joista noin 25% on glioomia. Glioblastoomia todetaan vuosittain noin 12000, joka on arviolta 15% kaikista uusista aivokasvaimista ja noin 50% kaikista pahanlaatuisista aivokasvaimista [9]. Suomessa vuosina 2013-2017 uusia glioomia todettiin keskimäärin vuodessa 375 kappaletta, joista noin 80% oli pahanlaatuisia. Pahanlaatuisista glioomista noin 60% todettiin glioblastoomiksi. Samalla aikavälillä Suomessa todettiin vuotuisia glioman aiheuttamia kuolemia noin 265 kappaletta, ja näistä noin 60% oli glioblastooman aiheuttamia. Vuosina 2015-2017 kaikkien glioomapotilaiden suhteellinen viiden vuoden elossaolo-osuus oli 33%, ja pahanlaatuisissa tapauksissa viisivuotisennuste oli hieman huonompi [26].

Diagnoosien varhaistumisen ja hoitomuotojen kehittymisten ansiosta glioomapotilaat ovat parempikuntoisia. Glioomien hoito on monivaiheista, ja leikkaustekniikat ja sädehoidon käyttö ovat kehittyneet. Näiden hoitomuotojen lisäksi myös solunsalpaajat ovat vakiinnuttaneet asemansa hoitomuotoina sekä glioman ensivaiheiden että uusiutuineiden kasvainten hoidossa [24]. Glioomia voidaan luokitella eri tavoilla, ja glioman diagnosointi mahdollistaa parhaan hoitosuunnitelman luomisen. Aikaisemmin glioomat jaoteltiin pelkän histologisen piirteen perusteella. Maailman terveysjärjestön (World Health Organization, WHO) vuonna 2016 julkaisemassa keskushermoston kasvainten luokittelussa on käytetty myös kasvainten molekulaarisia muutoksia histologisen luokittelun rinnalla. Histologinen tyyppi, pahanlaatuisuusaste (gradus I-IV) ja molekulaariset muutokset muodostavat yhdessä integroidun diagnoosin [34]. Molekulaarisista muutoksista etenkin isositraattidehydrogenaasi 1 (IDH1) proteiinia koodaavan geenin mutaatiota (IDH1-mutaatio) pystytään käyttämään glioomien luokittelussa [14]. IDH-mutaatio on käytössä itsenäisenä luokitteluperusteena, sillä se on vahva suotuisa ennustetekijä. IDH-mutaation perusteella glioomat voidaan jakaa mutatoituneisiin (IDH-mutant) ja mutatoitumattomiin (IDH-wildtype) kasvaimiin [34]. Artikkelissa [4] esitetyssä tutkimuksessa luotiin DNA-metylaatioklusterit, jotka nimettiin LGm 1-6. Näistä luokat 1-3 kuuluvat IDH-mutaatioon ja luokat 4-6 kuuluvat IDH-villityyppiin. Tässä tutkielmassa käytetään LGm-luokituksia. Glioomasta ja sen luokittelusta on kerrottu tarkemmin ja yksityiskohtaisemmin artikkeleissa [14] [4] [34] [24].

3 Raman-spektroskopia

Tässä luvussa esitellään hyvin yleisellä tasolla *Raman-spekstroskopia*, joka on näytettä vahingoittamaton keino analysoida näytteen molekyylitason koostumusta. Raman-spektroskopia kehitettiin alun perin itsenäisesti 1900-luvun ensimmäisellä puoliskolla Nobel-palkitun Chandrasekhara Venkata Ramanin ja Grigorij Samuilovič Landsbergin toimesta, mutta se vakiintui vasta laserilla varustettujen spektrometrien käyttöönoton myötä vuosisadan jälkipuoliskolla. Raman-spektroskopian kehitys mahdollisti materiaalien tarkemman tutkimuksen [28]. Raman-spektroskopiassa näytteeseen kohdistetaan monokromaattista lasersädettä, joka vuorovaikuttaa näytteen molekyylien kanssa ja aiheuttaa valon siroamisen, josta luodaan Raman-spektri [3]. Suurimpia etuja muihin tekniikoihin, kuten infrapunaspektroskopiaan, verrattuna on se, että veden läsnäolo vaikuttaa signaalin laatuun hyvin vähän. Tämän ansiosta Raman-spektroskopiaa voidaan hyödyntää monissa käyttökohteissa, joissa infrapunaspektroskopia ei ole luotettava [28]. Raman-spektroskopiaa voidaan hyödyntää useilla eri tutkimusaloilla, kuten geologiassa, laboratorioanalyyseissä ja biotieteissä [20].

Raman-spektri koostuu huipuista, pohjatasosta, häiriöstä ja kosmisten säteiden häiriöstä (engl. *cosmic rays interference*). Huiput ovat spektrin osia, joissa intensiteetti kasvaa tasaisesti, ja lokaalin maksimin jälkeen vähenee. Huiput sisältävät paljon informaatiota näytteen molekyylikoostumuksesta. Huippujen sisältämää informaatiota voidaan verrata saman spektrin sekä muiden spektrien huippujen välillä, minkä takia ne on tärkeä saada poimittua erilleen muista analyysejä hankaloittavista spektrin osista. Pohjataso voi siirtää huippujen välistä suhteellista intensiteettiä ympäristön valon, näytteen oman säteilyn tai laitteen vaikutuksesta. [32] Tässä työssä pohjatason siirtymä korjataan käyttämällä AirPLS-algoritmia. Algoritmi esitellään tämän työn luvussa (7.1). Raman-spektroskopia, sen toiminta, käyttökohteet ja spektridatan käsittely esitellään tarkemmin artikkeleissa [3] [28] [32].

4 Koneoppiminen

Tietokoneita käytetään lukuisiin eri tarkoituksiin. Kaikkiin tehtäviin ei ole kuitenkaan onnistuttu kehittämään algoritmia, jolla tehtävä voitaisiin suorittaa. Tällaiset tehtävät ovat usein sellaisia, joita ihmiset suorittavat päivittäin, kuten ihmisen tunnistaminen valokuvasta tai auton ajaminen. Koneoppimisen ideana on opettaa tietokone suoriutumaan tällaisista tehtävistä [2].

4.1 Koneoppimisen historia

Vuonna 1952 IBM:n tutkija Arthur Samuel kehitti tietokoneohjelman, joka pystyi tuottamaan ohjeistusta peräkkäisiin tammi-pelin siirtoihin. Hän keksi myös termin *koneoppiminen* ja määritteli sen tieteenalaksi, joka pystyy tarjoamaan tietokoneen kykyjä ilman ekspiliittistä koodaamista. Kehitys jatkui, ja vuonna 1967 kehitettiin KNN-algoritmi (k nearest neighbour algorithm), joka mahdollisti sen, että tietokoneet pystyvät suorittamaan yksinkertaista hahmontunnistusta (engl. *pattern recognition*). Tätä ennen oli jo alkanut *neuroverkkojen* kehitys, mutta niiden historiaa käydään läpi tarkemmin tämän tutkielman luvussa (5.1).

1960-luvun puolivälistä aina 1970-luvulle asti koneoppimisen kehitys oli erittäin hidasta ja se melkein seisoi paikallaan. Teoreettinen tutkimus ja tietokoneiden laitteistorajoitukset ajautuivat pullonkaulatilanteisiin.

Vuonna 1986 saavutettiin yksi koneoppimisen läpimurto, kun John Ross Quinlandin esitteli päätöspuun ja tarkemmin ID3 algoritmin. Seuraava suuri läpimurto koneoppmisessa oli tukivektorikoneet. Koneoppiminen jakautuikin käytännössä kahteen pääkoulukuntaan, joista toinen oli tukivektorikoneet ja toinen neuroverkot. 2000-luvun paikkeilla kernel-funktioiden yhdistäminen antoi suuren edun tukivektorikoneille ja ne suorituivat annetuista tehtävistä neuroverkkoja paremmin.

Neuroverkkojen kehitys on jatkutunut laadukkaana ja syväoppimisen avulla tulokset ovat parantuneet merkittävästi [35]. Vuoden 2024 fysiikan Nobel-palkinnon voittivat John J. Hopfield ja Geoffre Hinton työstään neuroverkkojen alalla. Tämä osoittaakin neuroverkkojen merkityksen tämän hetken koneoppimisessa.

4.2 Koneoppimisen perusteita

Koneoppiminen voidaan jakaa usealla eri tavalla, mutta yksi yleisimmistä on sen jakaminen kolmeen eri luokkaan koulutustavan mukaan. Koulutustavat ovat ohjattu oppiminen, ohjaamaton oppiminen ja vahvistusoppiminen. Ohjatulla oppimisella tarkoitetaan koneoppimismalleja, jotka tuottavat diskreetin tai jatkuvan ulostulon syötteestä, kun syöte ja ulostulo ovat valmiiksi tiedossa. Ohjaamattomalla oppimisella tarkoitetaan metodeja, joissa ulostulo ei ole valmiiksi tiedossa, vaan niiden määrittäminen on osa koneoppimismallin tehtävää. Vahvistusoppimissa käytetään hyödyksi vuorovaikutuksia, jotta opitaan käyttäytymään ympäristössä [33].

Tässä työssä esiteltävät neuroverkot ovat esimerkki ohjatusta oppimisesta. Ohjatussa oppimisessa on käytössä data D, joka muodostuu N parista d-ulottuvuuksisia havaintoja $\boldsymbol{x}_i \in \mathbb{R}^d$ ja luokista $y_i \in Y$ ja se määritellään $D = \{(\boldsymbol{x}_i, y_i)\}_{i \in 1,\dots,N}$. Ohjatun oppimisen tavoitteena on löytää funktio $f : \mathbb{R}^d \to Y$, joka kuvaa syötteen ulostuloiksi eli $y_j = f(\boldsymbol{x}_j)$, jota pystytään yleistämään D:n ulkopuolelle [33].

Ohjatun oppimisen ongelmat voidaan jakaa luokittelu- ja regressio-ongelmiksi [33]. Tässä tutkielmassa koulutetaan koneoppimismalli luokitteluun, joten sen vuoksi keskitytään luokittelun teoriaan. Luokitteluongelmassa pyritään jakamaan jokainen syöte ennalta määrättyihin kategorioihin. Esimerkiksi kuvantunnistuksessa voidaan jakaa kulkuneuvot autoihin, juniin ja lentokoneisiin [23]. Luokkien K lukumäärä on $K \leq 2$ [2].

4.3 Hold-out-validointi

Koneoppimismalli koulutetaan käyttämällä dataa. Luokitteluongelmassa malli oppii luokittelemaan datapisteet oikeisiin luokkiin. Malli, joka osaa luokitella vain tietyt datapisteet oikein, on kuitenkin usein hyödytön. Koneoppimismallin halutaan toimivan hyvin myös datapisteiden kanssa, jotka eivät ole olleet mukana koulutusvaiheessa käytetyssä datassa. Tätä varten mallin koulutusta varten oleva data voidaan jakaa koulutus- ja testidataan. Ideana on kouluttaa malli käyttäen koulutusdataa ja sen jälkeen testata sen toiminta testidatalla, jotka ovat mallille uusia datapisteitä. On hyvin yleistä käyttää 80% datasta koulutusdatana ja 20% datasta testidatana. Tämä ei kuitenkaan ole aina paras jako vaan datan suuruus vaikuttaa siihen miten data kannattaa jakaa. Esimerkiksi suurella määrällä dataa testidata voi olla alle 20% [12].

Datan jakaminen koulutusdataan ja testidataan toimii hyvin, jos verrataan kahta eri mallia toisiinsa. Molemmat mallit koulutetaan käyttämällä koulutusdataa ja testataan niiden toimivuus käyttämällä testidataa ja valitaan malleista paremmin toimiva. Koneoppimismalleissa on kuitenkin usein paljon hyperparametreja, joilla malleja voidaan säädellä. Hyvin yksinkertaisesta mallistakin saadaan lukuisia eri versioita säätämällä hyperparametreja. Jos hyperparametreja optimoidessa käytetään pelkästään jakoa koulutusdataan ja testidataan, optimoidaan malli toimimaan parhaiten kyseisellä testidatalla. Tämä voi johtaa heikkoon toimintaan, kun malli otetaan käyttöön ja se kohtaa sille uutta dataa. Tähän yleinen ratkaisu on jakaa alkuperäinen data kahden sijasta kolmeen eri osioon. Koulutusdatan ja testidatan lisäksi erotellaan koulutusdatasta vielä validointidata. Eri hyperparametrien arvoilla valitut mallit koulutetaan käyttämällä koulutusdataa ja ne arvioidaan validointidatan avulla. Malli, joka on toiminut parhaiten validointidatan kanssa, koulutetaan tämän jälkeen uudestaan käyttämällä alkuperäistä kokonaista koulutusdataa (koulutusdata + validointidata). Lopuksi malli arvioidaan vielä käyttämällä testidataa [12].

Validointidatan käyttäminen on hyvin yleinen ja toimiva metodi, kun käytössä on paljon dataa. Pienemmällä määrällä dataa kohdataan kuitenkin ongelmia. Liian pienellä testidatalla tulokset mallin toiminnasta eivät ole välttämättä luotettavia ja tuloksiin vaikuttaa testidatan valinta. Pienillä datamäärillä testidatan koon kasvattaminen riittävän suureksi johtaa liian pieniin koulutus- ja validointidatoihin, mikä aiheuttaa heikkoa toimintaa ja riippuvuutta validointidatan valintaan. Pienen datamäärän ongelma pystytään ratkaisemaan ristiinvalidoinnilla [21], joka esitellään seuraavassa luvussa.

4.4 Ristiinvalidointi

Ristiinvalidointi (engl. *cross-validation*) on toisto-otantamenetelmä, jota käytetään esimerkiksi koneoppimismallien arvioinnissa. Ristiinvalidoinnilla saadaan harhaton arvio mallin toiminnasta. Pienillä datamäärillä ristiinvalidointi tuottaa tarkempia arvioita malleista kuin hold-out-validointi. Erilaisia ristiinvalidointimenetelmiä on useita ja käydään niistä muutama yleisin läpi seuraavaksi.

4.4.1 k-kertainen-ristiinvalidointi

Ristiinvalidoinnista johdannaisessa k-kertaisessa ristiinvalidoinnissa (engl. *k-fold cross-validation*) datapisteet jaetaan k määrään ryhmiä. Koulutusprosessi iteroidaan k kertaa ja jokaisella kerralla yksi ryhmistä valitaan validointidataksi ja jäljelle jääneet ryhmät yhdistetään koulutusdataksi. Kun k:nnes iteraatiokierros on suoritettu, mallin arvioksi saadaan keskiarvo kaikkien iteraatiokerrosten kesken. Malli koulutetaan yhteensä k kertaa, joten tämä metodi on hold-out-validointiin verrattuna huomattavasti laskennallisesti raskaampi. Useampien koulutuskertojen ansiosta mallin arviointiin liittyvä varianssi on pienempi ja näin mallin arvio on luotettavampi.

Parametrin k arvo on käyttäjän päätettävissä. Yleisimpiä arvoja ovat $k = 5$ ja $k = 10$. Arvo pitää valita siten, että jokainen ryhmä edustaa hyvin koko dataa. Parametrin k arvon valitsemisessa pitää ottaa huomioon myös käytettävissä olevat laskennalliset resurssit.

4.4.2 Stratifioitu k-kertainen ristiinvalidointi

Stratifioitu k-kertainen ristiinvalidointi (engl. *stratified k-fold cross-validation*) ottaa huomioon datan epätasaisen jakauman luokkien suhteen, joka on hyvin yleistä koneoppimismallia koulutettaessa. Stratifioidussa k-kertaisessa ristiinvalidoinnissa ryhmien jaossa pidetään huolta siitä, että niiden jakauma luokkien suhteen vastaa koko datan jakaumaa luokkien suhteen. Näin vältetään tilanne, jossa jonkun luokan edustajia ei olisi ollenkaan tai niitä olisi hyvin vähän jossain ryhmistä, mikä vaikuttaisi kyseisellä iteraatiolla mallin arviointiin.

4.4.3 "Leave-one-out" -ristiinvalidointi

"Leave-one-out" -ristiinvalidointi on k-kertainen ristiinvalidointi, jossa $k = n$ (n on datapisteiden määrä koko datassa). Tällöin jokaisella iteraatiolla koulutusdata koostuu kaikista paitsi yhdestä datapisteestä ja validointidata on tällöin vain yksi datapiste. Malli koulutetaan n kertaa, joten tämä metodi on laskennallisesti hyvin raskas. "Leave-one-out" -ristiinvalidointia suositellaan käytettäväksi pienillä tai luokkien suhteen epätasaisesti jakautuneilla datoilla [21].

4.5 Luokittelumallin arviointi

Tämä luku on kirjoitettu kirjan [17] ja artikkelin [10] pohjalta. Ohjatun oppimisen luokitteluongelmassa datapisteiden oikeat luokat ovat tiedossa. Binääriseen luokitteluongelmaan, eli ongelmaan, jossa luokkien määrä on $K = 2$, koulutetun mallin hyvyyttä testattaessa, on mallin luokittelutuloksella neljä mahdollista vaihtoehtoa.

Vaihtoehdot ovat tosi positiivinen (engl. *true positive*), tosi negatiivinen (engl. *true negative*), väärä positiivinen (engl. *false positive*) ja väärä negatiivinen (engl. *false negative*). Näiden määritelmät esitetään seuraavaksi.

1. Tosi positiivinen: Datapisteen todellinen luokka on positiivinen ja malli luokittelee sen positiiviseksi. Määrästä käytetään merkintää N_{TP}.

2. Tosi negatiivinen: Datapisteen todellinen luokka on negatiivinen ja malli luokittelee sen negatiiviseksi. Määrästä käytetään merkintää N_{TN}.

3. Väärä positiivinen: Datapisteen todellinen luokka on negatiivinen ja malli luokittelee sen positiiviseksi. Määrästä käytetään merkintää N_{FP}.

4. Väärä negatiivinen: Datapisteen todellinen luokka on positiivinen ja malli luokittelee sen negatiiviseksi. Määrästä käytetään merkintää N_{FN}.

Luokittelutuloksista voidaan koota $2x2$-dimensiota oleva sekaannusmatriisi. Sekaannusmatriisi on esitetty seuraavassa taulukossa.

	Positiiviseksi ennustettu	**Negatiiviseksi ennustettu**
Todellinen positiivinen	Tosi positiivinen (N_{TP})	Väärä negatiivinen (N_{FN})
Todellinen negatiivinen	Väärä positiivinen (N_{FP})	Tosi negatiivinen (N_{TN})

Taulukko 1: Sekaannusmatriisi

Oikein luokiteltujen lukumäärä on $N_{TP} + N_{TN}$ ja väärien luokittelujen lukumäärä on $N_{FP} + N_{FN}$.

4.5.1 Yleisimmät metriikat

Luokittelumallin arviointia varten on kehitelty erilaisia metriikoita. Käydään seuraavaksi läpi yleisimpiä arviointimetriikoita.

Määritelmä 1. Luokittelumallin *virheaste* (engl. *error rate*), jota merkitään E, on mallin tekemien virheiden määrä suhteessa kaikkiin datapisteisiin

$$E = \frac{N_{FP} + N_{FN}}{N_{FP} + N_{FN} + N_{TP} + N_{TN}}. \tag{1}$$

Virheasteen vastakohta on mallin tekemien oikeiden luokittelujen määrä suhteessa kaikkiin datapisteisiin. Virheasteen vastakohtaa kutsutaan tarkkuudeksi (engl. *accuracy*). Tarkkuus on $1 - E$ ja merkataan sitä *Acc*. *Acc* esitettynä kaavan avulla on

$$Acc = \frac{N_{TP} + N_{TN}}{N_{FP} + N_{FN} + N_{TP} + N_{TN}}. \tag{2}$$

Virheaste ja tarkkuus toimivat hyvin, kun data on tasaisesti jakautunut positiivisten ja negatiivisten havaintojen suhteen. Datan ollessa epätasaisesti jakautunut tässä suhteessa, virheaste ja tarkkuus eivät ole enää hyviä arvioita. Otetaan esimerkiksi data, jossa 98% havainnoista on negatiivisia. Tällaiseen dataan koulutettu luokittelumalli, joka ennustaa kaikki arvot negatiiviseksi on oikeassa 98% kerroista. Tällainen malli voidaan kuitenkin todeta täysin hyödyttömäksi hyvästä tarkkuudesta huolimatta. Datoissa on usein luokkien epätasapainoa, joten tarvitaan arviointimetodeja, jotka toimivat hyvin epätasapainoisissa datoissa. Täsmällisyys (engl. *precision*) ja herkkyys (engl. *recall*) ovat arvioita, jotka ottavat huomioon vain toisen luokista. Käyttämällä datassa harvinaisempaa luokkaa saadaan luokittelumallista toimiva arvio. Oletetaan, että datassa on huomattavasti vähemmän positiiviseen luokkaan kuuluvia datapisteitä. Määritetään seuraavaksi käsitteet täsmällisyys ja herkkyys.

Määritelmä 2. *Täsmällisyys* on tosi positiivisten suhde kaikkiin datan positiivisen luokan pisteisiin. Käytetään täsmällisyydestä merkintää Pr. Pr lasketaan kaavalla

$$Pr = \frac{N_{TP}}{N_{TP} + N_{FP}}. \tag{3}$$

Täsmällisyys voidaan ajatella mallin oikeassa olemisen todennäköisyytenä, kun se luokittelee datapisteen positiiviseksi. *Herkkyydellä* tarkoitetaan todennäköisyyttä, että positiivinen datapiste tulee luokitelluksi positiiviseksi. Herkkyys on tosi positiivisten suhde datan kaikkiin positiivisiin datapisteisiin ja sitä merkataan Re. Herkkyyttä voidaan kutsua myös tosi positiivisten suhteeksi, jolloin siitä käytetään merkintää tp rate, ja se lasketaan kaavalla

$$Re = \frac{N_{TP}}{N_{TP} + N_{FN}}. \tag{4}$$

Esimerkki 1. Luokittelumalli on koulutettu binääriseen luokitteluongelmaan käyttäen dataa, jossa on 1000 datapistettä, joista 70 on positiivisia. Seuraavassa taulukossa on esitetty datan jakauma positiivisten ja negatiivisten luokkien välillä sekä mallin tulokset testidatalle. Tästä voidaan laskea tarkkuus, täsmällisyys ja herkkyys:

	Mallin antamat luokat:	
Todelliset luokat:	**Positiiviset**	**Negatiiviset**
Positiiviset	20	50
Negatiiviset	30	900

$$Acc = \frac{20 + 900}{1000} = 0.92; \quad Pr = \frac{20}{20 + 30} = 0.40; \quad Re = \frac{20}{20 + 50} = 0.29. \tag{5}$$

Esimerkissä (1) huomataan epätasaisesti jakautuneen datan vaikutus hyvyyden mittareihin. Mallin tarkkuus on hyvää luokkaa, sillä se on yli 90%, mutta täsmällisyys ja herkkyys kuitenkin paljastavat, että malli toimii heikosti. 40% täsmällisyys

tarkoittaa heikkoa laatua. Malli luokittelee positiiviseksi 50 näytettä, joista positiivisia on oikeasti 20 ja 30 vääriä positiivisia. Herkkyyden kohdalla tilanne on vielä huonompi. Testidatassa malli saa luokiteltua oikein vain 20 yhteensä 70:stä positiivisesta näytteestä.

Luokitteluongelman luonne sattaa vaikuttaa siihen, kumpaa arvostetaan enemmän, tarkkuutta vai herkkyyttä. Esimerkiksi verkkokaupat, joissa tuotetta katsellessa, verkkokauppa tarjoaa myös muita tuotteita, joita muut kyseisen tuotteen ostajat ovat ostaneet. Tällaisen tarjouslistan tekemisessä on saatettu käyttää koneoppimismallia kaupan ostohistoriadataa hyödyntäen. Listaan halutaan varmasti sellaisia tuotteita, jotka kiinnostavat asiakasta; muuten asiakas saattaa jättää huomioimatta listan eikä jatkossakaan kiinnitä siihen huomiota. Täsmällisyys perustuu siihen, että kaikki positiiviseksi luokitellut ovat varmasti positiivisia, eli juuri siihen mitä verkkokauppa haluaa. Herkkyys puolestaan kertoo enemmän siitä, tuleeko kaikki positiiviset luokitelluksi oikein. Verkkokaupan tilanteessa se ei kuitenkaan ole niin tärkeää, sillä lista tarjottavista tuotteista on yleensä melko lyhyt eikä kaikkia positiivisia tuotteita ole tarkoitus saada listaan mukaan.

Lääketieteellisissä ongelmissa herkkyyttä pidetään usein vastaavasti tärkeämpänä. Kaikille sairaudesta X kärsiville potilaille halutaan saada positiivinen diagnoosi. Tällöin halutaan siis minimoida väärien negatiivisten määrää. Herkkyyden määritelmästä (4) nähdään että pieni määrä vääriä negatiivisia tarkoittaa korkeaa herkkyyden arvoa.

Täsmällisyys ja herkkyys voidaan yhdistää yhdeksi metriikaksi tuloksen yksinkertaistamiseksi.

Määritelmä 3. F_β on yhdistelmä täsmällisyydestä ,ja herkkyydestä. Parametrin β arvoa vaihtelemalla voidaan säädellä, kumpaa näistä painotetaan. Kun $\beta > 1$, painotetaan enemmän herkkyyttä. F_β lasketaan

$$F_\beta = \frac{(\beta^2 + 1) \times Pr \times Re}{\beta^2 \times Pr + Re}. \tag{6}$$

Jos ei tiedetä, kumpaa metriikoista halutaan painottaa, voidaan parametri β asettaa nollaksi. Näin saadaan yleisesti käytetty F_1-arvo, joka on nyt muotoa

$$F_1 = \frac{2 \times Pr \times Re}{Pr \times Re} \tag{7}$$

4.5.2 Roc-käyrä ja auc

Roc-käyrä (engl. *receiving operating characteristic graph*) on visuaalinen keino mallin hyvyyden esittämiseksi. Käyrä piirretään asteikolle, jossa y-akselilla on herkkyys ja x-akselilla on väärien positiivisten suhde. Määritellään seuraavaksi väärien positiivisten suhde.

Määritelmä 4. Merkitään väärien positiivisten suhdetta fp rate ja lasketaan se

$$\text{fp rate} = \frac{N_{FP}}{N_{TN} + N_{FP}}. \tag{8}$$

Diskreettinen luokittelija on luokittelija, jonka ulostulona on pelkästään luokka, johon malli luokittelee datapisteen. Diskreetit luokittelijat voidaan sijoittaa roc-avaruuteen pisteinä. Täydellisesti toimiva luokittelija sijaitsee pisteessä (0,1). Täydelliselle luokittelijalle pätee tp rate $= 1,0$ ja fp rate $= 0,0$. Täysin satunnaisesti toimiva luokittelija sijoittuu diagonaalille $y = x$. Esimerkiksi luokittelija, joka arvaa puolet datapisteistä positiiviseen luokkaan, voidaan olettaa saavan puolet positiivisista ja puolet negatiivisista oikein. Tämä johtaa arvoihin tp rate $= 0,5$ ja fp rate $= 0,5$ ja malli sijoittuu pisteeseen $(0,5; 0,5)$. Kaikki mallit, jotka sijoittuvat diagonaalin alapuolelle ovat siis huonompia luokittelemisessa kuin satunnainen arvaus.

Osa luokittelumalleista, kuten neuroverkot, tuottavat ulostulona todennäköisyyden, jolla datapiste kuuluu luokkaan. Tällaisesta mallista saadaan diskreettinen luokittelija käyttämällä apuna kynnysarvoa. Mikäli ulostulo on suurempaa kuin kynnysarvo, luokitellaan datapiste positiiviseksi, muissa tapauksissa se luokitellaan negatiiviseksi. Kaikki eri kynnysarvot tuottavat eri pisteen roc-avaruuteen. Numeerisella määrällä kynnysarvoja saatu roc-käyrä on porrasfunktio, joka lähestyy todellista käyrää, kun kynnysfunktioiden määrä lähestyy ääretöntä. Aikaisemmin esitellyn, satunnaisesti puolet datapisteistä positiiviseksi luokittelevan mallin roc-käyrä on suora $y = x$.

Mallien vertailu roc-käyrien perusteella tehdään laskemalla roc-käyrän ja x-akselin väliin jäävän alueen pinta-ala. Tästä pinta-alasta käytetään lyhennettä auc (area under curve). Auc on osa yksikköneliön pinta-alasta, joten sen suuruus on aina välillä $(0, 1)$. Satunnaisen luokittelijan auc on $0,5$, joten minkään todellisen mallin auc-arvon ei tulisi olla alle tämän arvon. Käytetään jatkossa roc-käyrän auc-arvosta merkintää auroc (area under roc curve).

5 Neuroverkot

Neuroverkot ovat suosittuja koneoppimismalleja. Ne koostuvat eri kerroksista, joita ovat syötekerros, piilokerros ja ulostulokerros. Nämä kerrokset koostuvat yksinkertaisista laskennallisista yksiköistä, joita nimitetään neuroneiksi. Neuronit ovat yhdistetty toisiinsa painoilla. Jokaisen neuronin syötteeseen on liitetty paino, joka vaikuttaa yksikössä tapahtuvaan laskentaan. Neuroverkkojen oppiminen tapahtuu muokkaamalla näitä painoja [1].

5.1 Syväoppimisen historia

Syväoppiminen oli pitkään hyvin epäsuosittua ja sitä on historiansa aikana kutsuttu monilla eri nimillä, minkä takia syväoppiminen voikin vaikuttaa uudelta tieteenalalta. Syväoppimisen historia alkoi kuitenkin jo 1940-luvulta ja vuodesta 2006 sitä on kutsuttu syväoppimiseksi. Vuosina 1940-1960 syväoppiminen tunnettiin kybernetiikkana (engl. *cybernetics*) ja vuosina 1980-1990 konnektionismina (engl. *connectionism*). Aikaisimmat algoritmit luotiin muistuttamaan biologista oppimista, eli sitä miten aivot oppivat, tai miten niiden oletettiin oppivan, ja tästä tulikin nimitys neuroverkot. Vaikka neuroverkkoja on käytetty ymmärtämään aivojen toimintaa, eivät ne ole realistisia malleja biologisista funktioista. Neurologiaa pidetään tärkeänä inspiraation lähteenä syväoppimisen tutkimiselle, mutta enää sen vaikutus syväoppimisen kehityksessä ei ole niin merkittävä. Merkittävimpänä tekijänä tähän on se, että aivojen toiminnasta ei tiedetä tarpeeksi, jotta se voisi toimia ohjaajana syväoppimisen kehityksessä. Tämän luvun sisältö on peräisin kirjasta [13].

5.1.1 Kybernetiikka

Modernin syväoppimisen ensimmäinen aalto oli kybernetiikka, jossa kehitettiin yksinkertaisia lineaarisia malleja neurologian innoittamana. Nämä mallit olivat luotu yhdistämään joukko n syötearvoja $x_1, \ldots, x_n$ ulostuloon y. Mallit oppivat painokertoimet $w_1, \ldots, w_n$ ja laskivat ulostulon $f(\boldsymbol{x}, \boldsymbol{w}) = x_1 w_1 + \ldots + x_n w_n$.

Varhainen lineaarinen mallinnus arivojen toiminnasta oli McCulloh-Pittsin neuroni (McCulloch ja Pitts, 1943), jolla pystyttiin luokittelemaan syötteet kahteen luokkaan, sen perusteella oliko $f(\boldsymbol{x}, \boldsymbol{w})$ positiivinen vai negatiivinen. Jotta malli toimi oikein piti painokertoimet säätää halutun tuloksen mukaan. Painokertoimet vaativat ihmiskäyttäjän asettamisen. 1950-luvulla Rosenblattin luomasta perseptronista tuli ensimmäinen malli, joka pystyi oppimaan luokat määrittelevät painokertoimet saatuaan esimerkkidatapisteen kaikista luokista. Perseptronin rakenne ja toiminta esitellään tarkemmin luvussa (5.2).

Lineaaristen mallien tunnetuimpana heikkoutena on niiden kyvyttömyys oppia XOR-funktio, jonka totuustaulu esitetään taulukossa (2). Tämä aiheutti kritiikkiä biologisesti inspiroitunutta oppimista kohtaan ja aiheutti lopulta ensimmäisen ison kiinnostuksen ja suosion vähenemisen neuroverkkojen ja syväoppimisen historiassa.

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Taulukko 2: XOR-funktion totuustaulu

5.1.2 Konnektionismi

Syväoppimisen toinen aalto saapui 1980-luvulla konnektionismin muodossa. Konnektionismi kehittyi kognitiotieteen kontekstissa. Konnektionismin keskeisimpänä ideana on yksittäisten yksinkertaisten laskennallisten yksiköiden mahdollisuus älykkääseen toimintaan niitä yhdistettäessä yhdeksi yksiköksi. Tämä pätee sekä biologisiin neuroneihin, että piiloyksiköihin laskennallisissa malleissa. Konnektionismin yhteydessä kehitettiin konsepteja, jotka ovat edelleen käytössä modernissa syväoppimisessa. Yksi tällainen konsepti on vastavirta-algoritmin käyttäminen syviä neuroverkkoja koulutettaessa. Vastavirta-algoritmi esitellään tarkemmin luvussa (5.3).

Toinen aalto kesti 1990-luvun puoliväliin asti. Tuohon aikaan tekoälyyn sijoittamisesta kiinnostuneet tahot vaativat liikaa aikansa neuroverkoilta ja joutuivat pettymään niiden suorituskykyyn. Tämä ja muiden koneoppismallien kehitykset johtivat kiinnostuksen vähenemiseen neuroverkkoja kohtaan. Tätä jatkui aina 2000-luvun puoliväliin asti.

5.1.3 Kolmas aalto

Syväoppimisen kolmas aalto alkoi läpimurrolla vuonna 2006. Geoffrey Hinton osoitti, että DB-neuroverkkoja (engl. *deep belief network*) pystyttiin kouluttamaan käyttämällä hyödyksi ahnetta kerroskohtaista esikoulutusta (engl. *greedy layer-wise pretraining*). Tämä pystyttiin pian yleistämään myös muiden syvien neuroverkkojen kouluttamiseen. Hintonin tutkimus johti termin "syväoppiminen"syntymiseen. Nyt pystyttiin kouluttamaan syviä neuroverkkoja ja tutkimus alkoikin keskittyä neuroverkkon syvyyteen.

5.2 Neuroni

Luvussa (5.1.1) mainittiin perseptroni, joka on kaikista yksinkertaisin neuroverkkomalli [1]. Perseptronissa tapahtuva laskenta on lineaarista ja perseptronin ulostulo on binääristä [27]. Perseptroni toimii pohjana neuroverkoille ja laajempaa neuroverkkoa voidaankin kutsua monikerroksiseksi perseptroniksi (engl. *multilayer perceptron*), josta käytetään lyhennettä MLP. Tässä luvussa esitellään perseptronin rakenne, toiminta ja kouluttaminen. Tutkielman luvussa (5.3) esitellään rakenteeltaan monimutkaisempia monikerroksisia perseptroneita. Tämä luku on kirjoitettu lähteiden [2] [1] [27] [29] pohjalta.

5.2.1 Neuronin rakenne ja toiminta

Yksinkertaisuudessaan perseptroni koostuu syötteestä $x_j \in \mathbb{R}$, jossa $j = 1, \ldots, d$, painokertoimista $w_j \in \mathbb{R}$ ja binäärisestä ulostulosta y. Perseptronissa laskentaa tapahtuu vain yhdessä kerroksessa, joten sitä voidaan pitää yksikerroksisena neuroverkkona. Painokertoimet painottavat syötteen tärkeyttä päätöksenteossa ja ulostulo on syötteen painotettu summa $\sum_j w_j x_j$. Päätöksentekoa varten täytyy määritellä kynnysarvo ulostulolle. Ulostulo on 0, jos summa on alle määritetyn kynnysarvon ja ulostulo on 1, jos se ylittää kynnysarvon. Perseptronin toiminta esitettynä yhtälömuodossa on

$$\text{ulostulo} = \begin{cases} 0 & \text{jos} \sum_j w_j x_j \leq \text{kynnysarvo} \\ 1 & \text{jos} \sum_j w_j x_j > \text{kynnysarvo} \end{cases}. \tag{9}$$

Esitystä (9) voidaan parantaa käyttämällä ulostulosta merkintää y ja korvaamalla kynnysarvo vakiotermillä (engl. *bias*) b, jolle pätee $b = -\text{kynnysarvo}$. Näin yhtälömuodoksi saadaan

$$\text{y} = \begin{cases} 0 & \text{jos} \sum_j w_j x_j + b \leq 0 \\ 1 & \text{jos} \sum_j w_j x_j + b > 0 \end{cases}. \tag{10}$$

Vakiotermin voidaan ajatella myös olevan painokerroin w_0 ylimääräiselle vakiosyötteelle x_0, jonka arvo on aina 1. Tällöin perseptronin ulostulo painotettuna summana on

$$y = \sum_{j=1}^{d} w_j x_j + w_0 \tag{11}$$

ja vastaavasti pistetulona

$$y = \boldsymbol{w}^T \boldsymbol{x}, \tag{12}$$

missä $\boldsymbol{w} = [w_0, w_1, \ldots, w_d]^T$ ja $\boldsymbol{x} = [1, x_1, \ldots, x_d]^T$.

Perseptronin päätöksenteossa käytetään apuna kynnysarvoa, johon painotettua summaa verrataan. Tämä vastaa painotetun summan syöttämistä porrasfunktioon. Porrasfunktiota perseptronin yhteydessä kutsutaan sen aktivaatiofunktioksi. Aktivaatiofunktiota vaihtamalla päästään eroon vain binäärisestä ulostulosta. Sigmoid-funktio on yleisesti käytetty vaihtoehto porrasfunktiolle. Perseptronia, jonka aktivaatiofunktio on vaihdettu porrasfunktiosta sigmoid-funktioksi, kutsutaan sigmoid-neuroniksi tai yleisesti neuroniksi. Yleisimpiä aktivaatiofunktioita esitellään lisää luvussa (5.4.1). Sigmoid-funktiosta käytetään merkintää σ ja se määritellään

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{13}$$

Neuronin, jonka aktivaatiofunktio on sigmoid-funktio, ulostulo voidaan esittää muodossa

$$y = \frac{1}{1 + e^{-\boldsymbol{w}^T \boldsymbol{x}}}. \tag{14}$$

Sigmoid-funktion arvojoukko on avoin väli $(0, 1)$. Tällöin myös sigmoid-neuronin ulostulo on välillä $(0, 1)$ ja sitä voidaan pitää todennäköisyytenä. Binäärisessä luokittelussa, jossa on käytössä luokat 0 ja 1, neuronin ulostulo on havaintopisteen todennäköisyys kuulua luokkaan 1. Luokkaan 0 kuulumisen todennäköisyys on tällöin vastatodennäköisyys $100 - y$.

Luokkien lukumäärän ollessa $K > 2$ tarvitaan neuroneita rinnakkain K kappaletta. Jokaisella neuronilla on oma painovektori w_i ja jokainen neuroni laskee painotetun summan seuraavasti

$$y_i = \sum_{j_1}^{d} w_{ij} x_j + w_{i0} = \boldsymbol{w}_i^T \boldsymbol{x}, \tag{15}$$

missä w_{ij} on painokerroin syötteestä x_j ulostulolle y_i. Ulostulo voidaan esittää vektorina

$$\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}, \tag{16}$$

missä $\boldsymbol{W}$ on $K \times (d+1)$ painokerroinmatriisi, jonka riveinä ovat neuronien painokertoimet. Sarakkeiden määrä on $d+1$, joka koostuu syötteiden määrästä d ja vakiotermistä 1. Useamman luokan luokittelussa on tavallista käyttää aktivaatiofunktiona softmax-funktiota

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}, \forall i \in \{1, \ldots, k\}. \tag{17}$$

Softmax-funktio neuronien ulostuloarvoilla $\boldsymbol{y}$ antaa havaintopisteen todennäköisyyden kuulua luokkaan i. Saadut todennäköisyydet kaikkien luokkien välillä summautuvat luvuksi 1, jonka takia softmax-funktio on hyvin yleisesti käytetty aktivaatiofunktio useamman kuin kahden luokan luokittelussa.

5.2.2 Neuronin kouluttaminen

Neuronin kouluttaminen tapahtuu painokertoimia päivittämällä. On olemassa erilaisia oppimismetodeita, jotka perustuvat siihen, kuinka usein painokertoimia päivitetään. Painokertoimet voidaan päivittää jokaisen syötearvon jälkeen. Tätä oppimismetodia kutsutaan reaaliaikaiseksi oppimiseksi (engl. *online learning*). Toinen yleinen oppimismetodi on eräoppiminen (engl. *batch learning*), jossa painokertoimet päivitetään, jokaisen epookin (engl. *epoch*) jälkeen. Kaikkien syötearvojen läpikäymistä yhden kerran kutsutaan epookiksi. Näiden kahden oppimismetodin välimuotoa kutsutaan pieneräoppimiseksi (engl. *minibatch learning*), jossa painokertoimet päivitetään pienemmän satunnaisesti valikoidun erän jälkeen. Pieneräoppiminen on näistä metodeista nykyään yleisin.

Painokertoimet voidaan alustaa satunnaisilla arvoilla, ja tällöin myös neuronin tekemä ennuste, $\hat{y}$, on satunnainen, jolloin on epätodennäköistä, että se vastaa havaittua arvoa y. Painokertoimien päivittäminen perustuu ennusteen ja havaitun arvon erotuksen $y - \hat{y}$ minimointiin. Erotuksen arviointiin käytettävää funktiota kutsutaan tappiofunktioksi (engl. *loss function*). Seuraavassa määritelmässä esitetään yleisesti käytetyt tappiofunktiot regressio- ja luokitteluongelmissa.

Määritelmä 5. Käytetään yksittäisestä havaintopisteestä x merkintää (x,y), missä y on havaintopisteen todellinen arvo. Regressio-ongelmassa käytetään yleisesti tappiofunktiota

$$E(\boldsymbol{w}|x, y) = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}[y - (\boldsymbol{w}^T x)]^2. \tag{18}$$

Luokitteluongelmassa tappiofunktiona käytetään ristientropiaa (engl. *cross-entropy*), joka on muotoa

$$E(\boldsymbol{w}|x, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}). \tag{19}$$

Tappiofunktioista (18) (19) saadaan painokertoimille päivityssäännöt hyödyntämällä gradienttimenetelmää. Yksittäisen havaintopisteen tapauksessa tätä kutsutaan stokastikseksi gradienttimenetelmäksi. Gradienttimenetelmässä mimoidaan kohdefunktiota ottamalla askeleita gradientin vastaiseen suuntaan. Askelten pituudesta käytetään merkintään η, jota kutsutaan myös oppimistekijäksi (engl. *learning factor*). Tappiofunktion gradienttivektori koostuu jokaisen painokertoimen suhteen lasketusta osittaisderivaatasta, ja sen matemaattinen esitys on muotoa

$$\nabla_w E = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_d} \right]^T, \tag{20}$$

josta saadaan päivityssääntö

$$w_i = w_i + \Delta w_i = w_i - \eta \frac{\partial E}{\partial w_i}, \forall i. \tag{21}$$

Tarkastellaan seuraavaksi ristientropiaa. Gradienttivektorin laskemisessa ristientropialle käytetään ketjusääntöä. Sovelletaan ketjusääntöä kahdesti, jolloin saadaan

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w}, \tag{22}$$

missä $\hat{y} = \sigma(z)$ ja $z = \boldsymbol{w}^T \boldsymbol{x}$. Seuraavaksi lasketaan ketjusäännön avulla saadut osittaisderivaatat

$$\frac{\partial E}{\partial \hat{y}} = \frac{-y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}} \tag{23}$$

$$\frac{\partial \hat{y}}{\partial z} = \hat{y}(1 - \hat{y}) \tag{24}$$

$$\frac{\partial z}{\partial w} = x. \tag{25}$$

Osittaisderivaatta (24) on saatu hyvin yksinkertaiseen muotoon, joka saadaan sigmoidfunktion derivaatasta seuraavasti

$$\frac{\partial \hat{y}}{\partial z} = \frac{e^{-z}}{(1 + e^{-z})^2} = \frac{1}{1 + e^{-z}} \cdot \frac{e^{-z}}{1 + e^{-z}} = \hat{y}(1 - \hat{y}). \tag{26}$$

Kaikki ketjusäännön osat kerrotaan yhteen ja saadaan

$$\frac{\partial E}{\partial w} = \left(\frac{-y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}} \right) \cdot \hat{y}(1 - \hat{y}) \cdot x = (\hat{y} - y)x. \tag{27}$$

Nyt aikaisemmin esitetty päivityssääntö saadaan muotoon

$$w_i = w_i + \Delta w_i = w_i - \eta(y - \hat{y})x_i. \tag{28}$$

Triviaalissa tilanteessa ennuste ja havaittu arvo ovat yhtä suuret ja päivitystä ei tapahdu. Kun päivitys tapahtuu, riippuu sen suuruus ennusteen ja havaitun arvon erotuksen suuruudesta, syötteen arvosta ja oppimistekijän suuruudesta. Suuri oppimistekijä painottaa viimeisimpiä havaintopisteitä ja malli käyttäytyy, kuin sillä olisi lyhyt muisti. Liian pienellä oppimistekijän arvolla koulutus hidastuu ja vaadittavan laskentatehon määrä kasvaa. Sopivan oppimistekijän löytäminen on tärkeää ja luvussa (5.4.2) esitellään adaptiivinen oppimistekijä, joka on kehitetty tätä ongelmaa varten.

5.3 Monikerroksiset perseptroniverkot

Edellisessä luvussa esitellyt neuronit koostuivat vain syötteestä, ulostulosta ja niiden välillä olevista painokertoimista. Laskentaa perseptronissa tapahtuu vain ulostulokerroksessa. Syötekerros toimittaa datan ulostulokerrokselle, jossa kaikki laskenta tapahtuu käyttäjän nähtävissä. Yksinkertaiset perseptronit eivät kuitenkaan opi epälineaarista dataa. Jotta voisimme luokitella epälineaarista dataa, pitää perseptroneja kasata päällekkäin. Syötteen ja ulostulon väliin voidaan lisätä kerroksia. Kerroksia kutsutaan piilokerroksiksi, koska käyttäjä ei näe niissä tapahtuvaa laskentaa. Piilokerroksissa on useita neuroneita rinnakkain myös binäärisessä luokittelussa. Jokainen kerros siirtää dataa eteenpäin, ja data kulkee koko neuroverkon läpi samansuuntaisesti. Tällaisia neuroverkkomalleja kutsutaan eteenpäinsyöttäviksi verkoiksi (engl. *feed forward network*). Eteenpäinsyöttävää neuroverkkoarkkitehtuuria, jossa on käytössä vähintään yksi piilokerros, voidaan kutsua monikerroksiseksi perseptroniverkoksi. Monikerroksiset perseptroniverkot pystyvät luokittelemaan epälineaarista dataa. Jos piilokerroksia on useita, kutsutaan mallia syväksi neuroverkoksi. Aloitetaan kuitenkin käymällä läpi tapausta, jossa monikerroksinen perseptroniverkko koostuu vain syötteestä, yhdestä piilokerroksesta ja tulosteesta [2] [1].

5.3.1 Monikerroksisen perseptroniverkon rakenne

Yksinkertainen monikerroksinen perseptroniverkko koostuu siis syötteestä, piilokerroksesta ja tulosteesta [2]. Eteenpäinsyöttävässä verkossa kerroksen kaikki neuronit ovat yhteydessä kaikkiin seuraavan kerroksen neuroneihin [1]. Syötekerros toimittaa syötteen $\boldsymbol{x}$ piilokerrokselle, joka koostuu useista rinnakkaisista neuroneista. Piilokerroksen neuronit laskevat painotetut summat syötteelleen, kuten yksittäisen neuronin tapauksessa. Painotetut summat syötetään aktivaatiofunktioon, joka tässä tapauksessa on sigmoid-funktio. Näin saadaan

$$z_h = sig(\boldsymbol{w}_h^T \boldsymbol{x}) = \frac{1}{1 + \exp\left[-\left(\sum_{j=1}^{d} w_{hj} x_j + w_{h0}\right)\right]}, h = 1, \ldots, H. \tag{29}$$

Ulostulokerros on myös kerros perseptroneja, jotka ottavat syötteekseen piilokerroksen arvot z_h ja tuottavat ulostulon y_i:

$$y_i = \boldsymbol{v}_i^T \boldsymbol{z} = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}, \tag{30}$$

missä z_0 ja v_{i0} ovat vakiotermi ja -paino. Vakiotermin z_0 arvo on tässäkin tapauksessa 1. Kuten aikaisemmin yksinkertaisessa perseptronissa syötekerroksessa ei tapahdu laskentaa, eikä sitä myöskään lasketa kerrokseksi, kun määritellään neuroverkon syvyyttä. Näin ollen tämä neuroverkko on kaksikerroksinen. Kahden luokan luokitteluongelmassa ulostulon aktivaatiofunktiona on sigmoid-funktio ja ulostulokerros koostuu yhdestä neuronista. Kuten aikaisemmin todettiin, sigmoid-funktion ulostulot ovat välillä $(0, 1)$, ja ne edustavat todennäköisyyksiä kuulua luokkiin. Luokkien määrän K ollessa $K > 2$, neuroneiden määrä ulostulokerroksessa on myös K, ja silloin käytetään ulostulon aktivaatiofunktiona softmax-funktiota. Nämä aktivaatiofunktiot ovat epälineaarisia ja takaavat näin myös, että malli kykenee suorittamaan epälineaarista jaottelua. Jos neuroverkossa kaikki funktiot ovat lineaarisia, myös lopputulos on lineaarinen, sillä lineaarikombinaation lineaarikombinaatio on edelleen lineaarikombinaatio [2].

5.3.2 Monikerroksisen perseptroniverkon kouluttaminen

Monikerroksisen perseptroniverkon kouluttaminen on käytännössä hyvin samanlaista kuin yksittäisen perseptronin kouluttaminen. Isoimpana erona näiden välillä on se, että monikerroksisissa perseptroniverkoissa ulostulo on epälineaarista piilokerroksen epälineaaristen aktivaatiofunktioden ansiosta. Piilokerros toimii ulostulokerroksen syötteenä, ja näin ulostuloa voidaan pitää neuronina ja sen painojen päivitykseen käytetään aikaisemmin esitettyä päivityssääntöä (28). Piilokerroksen päivittämiselle ei ole suoraa tapaa, koska niiden laskemia arvoja ei voida verrata suoraan mihinkään tunnettuihin arvoihin eikä näin voida soveltaa tappiofunktiota. Päivitykset voidaan kuitenkin laskea takautuvasti ulostulon tappiofunktiota hyödyntäen. Gradientin laskemisessa hyödynnetään tässäkin ketjusääntöä ja saadaan

$$\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial z_h}\frac{\partial z_h}{\partial w_{hj}}. \tag{31}$$

Nämä osittaisderivaata ovat

$$\frac{\partial E}{\partial \hat{y}_i} = -(y - \hat{y}) \tag{32}$$

$$\frac{\partial \hat{y}}{\partial z_h} = v_h \tag{33}$$

$$\frac{\partial z_h}{\partial w_{hj}} = z_h(1 - z_h)x_j. \tag{34}$$

Päivityssäännöiksi saadaan nyt

$$\Delta v_h = \eta(y - \hat{y})z_h \tag{35}$$

$$\Delta w_{hj} = \eta(y - \hat{y})v_h z_h(1 - z_h)x_j, \tag{36}$$

missä v_h ovat ulostulokerroksen painokertoimet ja w_{hj} ovat piilokerroksen painokertoimet [2].

5.4 Syvät neuroverkot

Neuroverkkojen arkkitehtuuria voidaan kasvattaa, joko lisäämällä piilokerroksia tai piilokerroksissa olevien neuroneiden määrää. Vähemmän rinnakkaisia neuroneita ja enemmän piilokerroksia sisältävä neuroverkko on rakenteeltaan pitkä, mutta ohut. Empiirinen fakta on, että pitkät ja ohuet neuroverkot sisältävät vähemmän parametreja ja suoriutuvat paremmin tuntemattoman datan kanssa. Tämän takia syvät neuroverkot ovat rakenteeltaan pitkiä ja ohuita. Syvien neuroverkkojen kanssa kohdataan myös mahdollisia ongelmia. Syvät neuroverkot ovat huomattavasti suurempia, ja niissä on enemmän painokertoimia ja prosessoivia yksiköitä. Niiden kouluttaminen vaatii enemmän laskutehoa ja muistia. Teknologian kehityksen myötä näiden resurssien hinta on laskenut ja saatavuus parantunut. Piilokerrosten suurempi lukumäärä aiheuttaa ongelmia myös vastavirta-algoritmin laskemisessa. Laskettava gradientti saattaa kasvaa liian suureksi tai kadota käytännössä kokonaan [2]. Alaluvussa (5.4.1) esitellään lisää aktivaatiofunktioita ja alaluvussa (5.4.2) esitellään syvien neuroverkkojen laskennan helpottamiseksi kehitettyjä metodeita.

5.4.1 Aktivaatiofunktiot

Tässä luvussa esitellään vaihtoehtoisia aktivaatiofunktioita aikaisemmin esitellylle sigmoid-funktiolle. Kuten tässä tutkielmassa aikaisemmin mainittiin epälineaariset aktivaatiofunktiot mahdollistavat epälineaarisen datan oppimisen. Tämän lisäksi aktivaatiofunktiot eivät saa kasvattaa neuroverkon laskennallista kompleksisuutta liikaa. Sigmoid-funktio on logistinen funktio ja sille toinen yleisesti käytetty vaihtoehtoinen logistinen funktio on hyperbolinen tangenttifunktio

$$tanh(x) = \frac{e^x - e^x}{e^x + e^x}. \tag{37}$$

Hyperbolisen tangenttifunktion arvojoukko on $[-1, 1]$. Myös hyperbolisen tangenttifunktion kanssa kohdataan katoavan gradientin ongelmaa.

Katoavan gradientin ongelman takia sigmoid- ja hyperbolinen tangenttifunktio ovat hieman vanhentuneita ja näille aktivaatiofunktioille on löydetty parempia vaihtoehtoja. Yksi nykyään yleisesti käytetty aktivaatiofunktio on ReLU-funktio (Rectified Linear Unit)

$$ReLU(x) = \begin{cases} x, \text{ jos } x \geq 0 \\ 0, \text{ muulloin} \end{cases}. \tag{38}$$

ReLU-funktion arvojoukko on $[0, \infty)$ [6]. ReLU-funktio ei ole differentioituva pisteessä $x = 0$. Tämä ei ole ongelma, sillä funktiosta voidaan käyttää vasenta derivaattaa, joka on

$$ReLU'(x) = \begin{cases} 1, \text{ jos } x \geq 0 \\ 0, \text{ muulloin} \end{cases} \tag{39}$$

[2]. ReLU-funktio on laskennallisesti huomattavasti yksinkertaisempi kuin sigmoid- ja hyperbolinen tangenttifunktio. ReLU-funktion kanssa kohdataan kuitenkin ongelma gradientin katoamisesta negatiivisilla arvoilla. Siitä huolimatta ReLU-funktio on

hyvin suosittu aktivaatiofunktio syvissä neuroverkoissa. Negatiivisilla arvoilla tapahtuvasta gradientin katoamisesta päästään eroon käyttämällä LReLU-funktio (Leaky Rectified Linear Unit)

$$LReLU(x) = \begin{cases} x, \text{ jos } x \geq 0 \\ \alpha x, \text{ muulloin} \end{cases} . \tag{40}$$

Tässä α on arvoltaan, jotain pientä esimerkiksi $\alpha = 0.01$, ja se pitää huolen siitä, että myös negatiivisilla arvoilla tapahtuu oppimista [2] [6]. Artikkelissa [6] on esiteltynä lukuisia muitakin mahdollisia aktivaatiofunktioita. Artikkelissa [6] esitellään aktivaatiofunktioiden toimintaa sekä vertailee niiden hyvyyttä.

5.4.2 Koulutustehokkuuden parantaminen

Gradienttimenetelmä on hyvin yksinkertainen tapa kouluttaa neuroverkkoja, mutta se konvergoituu hitaasti. Konvergoitumisen nopeuttamiseksi on kehitetty erilaisia metodeita, joita esitellään tässä luvussa.

Momenttumiksi kutsutaan gradienttien arvoista laskettua keskiarvoa. Keskiarvo lasketaan erikseen jokaiselle painokertoimelle w_i ja sitä pidetään yllä muuttujassa s_i. Painokertoimien päivityksessä käytetään keskiarvoa ja sen avulla pyritään välttämään konvergoitumista hidastavaa heilahtelua, jonka voi aiheuttaa suuret erot perättäisten päivitysten välillä. Keskiarvo s_i lasketaan

$$s_i^t = \alpha s_i^{t-1} + (1 - \alpha)\frac{\partial E^t}{\partial w_i}, \tag{41}$$

missä t on aikaindeksi, joka vastaa reaaliaikaisessa oppimisessa käytettyä aikaindeksiä. Kaavassa α on unohdustekijä, jolle pätee $0 < \alpha < 1$, ja yleensä sille annetaan arvo 0.9 [2]. Momenttumia hyödyntäen painojen päivityssäännöksi saadaan

$$\Delta w_i^t = -\eta s_i^t. \tag{42}$$

Tämä päivityssääntö on käytössä kaikille neuroverkon painoille. Momenttumin käyttämisessä on haittapuolena ylimääräinen muistintarve, jonka keskiarvon ylläpitäminen vaatii.

Oppimistekijä vastaa siitä kuinka suuria muutoksia painojen päivityksissä tehdään. Yleensä oppimistekijän arvo ei ole yli 0.2. Konvergoitumisen nopeuttamiseksi on kehitetty adaptiivinen oppimistekijä. Oppimistekijän arvo pidetään suurempana, kun oppimista tapahtuu, ja sen arvoa pienennetään oppimisen hidastuessa. Etenkin syvissä neuroverkoissa kaikkien painokertoimien vaikutus ennusteen ja havaitun arvon väliseen erotukseen ei ole yhtä suurta, ja tämän takia adaptiivinen oppimistekijä voidaan määritellä erikseen jokaiselle painokertoimelle. Erilaisia adaptiivisia oppimistekijöitä on useampia ja yksi niistä on Geoff Hintonin vuonna 2012 kehittämä RMSProp. Siinä pidetään yllä liukuvaa keskiarvoa kaikista aikaisemmista gradienteista. Keskiarvon laskennassa painotetaan viimeisimpiä gradientteja. Mille tahansa neuroverkon painokertoimelle w_i päivityssääntö on

$$\Delta w_i^t = -\frac{\eta}{\sqrt{r_i^t}}\frac{\partial E^t}{\partial w_i}, \tag{43}$$

missä r_i on jokaiselle painokertoimelle erikseen laskettu liukuva keskiarvo aikaisemmista gradienteista. Sen arvo on alussa 0, mutta se päivitetään jokaisen epookin jälkeen kaavalla

$$r_i^t = \rho r_i^{t-1} + (1-\rho)\left|\frac{\partial E^t}{\partial w_i}\right|^2, \tag{44}$$

missä yleensä $\rho = 0,999$ [2]. Artikkelissa [15] on esitelty adaptiivinen oppimistekijä Adam.

Syötearvojen normalisointi samalle skaalalle on hyvin yleistä. Normalisoinnin jälkeen syötearvojen keskiarvo on 0 ja varianssi on 1. Normalisoinnin avulla hyvin pienelle alueelle klusteroituneet syötearvot pystytään hajauttamaan laajemmalle alueelle. Kun syötearvot ovat samalla skaalalla, myös niitä vastaavat painokertoimet ovat samalla skaalalla. Tällöin niiden alustaminen samalle vaihteluvälille on perusteltua. Neuroverkkojen piiloyksiköiden laskemat arvot voidaan myös normalisoida ennen niiden syöttämistä käytössä olevalle aktivaatiofunktiolle. Tätä kutsutaan erän normalisoinniksi (engl. *batch normalization*). Merkitään jokaisen piiloyksikön j laskemaa painotettua summaa a_j. Lasketaan jokaiselle satsille tai minisatsille keskiarvo m_j ja keskihajonta s_j, ja z-normalisoidaan a_j seuraavasti

$$\tilde{a}_j = \frac{a_j - m_j}{s_j}. \tag{45}$$

Normalisoinnin jälkeen painotetut summat voidaan muuntaa siten, että niillä on mielivaltainen keskiarvo ja skaala

$$\hat{a} = \gamma_j \tilde{a}_j + \beta_j. \tag{46}$$

Nyt $\hat{a}$ voidaan syöttää aktivaatiofunktioon. Tässä γ_j ja β_j ovat parametreja, joita päivitetään jokaisen erän tai pienerän jälkeen. Nämä parametrit mahdollistavat satunnaisen keskiarvon ja keskihajonnan jokaiselle piiloyksikön aktivaatiolle. Joidenkin pienerien syötteet voivat olla liian lähekkäin toisiaan, mikä johtaa hyvin pieniin eroihin a_j arvoissa. Erän normalisoinnilla erot saadaan selvemmiksi. Erän normalisoinnin käyttämiseen on myös toinen syy. Piilokerrosten arvot riippuvat aikaisempien kerrosten arvoista, joita päivitetään koko oppimisen ajan. Tämä tarkoittaa vaihtelua a_j arvoissa, kun neuroverkon aikaisempia painokertoimia päivitetään. Erän normalisoinnilla päästään eroon tästä vaihtelusta.

5.5 Konvoluutioneuroverkko

Konvoluutioneuroverkot ovat erikoisversioita neuroverkoista, jotka toimivat erityisen hyvin ruudukkomallisissa datoissa. Tällaisista datoista esimerkkejä ovat aikasarjadata, jota voidaan ajatella 1-ulotteisena ruudukkona, jossa tasaiset aikaintervallit toimivat ruutuina ja kaksiulotteiset kuvat, joissa ruudukko muodostuu kuvan pikseleistä [13]. Ruudukkomallisissa datoissa datapisteen sijainnilla on yleensä suuri merkitys. Kaksiulotteisissa kuvissa vierekkäiset pikselit ovat esimerkiksi usein samaa värisävyä [1].

Konvoluutioneuroverkon toiminnan perustana onkin oletus siitä, että kaikki syötteen datapisteet eivät korreloi keskenään, vaan korrelaatiota on havaittavissa vain

lähellä sijaitsevien datapisteiden kanssa. Konvoluutioneuroverkkoja ovat sellaiset neuroverkot, joissa on vähintään yksi konvoluutiokerros. Konvoluutiokerros poikkeaa normaalista neuroverkon kerroksesta siten että konvoluutiokerroksen piiloyksiköt saavat syötteekseen vain osajoukon syötearvojen joukosta. Konvoluutioneuroverkossa voi olla useita konvoluutiokerroksia päällekkäin [2]. Yleisimpiä konvoluutioneuroverkon osia ovat konvoluutiokerros, pooling-kerros ja ReLU-aktivaatio, jotka esitellään myöhemmin tutkielmassa. Konvoluutioneuroverkon lopussa on yleensä normaali täysin yhdistetty neuroverkkokerros. Datapisteiden paikkakohtaisuus säilyy konvoluutiokerroksia ylöspäin mentäessä, sillä jokaisen ylemmän kerroksen osajoukko polveutuu aikaisemman kerroksen osajoukosta [1].

5.5.1 Rakenne

Jokainen konvoluutiokerros on kolmiulotteinen ruudukkorakenne, jolla on korkeus, leveys ja syvyys. Syvyys kertoo kerroksen kanavien (engl. *channel*) määrän. Käytetään esimerkkisyötteenä konvoluutioneuroverkolle kaksiulotteista värillistä kuvaa. Kuva on kaksiulotteinen ruudukkorakenne, jossa jokainen yksittäinen kohta on nimetty pikseliksi. RGB-värimalli on yleinen tapa käsitellä kuvien värit. Siinä värit muodostetaan määrittelemällä jokaisen päävärin intensiteetti. Jokainen pääväreistä on nyt yksi kuvan kanavista. Pikseleiden määrä, 32 pikseliä leveässä värikuvassa, on siis $32 \times 32 \times 3$. Konvoluutioneuroverkon piilokerroksissa kanavat voivat värien sijaan vastata erilaisia muotoja tietyillä kuvan alueilla.

Syötteen kokoa q:nnella kerroksella voidaan esittää seuraavasti: $L_q \times B_q \times d_q$, missä L on korkeus, B on leveys ja d on syvyys. Ensimmäisessä konvoluutiokerroksessa nämä arvot saadaan suoraan kuvan rakenteesta. Nämä samat ulottuvuudet säilyvät läpi koko konvoluutioneuroverkon, mutta jatkokerroksilla kaksiulotteinen ruudukko ei enää vastaa suoraan syötteen pikseleitä. Kerroksille $q > 1$ näitä ruudukkorakenteita kutsutaankin ominaisuuskartoiksi (engl. *feature maps*) ja niiden arvot vastaavat tavallisen neuroverkon piilokerroksen arvoja.

Konvoluutiokerroksissa parametrit ovat kolmiulotteisissa muodostelmissa ja niitä kutsutaan *kerneleiksi*. Kernelit ovat kaksiulotteisesti katsottuna neliöitä, ja niiden koko on huomattavasti pienempi kuin syöte johon kerneliä käytetään. Kernelin syvyys on aina sama, kuin sen kohteena olevan kerroksen syvyys. On yleistä käyttää kerneliä jonka leveyden/korkeuden arvo on pieni ja pariton. Yleisimpiä arvoja ovat esimerkiksi 5 ja 3. RGB-kuvan ollessa syötteenä ensimmäisen konvoluutiokerroksen kernelin syvyys on 3, jolloin se vastaa syötteen syvyyttä. Käytetään q:nnella kerroksella filterin koosta merkintää: $F_q \times F_q \times d_q$, missä F on filtterin särmän pituus ja d syvyys [1].

5.5.2 Konvoluutio

Konvoluutio-operaatio sijoittaa kernelin kaikkiin mahdollisiin syötteen kohtiin siten että se käy koko syötteen läpi ja laskee pistetulon kernelin ja syötteen arvoille. Jokainen laskettu pistetulo on tuotetun ominaisuuskartan alkio. Näin ollen ominaisuuskartan leveys ja korkeus määräytyy sen mukaan, kuinka monta kertaa kernel voidaan asettaa syötteen päälle. Seuraavaksi esitetään laskentakaavat $q + 1$ kerroksen leveydelle ja korkeudelle, sillä oletuksella, että kernel on täysin syötteen rajojen

sisällä.

$$L_{q+1} = (L_q - F_q + 1) \tag{47}$$
$$B_{q+1} = (B_q - F_q + 1) \tag{48}$$

Täten pistetulojen määrä on siis $L_{q+1} \times B_{q+1}$, joka on siis samalla ominaisuuskartan koko. Ulostulon syvyys vastaa ominaisuuskarttojen määrää. Ominaisuuskarttojen määrä on sama kuin käytettyjen filttereiden määrä. Määritellään seuraavaksi konvoluutio-operaatio.

Määritelmä 6. Kolmiulotteinen tensori $W^{(p,q)} = [w^{(p,q)}_{i,j,k}]$ pitää sisällään p:nnen kernelin q:nnen kerroksen parametrit. Indeksit i, j, k kertovat sijainnin kernelissä leveyden, korkeuden ja syvyyden suhteen. q:nnen kerroksen ominaisuuskartta on esitetty kolmiulotteisessa tensorissa kaavalla $H^{(q)} = [h^{(q)}_{i,j,k}]$. $H^{(1)}$ on syötekerros. Nyt *konvoluutio-operaatio* q:nnesta kerroksesta kerrokseen $q+1$ voidaan esittää seuraavasti

$$h^{(q+1)}_{i,j,p} = \sum_{r=1}^{F_q} \sum_{s=1}^{F_q} \sum_{k=1}^{d_q} w^{(p,q)}_{r,s,k} h^{(q)}_{i+r-1,j+s-1,k} \quad \begin{array}{l} \forall i \in \{1, \ldots, L_q - F_q + 1\} \\ \forall j \in \{1, \ldots, B_q - F_q + 1\} \\ \forall p \in \{1, \ldots, d_{q+1}\} \end{array} \tag{49}$$

5.5.3 Reunatäyttö

Aikaisemmin esitettiin, että ominaisuuskartta on aina pienempi kuin kerroksen syöte. Tämä pieneneminen, ja sen myötä mahdollisesti tapahtuva informaation katoaminen kuvan reunoilla, voidaan välttää reunatäytöllä (engl. *padding*). Reunatäytössä syötteen jokaiselle reunalle lisätään nollarivejä/-sarakkeita. Rivien ja sarakkeiden määrä puolittaisessa reunatäytössä (engl. *half padding*) on $(F_q - 1)/2$. Näin syötteen leveys ja korkeus molemmat kasvavat $(F_q - 1)$ kappaletta, joka on yhtä suuri kuin kaavoissa (47) (48) esitetyt pienenemiset. Näin pieneneminen voidaan estää ilman että vaikutetaan laskentaan. Ilman reunatäyttöä ominaisuudet kuvan reunoilla vaikuttavat vähemmän kerroksen ulostuloon. Jotta ominaisuuden sijainnilla suhteessa kuvan keskustaan ei olisi merkitystä, on yleistä käyttää reunatäyttöä jokaisessa konvoluutiokerroksessa [1].

5.5.4 Askellus

Normaalisti konvoluutio-operaatiossa kernel käy läpi jokaisen syötteen pisteen. Ulostulon kokoa voidaan pienentää jättämällä osa pisteistä käymättä. *Askelluksella* (engl. *stride*) voidaan säädellä kuinka monta pikseliä kernel liikkuu kerrallaan. Askellus vaikuttaa myös korkeussuunnassa liikkumiseen. Askellusta kerroksessa q merkitään S_q. Askellus pitää huomioida ulostulon leveyden ja korkeuden laskemisessa. Näin ollen kaavat (47) (48) päivittyvät muotoihin:

$$L_{q+1} = (L_q - P_q)/S_q + 1 \tag{50}$$
$$B_{q+1} = (B_q - P_q)/S_q + 1 \tag{51}$$

Tavallisinta on käyttää askelluksen arvoa 1. Joissain tapauksissa myös askelluksen arvo 2 toimii, mutta on todella harvinaista käyttää suurempaa kuin 2 [1] [16].

5.5.5 Pooling

Pooling-kerroksessa tapahtuva *pooling-operaatio* vastaa idealtaan konvoluutio-operaatiota. $P_q \times P_q$-kokoinen ruudukko käy läpi kaikki kerroksen q syötteen kohdat. Operaatio tuottaa yhden arvon ulostuloonsa ja siirtyy seuraavaan kohtaan. Arvon suuruus riippuu siitä, käytetäänkö keskiarvo-poolingia vai maksimi-poolingia. Maksimi-pooling valitsee kaikkien kohtien suurimman arvon, kun taas keskiarvo-pooling laskee kaikkien ruudukon arvojen keskiarvon. Myös pooling-operaatiossa voi käyttää askellusta. Pooling-kerroksen ulostulon koko lasketaan seuraavasti.

$$L_{q+1} = (L_q - F_q)/S_q + 1 \tag{52}$$
$$B_{q+1} = (B_q - F_q)/S_q + 1 \tag{53}$$

Pooling-kerros pienentää ominaisuuskartan kokoa huomattavasti ja tämän takia ei tarvita montaa konvoluutioneuroverkon arkkitehtuurissa. Pooling-operaatiolla ja konvoluutio-operaatiolla on yksi merkittävä ero. Pooling-operaatio ei muuta ominaisuuskarttojen määrää, vaan käsittelee jokaisen kanavan yksitellen ja säilyttää näin syvyyden ennallaan.

Pooling-operaatiossa käytetään usein ruudukkoa, jonka koko on 2×2 ja askellus $= 2$. Näillä asetuksilla vältetään pooling-operaatioiden päällekkäisyyksiä ja näin ominaisuuskarttojen kokoa saadaan pienennettyä ilman suurta informaation menetystä. Max-pooling on kahdesta pooling-tyypistä yleisempi ja siinä poimitaan kuvan dominoivimmat ominaisuudet. Pooling-kerrosten avulla tapahtuva ominaisuuskarttojen pieneneminen nopeuttaa ja keventää mallien kouluttamista ja näin pystytään kouluttamaan syvempiä neuroverkkoja ja parantamaan tuloksia [1].

6 DBSCAN

Klusterointi on ohjaamattomaan oppimiseen perustuvaa luokittelua. Klusteroinnissa pyritään etsimään luonnollisia ryhmiä datasta ilman tietoa alkuperäisistä ryhmistä [22]. Luokitelussa yleisesti hyvin toimivien klusterointialgoritmien kanssa kohdataan ongelmia kun luokittelu pitää suorittaa suuressa paikkasidonnaisessa datassa (engl. *spatial data*). Tällaisten datojen kanssa seuraavat kolme vaatimusta pätevät klusterointialgoritmille:

- Syöteparametrien määrittämisen minimivaatimus on ympäristön (engl. *domain*) tunteminen, koska tarkkoja tietoja ei usein ole laajojen datojen kanssa saatavilla tai niitä ei pystytä määrittämään.
- Kyky löytää satunnaisen muotoisia klustereita.
- Hyvä tehokkuus suurissa datoissa [7].

Tässä luvussa esitellään DBSCAN-klusterointialgoritmi (Density-based spatial clustering of applications with noise), joka täyttää edellä esitellyt vaatimukset. DBSCAN-algoritmi on julkaistu artikkelissa [7] ja tämän luvun sisältö pohjautuu kyseiseen artikkeliin.

DBSCAN tarvitsee vain yhden syöteparametrin ja algoritmi tukee käyttäjää parametrin arvon asettamisessa. DBSCAN on tehokas suurissa datoissa ja pystyy määrittelemään satunnaisen muotoisia klustereita. DBSCAN soveltuu sekä kaksi- että kolmiulotteiseen euklidiseen avaruuteen kuin myös korkeamman ulottuvuuden piirreavaruuteen.

6.1 Klusterin määrittely tiheyden avulla

Artikkelissa klusterit määritellään pisteiden tiheyden avulla. Tämä pohjautuu siihen, että klustereissa pisteitä on tiheämmin, kuin ympäröivässä häiriössä. Jokaisen klusteripisteen naapurustossa on oltava minimimäärä pisteitä, eli annetulla säteellä olevan naapuruston pistetiheyden on ylitettävä tietty kynnysarvo. Naapuuruston muoto määräytyy pisteiden p ja q välisen etäisyyden laskemisessa käytetyn funktion mukaan. Käytetään etäisyydestä merkintää $dist(p, q)$. Esimerkiksi *Manhattanin-etäisyyttä* käytettäessä kaksiulotteisessa datassa naapurusto on muodoltaan suorakulmainen. Algoritmi toimii kaikilla etäisyysfunktioilla, mutta yksinkertaisuuden vuoksi käsitellään kaikki tämän luvun esimerkit kaksiulotteisen avaruuden Euklidisella etäisyydellä. Käytetään datasta merkintää D ja määritellään seuraavaksi pisteen ϵ-naapurusto.

Määritelmä 7. Pisteen p *ϵ-naapurusto*, josta käytetään merkintää $N_\epsilon(p)$ määritellään seuraavasti

$$N_\epsilon(p) = \{q \in D | dist(p, q) \leq \epsilon\}. \tag{54}$$

Klustereissa on kahdenlaisia pisteitä, joita ovat *ydinpisteet* ja *reunapisteet*. Ydinpisteet sijaitsevat klusterin sisällä ja reunapisteet klusterin reunalla, ja yleensä reunapisteen ϵ-naapurusto pitää sisällään huomattavasti vähemmän pisteitä verrattuna ydinpisteen ϵ-naapurustoon. Näin ollen klusteripisteen kriteeriksi ei voida suoraan

asettaa minimipistemäärää, josta käytetään jatkossa merkintää minPts, pisteen ϵ-naapurustossa. Tällöin minPts pitäisi asettaa erittäin matalaksi, jolloin häiriö ei enää erottuisi klustereista. Tämän ongelman välttämiseksi on kehitetty *suoraan tiheyden suhteen saavutettavuus* (engl. *directly density-reachable*), jossa vaaditaan, että jokaiselle klusterin C pisteelle p on olemassa piste q klusterissa C siten että p on pisteen q ϵ-naapurustossa ja $N_\epsilon(q)$ pitää sisällään vähintään minPts määrän pisteitä. Seuraavaksi on esitelty suoraan tiheyden suhteen saavutettavuuden määritelmä.

Määritelmä 8. Piste p on *suoraan tiheyden suhteen saavutettava* pisteestä q, jos

- $p \in N_\epsilon(q)$ ja
- $|N_\epsilon(q)| \geq minPts$ (ydinpiste-ehto).

Suoraan tiheyden suhteen saavutettavuus on symmetrinen kahden ydinpisteen välillä, mutta ydinpisteen ja reunapisteen välillä symmetria ei ole voimassa. Seuraavaksi esitetään tiheyden suhteen saavutettavuus, joka on kanooninen laajennus suoraan tiheyden suhteen saavutettavuudelle.

Määritelmä 9. Piste p on *tiheyden suhteen saavutettava* pisteestä q, jos on olemassa jono pisteitä $p_q, \ldots, p_n, p_1 = q, p_n = p$, siten että p_{i+1} on suoraan tiheyden suhteen saavutettava pisteestä p_i.

Saman klusterin C kaksi reunapistettä eivät välttämättä ole tiheyden suhteen saavutettavia, sillä ydinpiste-ehto ei välttämättä toteudu molemmille pisteille. Siitä huolimatta klusterissa C pitää olla olemassa jokin ydinpiste, josta molemmat reunapisteet ovat tiheyden suhteen saavutettavia. Seuraavaksi esitellään määritelmä pisteille, jotka ovat *tiheyden suhteen kytkettyjä* (engl. *density-connected*), jonka avulla voidaan varmistaa tiheyden suhteen saatavuus kahden reunapisteen välillä.

Määritelmä 10. Piste p on *tiheyden suhteen kytketty* pisteeseen q, jos on olemassa piste o, siten että molemmat pisteet p ja q ovat tiheyden suhteen saavutettavia pisteestä o.

Näiden määritelmien avulla pystytään antamaan tiheyteen perustuva klusterin määritelmä. Klusterin määritellään olevan joukko tiheyden suhteen kytkettyjä pisteitä. Häiriö määritellään annettujen klustereiden suhteen ja häiriötä ovat ne pisteet tietokannassa, jotka eivät kuulu mihinkään tietokannan klustereista. Esitetään seuraavaksi klusterin ja häiriön määritelmät.

Määritelmä 11. Olkoon D pisteistä koostuva tietokanta. *Klusteri* C on epätyhjä D:n osajoukko, joka täyttää seuraavat ehdot

- $\forall\, p, q$: jos $p \in C$ ja q on tiheyden suhteen saavutettava pisteestä p, niin $q \in C$ (Maksimaalisuus)
- $\forall\, p, q \in C : p$ on tiheyden suhteen kytketty pisteeseen q (Yhteys).

Määritelmä 12. Olkoon $C_1 \ldots, C_k$ klustereita tietokannassa D. Tällöin *häiriö* on joukko pisteitä tietokannassa D, jotka eivät kuuluu mihinkään klusteriin C_i, täten häiriö$= \{p \in D | \forall i : p \notin C_i\}$.

Klusteri C pitää sisällään vähintään $minPts$ määrän pisteitä. Tämä perustuu siihen, että klusterissa on vähintään yksi piste p ja pisteen p pitää olla tiheyden suhteen kytketty itseensä, jonkun pisteen o kautta (o voi olla p). Täten ainakin pisteen o pitää täyttää ydinpiste-ehto ja näin ollen pisteen o ϵ-naapurusto pitää sisällään vähintään $minPts$ määrän pisteitä.

Määritellään seuraavaksi kaksi lemmaa, jotka ovat tärkeitä algoritmin validoinnissa.

Lemma 1. *Olkoon p piste D:ssä ja $|N_\epsilon(p)| \geq minPts$. Näin joukko $O = \{o|o \in D$ ja o on tiheyden suhteen saavutettavissa pisteestä p $\}$ on klusteri.*

Ei ole itsestään selvää, että klusteri C on yksikäsitteisesti määritelty minkään sen ydinpisteen avulla. Jokainen piste klusterissa C on kuitenkin tiheyden suhteen saavutettavia kaikista C:n ydinpisteistä ja näin ollen klusteri C sisältää tarkalleen ne pisteet, jotka ovat tiheyden suhteen saavutettavia satunnaisesta C:n ydinpisteestä.

Lemma 2. *Olkoon C klusteri ja piste p, jokin klusterin C piste ehdolla $|N_\epsilon(p)| \geq minPts$. Näin C on yhtäsuuri joukon $O = \{o|$ o on tiheyden suhteen saavutettava pisteestä p $\}$ kanssa.*

6.2 Algoritmin toiminta

DBSCAN erottelee paikkadatasta määritelmän (11) mukaiset klusterit ja määritelmän (12) mukaiset häiriöt. Ideaalitilanteessa tarvittaisiin tiedot jokaisen klusterin parametreista ϵ ja minPts sekä yhdestä klusteriin kuuluvasta pisteestä. Näin voitaisiin etsiä kaikki pisteet, jotka ovat tiheyden suhteen saavutettavia annetusta pisteestä käyttämällä oikeita parametreja. Näitä tietoja ei kuitenkaan yleensä ole saatavilla etukäteen jokaisesta datan klusterista. Myöhemmin esitellään tehokas tapa ϵ ja $minPts$ parametrien määrittämiselle datan ohuimmalle, eli tiheyden suhteen pienimmälle, klusterille. DBSCAN käyttää näitä parametreja kaikille klustereille. Parametrien arvot toimivat hyvin globaalisti, koska ne ovat peräisin tiheyden suhteen pienimmästä klusterista, joka ei kuitenkaan ole häiriötä.

DBSCAN aloittaa yhdestä satunnaisesta pisteestä ja etsii kaikki pisteet, jotka ovat tiheyden suhteen saavutettavia aloituspisteestä p. Jos p on ydinpiste, tämä vaihe tuottaa klusterin lemman (2) mukaan. Jos taas piste p on reunapiste, ei yksikään piste ole tiheyden suhteen saavutettava pisteestä p ja algoritmi siirtyy tarkastelemaan seuraavaa pistettä. Koska käytössä on globaalit arvot parametreille ϵ ja $minPts$, saattaa algoritmi yhdistää kaksi klusteria määritelmän (11) mukaisesti jos kaksi eri tiheyksistä klusteria sijaitsevat lähellä toisiaan. Olkoon kahden pistejoukon S_1 ja S_2 etäisyys toisistaan $dist(S_1, S_2) = \min\{dist(p, q)|p \in S_1, q \in S_2\}$. Nyt kaksi pistejoukkoa, joilla on vähintään ohuimman klusterin tiheys, erotetaan toisistaan vain, jos joukkojen etäisyys toisistaan on suurempi kuin ϵ. Näissä tilanteissa algoritmin rekursiivinen kutsuminen voi olla tarpeellista klustereiden kohdalla, joilla on suuri $minPts$-arvo. Tämä ei kuitenkaan ole haittapuoli, koska algoritmin rekursiivinen käyttö tuottaa elegantin ja todella tehokkaan perusalgoritmin. Lisäksi pisteiden rekursiivinen klusterointi on tarpeellista vain helposti havaittavissa tilanteissa. Kahden klusterin C_1 ja C_2 sijaitessa todella lähellä toisiaan, on mahdollista, että jokin piste p kuuluu molempiin klustereihin C_1 ja C_2. Tällöin pisteen p on oltava rajapiste

molemmissa klustereissa, sillä muuten $C_1 = C_2$. Tämä johtuu siitä, että käytetään globaaleita parametreja. Tällaisessa tapauksessa piste p sijoitetaan ensimmäisenä löydettyyn klusteriin. Tällaista harvinaista tilannetta lukuunottamatta DBSCAN on riippumaton pisteiden läpikäymisjärjestyksestä lemman (2) nojalla.

6.3 Parametrien ϵ ja $minPts$ määrittäminen

Tässä aliluvussa esitellään yksinkertainen ja tehokas heuristiikka ohuimman klusterin ϵ ja $minPts$ parametrien määrittämiselle. Heuristiikka perustuu seuraavaan havaintoon: olkoon d etäisyys pisteestä p sen k:nteen lähimpään naapuriin. Tällöin pisteen p d-naapurusto pitää sisällään täsmälleen $k+1$ pistettä suurimmalla osalla pisteistä p. Pisteitä voi olla enemmän vain jos useammalla pisteellä on täsmälleen sama etäisyys d pisteestä p, mikä on erittäin epätodennäköistä. Lisäksi k:n arvon vaihtaminen ei aiheuta suurta muutosta etäisyydessä d. Muutos tapahtuu vain jos pisteen p kaikki k naapuria $k = 1, 2, 3 \ldots$ ovat sijoittuneet suoralle linjalle, mikä ei ole todennäköistä.

Annetulle arvolle k määritellään funktio $k - dist : D \to \mathbb{R}$, joka kuvaa kaikkien pisteiden etäisyyden pisteen k:nnesta naapurista. Kun datan pisteet järjestetään laskevaan järjestykseen $k - dist$ arvon mukaan tämän funktion kuvaaja antaa vihjeitä datan tiheyden jaukaumasta. Tätä kuvaajaa kutsutaan järjestetyksi $k - dist$ kuvaajaksi. Jos valitaan satunnainen piste p ja ϵ sen $k - dist(p)$ parametriksi, ja k $minPts$ parametriksi, niin kaikki pisteet, joilla on pienempi tai yhtäsuuri $k-dist$ arvo ovat ydinpisteitä. Jos olisi mahdollista löytää kynnysarvopiste, jolla on maksimi $k-dist$ arvo datan D ohuimmassa klusterissa, olisi nämä halutut parametrin arvot. Kaikki arvot kynnysarvopisteen vasemmalla puolella järjestetyssä pistelistassa ovat häiriötä, kun taas kaikki pisteet sen oikealla puolella ovat osa jotain klusteria.

Optimaalisen kynnysarvopisteen löytäminen voi olla hankalaa. Artikkelin kirjoittajan tutkimukset ovat osoittaneet, että arvoilla $k > 4$ laskutehoon nähden saatu hyöty ei ole tarpeeksi arvokasta. Näin ollen kirjoittaja kehottaa pitämään $minPts$ arvon 4 kaikilla datoilla. Näin ollen jäljelle jää yksi määritettävä parametri ϵ,jonka määrittämiselle on kehitetty interaktiivinen metodi:

- Systeemi laskee ja esittää $4 - dist$ kuvaajan datalle.
- Jos käyttäjä pystyy arvioimaan häiriön prosentuaalisen määrän, tämä prosenttiarvo syötetään järjestelmään, joka arvioi kynnysarvopisteen arvion avulla.
- Käyttäjä voi joko hyväksyä systeemin antaman arvon tai valita eri pisteen kynnysarvopisteeksi. Kynnysarvopisteen $4-dist$ arvoa käytetään ϵ parametrin arvona DCSCAN algoritmissa.

7 APOLLO

APOLLO-prosessi (Raman-based Pathology of malignant glioma) esitellään artikkelissa [18]. Prosessin tarkoituksen on pyrkiä luokittelemaan Raman-spektroskopian avulla saadut glioomanäytteet. Tässä tutkielmassa tarkastellaan prosessin kolmea vaihetta. Nämä vaiheet ovat klusterointi, binäärinen luokittelu ja $K = 6$ luokan luokittelu.

Ensimmäisenä vaiheena oli todistaa, että hyödyntämällä ohjaamatonta oppimista Raman näytteistä pystytään erottelemaan kasvainsolut ei-kasvainsoluista. Tämä prosessin vaihe on esitelty tarkemmin artikkelissa [19].

Toinen vaihe oli kouluttaa koneoppimismalli luokittelemaan glioomanäytteet villityyppisiin ja mutatoituviin kasvaimiin. Tämä pystyttiin suorittamaan kohtuullisen hyvällä tarkkuuden arvolla, joka oli noin 80% luokkaa.

Kolmantena vaiheena oli kouluttaa koneoppimismalli luokittelemaan glioomanäytteet kaikkiin kuuteen eri luokkaan LGm 1-6. Tässä ei projektissa onnistuttu riittävällä tarkkuudella.

Kaikki nämä alivaiheet vaativat datan esikäsittelyn. Datan esikäsittelyssä on käytetty airPLS-algoritmia pohjatason korjaukseen (adaptive iteratively reweighted Penalized Least Square). Seuraavassa aliluvussa esitellään algoritmi ja sen toiminta, jotka on julkaistu artikkelissa [36]. Aliluvun (7.1) jälkeen esitellään prosessissa käytetty data ja sen käsittely.

7.1 AirPLS

Pohjatason siirtymä sumentaa ja kadottaa signaaleja ja vaikeuttaa ja heikentää näin analyysien tekoa spektri-datan pohjalta. Tästä johtuen pohjatason korjaus on tärkeä suorittaa ennen datan analysointia. Adaptive iteratively reweighted Penalized Least Square (airPLS) on algoritmi, joka ei vaadi käyttäjän puuttumista eikä aiempaa informaatiota, kuten huippujen havainnointia (engl. *peak detection*), toisin kuin monet muut pohjatason korjauksen metodit.

AirPLS-algoritmi hyödyntää Whittakerin vuonna 1992 kehittämää rangaistun pienimmän neliösumman algoritmia, joka on joustava silottelumetodi. PLS-algortimi tasapainottelee alkuperäistä dataa kohtaan olevan tarkkuuden (engl. *fidelity*) ja sovitetun datan karkeuden välillä (engl. *roughness*). Käydään seuraavassa aliluvussa sen toiminta läpi.

7.1.1 Rangaistu pienimmän neliösumman algoritmi

Määritelmä 13. Olkoon $\boldsymbol{x}$ analyyttisen signaalin sisältävä vektori, $\boldsymbol{z}$ sovitettu vektori ja molemmat pituudeltaan m. Merkitään sovitetun vektorin *tarkkuutta* alkuperäistä signaalia kohtaan F. Tämä voidaan ilmaista vektoreiden välisen erotuksen neliöiden summana vektoreiden kaavalla

$$F = \sum_{i=1}^{m} (x_i - z_i)^2. \tag{55}$$

Sovitetun datan *karkeus* määritellään perättäisten alkioiden erotusten neliöiden summana:

$$R = \sum_{i=2}^{m} (z_i - z_{i-1})^2 = \sum_{i=1}^{m-1} (\Delta z_i)^2. \tag{56}$$

AirPLS-algoritmi mahdollistaa erotuksen asteen valitsemisen. Tässä tutkielmassa käytetään merkintöjen selkeyttämiseksi ensimmäisen asteen erotuksia.

Määritelmä 14. Tarkkuuden ja sileyden *tasapaino* voidaan ilmaista tarkkuuden ja karkeuden aiheuttamien sakkojen summana ja siitä käytetään merkintää Q:

$$Q = F + \lambda R = \|\boldsymbol{x} - \boldsymbol{z}\|^2 + \lambda \|\boldsymbol{D}\boldsymbol{z}\|^2. \tag{57}$$

Tässä λ on käyttäjän säädettävissä oleva parametri ja sillä säädellään tarkkuuden ja tasaisuuden suhdetta. Sovitettu vektori on sitä tasaisempi, mitä suurempi on parametrin λ arvo. $\boldsymbol{D}$ on identiteettimatriisin derivaatta, jolle pätee $\boldsymbol{D}\boldsymbol{z} = \boldsymbol{\Delta}\boldsymbol{z}$.

Asettamalla osittaisderivaatta $\frac{\delta Q}{\delta \boldsymbol{z}} = 0$ saadaan lineaarinen yhtälöjoukko, joka on helppo ratkaista

$$(\boldsymbol{I} + \lambda \boldsymbol{D}'\boldsymbol{D})\boldsymbol{z} = \boldsymbol{x}. \tag{58}$$

Yhtälö (58) on silottelumetodi, joka käyttää rangaistua pienimmän neliösumman algoritmia. Jotta PLS-algoritmilla voidaan korjata pohjataso, lisätään tarkkuuden painovektori, jonka arvot ovat nollia vektorin $\boldsymbol{x}$ piikkikohdissa. Tarkkuuden F yhtälö (55) muuttuu nyt muotoon

$$F = \sum_{i=1}^{m} w_i (x_i - z_i)^2 = (\boldsymbol{x} - \boldsymbol{z})^\intercal \boldsymbol{W} (\boldsymbol{x} - \boldsymbol{z}). \tag{59}$$

Tässä $\boldsymbol{W}$ on diagonaalimatriisi, jonka diagonaalilla ovat arvot w_i. Nyt yhtälö (58) muuttuu muotoon

$$\boldsymbol{W}\boldsymbol{x} = (\boldsymbol{W} + \lambda \boldsymbol{D}'\boldsymbol{D})\boldsymbol{z}. \tag{60}$$

Yllä olevan lineaariyhtälön ratkaisuna saadaan sovitettu vektori

$$\boldsymbol{z} = (\boldsymbol{W} + \lambda \boldsymbol{D}'\boldsymbol{D})^{-1} \boldsymbol{W}\boldsymbol{x}. \tag{61}$$

PLS-algoritmi tällaisenaan vaatii huippujen havainnointia, jotta sillä voidaan suorittaa pohjatason korjaus. Huippujen havainnointi voi kuitenkin datasta riippuen olla hyvin aikaa vievää. Adaptiivisen iteratiivisen uudelleenpainotus -proseduurin käyttö PLS-alogritmin yhteydessä poistaa huippujen havainnoinnin tarpeen ja näin voi säästää paljonkin resursseja.

7.1.2 Adaptiivinen iteratiivinen uudelleenpainotusproseduuri

Painojen laskentatapa ja virhetermin lisääminen sovitetun pohjatason sileyden kontrolloimiseen erottavat adaptiivisen iteratiivisen uudelleenpainotusproseduurin (air-proseduuri) painotetusta pienimmän neliösummanmetodista ja iteratiivisesta painotetusta pienimmän neliösummanmetodista. Air-proseduurin jokaisessa vaiheessa ratkaistaan paiunotettu rangaistu pienimmän neliösumman ongelma, joka on muotoa

$$\boldsymbol{Q}^t = \sum_{i=1}^{m} w_i^t |x_i - z_i^t|^2 + \lambda \sum_{j=2}^{m} |z_j^t - z_{j-1}^t|^2. \tag{62}$$

Painovektori $\boldsymbol{w}$ saadaan adaptiivisesti käyttämällä iteratiivista menetelmää. Alustetaan $\boldsymbol{w^0} = 1$ askeleella $t = 1$. Alustuksen jälkeen $\boldsymbol{w}$ ratkaistaan jokaisella askeleella käyttäen laskentakaavaa

$$w_i^t = \begin{cases} 0 & x_i \geq z_i^{t-1} \\ \exp \frac{t(x_i - z_i^{t-1})}{|\boldsymbol{d}^t|} & x < z_i^{t-1} \end{cases}. \tag{63}$$

Vektori $\boldsymbol{d}^t$ koostuu vektoreiden $\boldsymbol{x}$ ja $\boldsymbol{z}^{t-1}$ erotuksista iteraatioaskeleella t. Sovitettu arvo $\boldsymbol{z}^{t-1}$ edeltävässä (t-1) iteraatiossa on ehdotus pohjatasoksi. Jos arvo pisteessä i on suurempi kuin ehdotus, arvoa voidaan pitää osana piikkiä. Täten paino asetetaan nollaksi, jotta se jää huomioimatta seuraavalla iteraatiokierroksella. AirPLS-algoritmissa iteratiiviset ja uudelleenpainottavat metodit eliminoivat automaattisesti ja askeleittain piikkien kohdat ja säilyttävät pohjatason kohdat painovektorissa $\boldsymbol{w}$. Iterointi loppuu, kun maksimikierrosluku saavutetaan tai lopetuskriteerin täyttyy. Lopetuskriteeri on määritelty seuraavasti

$$|\boldsymbol{d}_t| < 0.001 \times |\boldsymbol{x}|. \tag{64}$$

Myös tässä vektori $\boldsymbol{d}^t$ koostuu vektoreiden $\boldsymbol{x}$ ja $\boldsymbol{z}^{t-1}$ negatiivisista erotuksista.

7.1.3 Yhteenveto

Artikkelissa esitellään airPLS-algoritmin testauksen tuloksia. Algoritmi todistettiin hyvin toimivaksi myös Raman-spektridatan luokitteluongelmassa. AirPLS-algoritmissa on yksi käyttäjän säätelemä parametri, λ. Parametrin λ optimiarvon voi etsiä esimerkiksi binäärihakualgoritmilla.

7.2 Data

Glioomanäytteitä kerättiin yhteensä 46 potilaalta. Osalta potilaista otettiin useampi kuin yksi näyte ja lopullinen näytteiden määrä oli 59 kappaletta. Näytteet olivat muodoltaan suorakulmioita ja niiden koko vaihteli pienimmästä 46x46 kohdan kokoisesta näytteestä aina 245x61 kohdan kokoiseen näytteeseen. Jokainen kohta jokaisessa näytteessä sai oman yksilöllisen Raman-spektroskopian, jonka pituus oli 1738. Raman-spektri toimi ikään kuin kohdan sormenjälkenä. Raman-spektrin aallonpituudet vaihtelivat välillä 50-3399 cm^{-1}. Näin ollen data koostui 59 kolmiulotteisesta

vektorista. Kolme ulottuvuutta ovat näytteen korkeus, leveys ja jokaisen kohdan sormenjälkimäinen, 1738 pituinen, Raman-spektrin sisältävä vektori.

Usean kirjallisuuslähteen avulla tiedettiin jo etukäteen, että näytteiden spektrit sisältävät niin sanottuja mykkiä alueita (engl. *silent regions*), jotka sijaitsevat vektorissa alkioissa 1-103 ja 907-1374. Näitä alkioita vastaavat Ramanin-aallonpituudet ovat 50-248 cm $^{-1}$ ja 1799-2679 cm $^{-1}$. Hiljaisten alueiden poiston jälkeen dataan sovellettiin pohjatason korjausta. Tähän käytettiin airPLS-algoritmia parametreilla $\lambda = 4$ ja $porder = 1$. Pohjataso korjattiin erikseen molemmista jäljelle jääneistä pätkistä dataa. Algoritmin jäljiltä jäi muutama negatiivinen arvo ja nämä muutettiin nolliksi, jonka jälkeen pätkät yhdistettiin ja vektorin uudeksi pituudeksi muodostui 1166. Kasvainnäytteiden koko vaikuttaa suoraan Ramanin-intensiteettiin tehden niistä vertailukelvottomia. Tämän takia uudet vektorit piti vielä normalisoida jakamalla ne niiden L2-normeilla.

Datan esikäsittelyn jälkeen oli mahdollista siirtyä klusterointiin eli projektin ensimmäiseen alivaiheeseen, joka käydään läpi seuraavassa luvussa.

7.3 Klusterointi

Datan glioomanäytteet sisälsivät sekä terveitä soluja, että kasvainsoluja. Yksi näyte saattoi sisältää useamman eri kasvainalueen. Näytteen jokaiselle solulle oli oma Raman-spektrinsä ja hypoteesina oli, että kasvainsolun Raman-spektri erottuu selkeästi ei-kasvainsolun Raman-spektristä. Klusterointi suoritettiin kaikille 59 glioomanäytteelle ja klusteroinnin avulla pyrittiin erottamaan jokaisen näytteen kohdalla kasvainsolun Raman-spektrit ei-kasvainsolun Raman-spektreistä.

Klusteroinnissa käytettiin DBSCAN-algoritmia, jonka toimintaa esiteltiin luvussa (6). DBSCAN löytää automaattisesti optimaalisen määrän klustereita. Tämän datan kohdalla klustereiden optimaalinen määrä vaihteli yhden ja neljän välillä. H&E-tulosten pohjalta oletettiin, että suurin osa Raman-spektreistä on kasvainsoluista. Klusterit piti vielä tunnistaa kasvaimiksi tai ei-kasvaimiksi. Klustereiden spektrejä verrattiin koko datasta laskettuun spektrikeskiarvoon. Klustereista valittiin se, jonka spektri oli lähimpänä keskiarvospektriä yli koko datan, ja tämä klusteri luokiteltiin kasvainklusteriksi. Loput klustereista ja häiriöalueet luokiteltiin ei-kasvaimiksi. Kaikkien näytteiden kohdalla suurin klustereista oli kasvainklusteri. 86% datasta luokiteltiin klustereiden perusteella kasvaimeksi ja vain 14% ei-kasvaimeksi.

Klusteroinnin tuloksia arvioitiin siluetti-indeksin avulla. Määritellään seuraavaksi siluetti-indeksi. Siluetti-indeksin määritelmä ja selitys ovat julkaistu artikkelissa [31].

Määritelmä 15. *Siluetti-indeksi* $s(x_i)$ datapisteelle x_i on

$$s(x_i) = \frac{b(x_i) - a(x_i)}{\max\{b(x_i), a(x_i)\}}, \tag{65}$$

missä x_i on datapiste klusterissa C_k, $a(x_i)$ on keskiarvoetäisyys pisteestä x_i muihin klusterin C_k pisteisiin ja

$$b(x_i) = \min\{d_l(x_i)\} \text{ kaikista klustereista } C_l, \text{ missä } l \neq k, \tag{66}$$

missä $d_l(x_i)$ on keskiarvoetäosyys pisteestä x_i kaikkiin klusterin C_l pisteisiin, missä $l \neq k$.

Siluetti-indeksin arvot ovat väliltä $[-1, 1]$. Negatiivisia siluetti-indeksin arvoja saadaan tapauksessa $a(x_i) < b(x_i)$, joka tarkoittaa, että datapisteen keskiarvoetäisyys muiden klustereiden pisteisiin on suurempi kuin keskiarvoetäisyys datapisteen oman klusterin muihin datapisteisiin. Negatiiviset siluettiarvot siis kertovat, että datapiste on sijoitettu väärään klusteriin. Positiivinen indeksi kertoo datapisteen olevan oikeassa klusterissa ja mitä lähempänä se on arvoa 1, sitä vahvempi on samankaltaisuus muiden klusterissa olevien pisteiden kanssa. Siluetti-indeksi laskettiin jokaisen näytteen kaikille niille datapisteille, jotka klusteroitiin kasvainklusteriin. Tulosten pohjalta todettiin, että kasvainta olevien datapisteiden välillä on homogeenisyyttä niin yhdessä näytteessä, kuin toisten näytteidenkin välillä. Kolme näytteistä osoittautuivat olevan lähes 100% kasvainta. Yksi näytteistä poistettiin, koska se osoitti heikkoja siluetti-indeksin arvoja.

7.4 Luokittelu

APOLLO-projektin luokitteluvaiheessa oli kaksi eri tavoitetta. Ensimmäisenä koulutettiin malli binääriseen luokitteluun mutanttien ja villityyppien välillä. Toisena tavoitteena oli kouluttaa malli luokittelemaan glioomanäytteet kuuteen eri IDH-mutaatiolukkaan. Molempien luokitteluongelmien kouluttamisessa käytettiin stratifioitua 5-kertaista ristiinvalidointia. Stratifiointi suoritettiin siten että jokaisessa validointidatassa on jokaisesta kuudesta luokasta vähintään yksi näyte ja vähintään 15% luokan näytteistä. Mallien arvioinnissa käytettiin tarkkuutta, täsmällisyyttä, herkkyyttä ja F1-arvoa. Näistä metriikoista laskettiin kaikkien viiden koulutus- ja validointikerran välinen keskiarvo.

7.4.1 Mutantti vs. villityyppi

Glioomakasvaimet voidaan jaotella IDH-mutaatiotyypin mukaan villi- ja mutanttityyppisiksi. Villityyppisiä ovat luokat LGm 1-3 ja niistä käytetään merkintää $IDH1^{mut}$. Mutanttityyppisiä ovat luokat LGm 4-6 ja niitä merkitään $IDH1^{WT}$. Projektin seuraavana vaiheena oli kouluttaa koneoppimismalli tätä binääristäluokitteluongelmaa varten. Tähän ongelmaan koulutettiin useita erilaisia luokittelumalleja, jotka olivat satunnaismetsiä (engl. *random forest*), tukivektorikoneita (engl. *supprot vector machine*), vahvistettuja päätöspuita (engl. *boosted decision trees*) ja konvoluutioneuroverkkoja.

Korkein tarkkuuden keskiarvo 81% saatiin satunnaismetsän ja tukivektorikoneen yhdistelmällä. Yhdistelmässä koulutetaan jokaisella ristiinvalidointi-interaatiolla ensin satunnaismetsä, jonka avulla erotellaan 20 tärkeintä ominaisuutta. Tämän jälkeen tukivektorikone koulutetaan käyttäen samaa ristiinvalidointi-iteraatiota ja eroteltuja ominaisuuksia. Tukivektorikone suorittaa lopullisen luokittelun. Näin tehtiin häiriön minimoimiseksi ja parempien tulosten saavuttamiseksi. Tämän mallin tulokset arviointimetriikoille on esitetty seuraavissa taulukoissa (3) ja (4), joissa kaikki esitetyt arvot ovat keskiarvoja.

Metriikka	Arvo
Tarkkuus	0,813

Taulukko 3: Yhdistelmämallin tarkkus

Metriikka	IDH^{WT}	IDH^{MUT}
Täsmällisyys	0,895	0,741
Herkkyys	0,730	0,895
F1-arvo	0,799	0,808

Taulukko 4: Yhdistelmämallin täsmällisyys, herkkyys ja F1-arvo

Taulukossa (4) on esitetty täsmällistts, herkkyys ja F1-arvo molemmille luokille IDH^{WT} ja IDH^{MUT}.

7.4.2 LGm1-LGm6

Toinen luokitteluongelma APOLLO-projektissa oli $K = 6$ luokan tapaus, jossa näytteet pyrittiin luokittelemaan kaikkiin LGm1-LGm6 luokkiin. Tähänkin luokitteluongelmaan koulutettiin useita eri koneoppimismalleja, mutta kaikilla malleilla tulokset jäivät heikoiksi. Satunnaismetsän ja tukivektorikoneen yhdistelmämallin tulokset on esitelty taulukoissa (5) ja (6), joissa kaikki esitetyt arvot ovat keskiarvoja.

Metriikka	Arvo
Tarkkuus	0,249

Taulukko 5: Yhdistelmämallin tarkkuus

Metriikka	LGm1	LGm2	LGm3	LGm4	LGm5	LGm6
Täsmällisyys	0,260	0,647	0,122	0,224	0,279	0,063
Herkkyys	0,176	0,513	0,166	0,307	0,158	0,172
F1-arvo	0,165	0,561	0,125	0,255	0,198	0,091

Taulukko 6: Yhdistelmämallin täsmällisyys, herkkyys ja F1-arvo

8 Oma työ

Tämän tutkielman tarkoituksena on tutkia tarkemmin konvoluutioneuroverkkojen toimivuutta glioomanäytteiden luokittelussa. APOLLO-prosessin yhteydessä koulutettiin jo konvoluutioneuroverkkoja, mutta arkkitehtuuri- ja hyperparametrivaihotehtoja on valtava määrä, joten tutkimusta oli mahdollista jatkaa. APOLLO-prosessin yhteydessä päästiin konvoluutioneuroverkolla parhaimmillaan n. 80% tarkkuuteen. Tämä arvo toimi vertailukohtana tässä tutkielmassa esitettyjen mallien kanssa. Konvoluutioneuroverkkojen käyttöön Raman-spketrien kanssa kannustivat etenkin artikkeleissa [20] ja [8] saadut hyvät tulokset.

Tässä työssä esitellyt konvoluutioneuroverkot on koulutettu käyttämällä samaa dataa ja datan esikäsittelyä sekä ristiinvalidointimetodia kuin APOLLO-prosessissa. Konvoluutioneuroverkot koulutettiin käyttämällä 200 parasta ominaisuutta, joiden valinta tapahtui käyttämällä APOLLO-prosessin satunnaismetsää. Malleja suunniteltiin 6 kappaletta ja suurin osa niistä pohjautui muissa lähteissä esiteltyihin konvoluutioneuroverkkoihin. Kaikkien mallien rakenteissa yhteisenä tekijänä oli lopun kaksi täysin yhdistettyä kerrosta. Mallit koulutettiin käyttämällä Python-ohjelmointikieltä, ja koulutuksessa hyödynnettiin Tensorflow- ja Keras-kirjastoja.

Kaikki 6 mallia koulutettiin binääriseen luokitteluun IDH^{mut} ja IDH^{WT} luokkien välillä. Tarkoituksena löytää binäärisessä luokittelussa parhaiten toimiva malli ja verrata sitä APOLLO-prosessin tuloksiin. Seuraavaksi on esitelty tämän työn konvoluutioneuroverkkojen rakenteet ja niiden pohjalla olevat lähteet. Kaikki seuraavaksi esitetyt tarkkuuden arvot ovat koulutuksessa käytettyjen ristiinvalidointikierrosten keskiarvoja. Tuloksista ja niiden merkitystä käydään läpi luvussa (9).

- **Malli 1:** Ensimmäinen malli oli rakenteeltaan hyvin yksinkertainen. Tämän mallin tarkoituksena oli asettaa pohjataso tuloksille ja verrata syvyyden vaikutusta mallin laatuun. Mallissa oli vain yksi konvoluutiokerroksen. Tämän mallin koulutuksessa käytetty koodi on avoimesti saatavilla GitHubissa [25]. Mallin tulokset esitelty taulukoissa (7) ja (8).

Metriikka	Arvo
Tarkkuus	0,806

Taulukko 7: Mallin 1 tarkkuus

Metriikka	IDH^{WT}	IDH^{MUT}
Täsmällisyys	0,740	0,885
Herkkyys	0,879	0,734
F1-arvo	0,800	0,796

Taulukko 8: Mallin 1 täsmällisyys, herkkyys ja F1-arvo

- **Malli 2:** Toinen malli pohjautui artikkelissa [20] esiteltyyn konvoluutioneuroverkkoon. Malli oli ensimmäistä huomattavasti syvempi ja se koostui kolmesta konvoluutiokerroksesta. Jokaisen konvoluutiokerroksen jälkeen oli LRelU- ja pooling-kerrokset. Tämä malli oli kuitenkin liian raskas kouluttaa ja tulokset

ensimmäisen ristiinvalidointikierroksen jälkeen vaikuttivat todella huonoilta, joten koulutusta ei suoritettu loppuun.

- **Malli 3A:** Kolmas malli pohjatutui artikkelissa [8] esiteltyyn konvoluutioneuroverkkoon. Mallissa oli 2 konvoluutiokerrosta ja niiden jälkeen maksimipooling-kerrokset. Mallin tulokset esitelty taulukoissa (9) ja (10).

Metriikka	Arvo
Tarkkuus	0,804

Taulukko 9: Mallin 3A tarkkuus

Metriikka	IDH^{WT}	IDH^{MUT}
Täsmällisyys	0,752	0,872
Herkkyys	0,859	0,749
F1-arvo	0,796	0,796

Taulukko 10: Mallin 3A täsmällisyys, herkkyys ja F1-arvo

- **Malli 3B:** Tämä on yksinkertaistettu malli mallista 3A. Kuten tuloksista voidaan huomata, malli 1 on toiminut kaikista parhaiten. Se on malleista myös rakenteeltaan selvästi yksinkertaisin. Näin ollen tämän mallin ideana oli testata mitä käy kun mallista 3A poistetaan toinen konvoluutio- ja maksimipooling-kerros. Mallin tulokset esitelty taulukoissa (11) ja (12). Tuloksista huomataan, että tässä tapauksessa syvempi versio mallista oli parempi.

Metriikka	Arvo
Tarkkuus	0,795

Taulukko 11: Mallin 3B tarkkuus

Metriikka	IDH^{WT}	IDH^{MUT}
Täsmällisyys	0,747	0,854
Herkkyys	0,841	0,75
F1-arvo	0,787	0,792

Taulukko 12: Mallin 3B täsmällisyys, herkkyys ja F1-arvo

- **Malli 4:** Neljäs malli pohjautui yleisesti tunnettuun LeNet-5 -arkkitehtuuriin. LeNet-5 -arkkitehtuurissa on kolme konvoluutiokerrosta, joista kahden ensimmäisen jälkeen on keskiarvopooling-kerros. Mallin tarkkuus oli $0,77$, joten se pääsi hyvin lähelle mallia 3A. Mallin tulokset esitelty taulukoissa (13) ja (14).

Metriikka	Arvo
Tarkkuus	0,774

Taulukko 13: Mallin 4 tarkkuus

Metriikka	IDH^{WT}	IDH^{MUT}
Täsmällisyys	0,740	0,818
Herkkyys	0,791	0,756
F1-arvo	0,760	0,779

Taulukko 14: Mallin 4 täsmällisyys, herkkyys ja F1-arvo

- **Malli 5:** Viides malli pohjautui artikkelissa [11] esiteltyyn konvoluutioneuroverkkoon. Mallissa oli 2 konvoluutiokerrosta, joiden välissä oli ReLU-kerrokset. Mallin tulokset ovat taulukoissa (15) ja (16).

Metriikka	Arvo
Tarkkuus	0,785

Taulukko 15: Mallin 5 tarkkuus

Metriikka	IDH^{WT}	IDH^{MUT}
Täsmällisyys	0,729	0,843
Herkkyys	0,835	0,735
F1-arvo	0,777	0,783

Taulukko 16: Mallin 5 täsmällisyys, herkkyys ja F1-arvo

9 Yhteenveto

APOLLO-prosessissa binääriluokittelusta parhaiten suoriutunut malli oli satunnaismetsän ja tukivektorikoneen yhdistelmä. Tämän mallin tarkkuus oli 81%, ja aikaisemmin tutkielmassa esitellyn lääketieteellisesti tärkeän metriikan, herkkyyden, arvot luokittain olivat $0,730$ ja $0,895$. Tutkielmassa koulutetuista konvoluutioneuroverkoista parhaimman tarkkuus oli 80% ja herkkyyden arvot olivat luokkakohtaisesti $0,879$ ja $0,734$. Tutkielmassa koulutetuilla malleilla pystyttiin lähes toistamaan alkuperäisen prosessin tulokset. Tutkielmassa koulutetuista malleista parhaimmat tulokset saanut oli arkkitehtuuriltaan hyvin yksinkertainen ja matala. Vaikka saadut tulokset vaikuttavatkin kohtalaisen hyviltä, pitää niiden hyvyyttä silti arvioida kriittisesti. Luokittelun kohteena ovat syöpänäytteet, joten voidaan olettaa paremman tarkkuuden vaatimista, jotta koneoppimismallia voitaisiin todellisuudessa hyödyntää diagnosoinnissa. Kuuden luokan luokittelussa tulokset olivat todella heikot. Tuloksien selittävänä tekijänä voi olla datan määrä ja laatu. Glioomanäytteiden määrä oli rajallinen ja luokkien väliset näytemäärät vaihtelivat huomattavasti. Tämä selittää osaltaan, miksi etenkin kuuden luokan luokittelun tulokset jäivät heikoksi.

Yksi mahdollinen ratkaisu tähän ongelmaan voisi olla datan augmentointi. Datan augmentointi on hyvin yleinen metodi esimerkiksi lääketieteellisten kuvadatojen kanssa. Datan augmentoinnissa pyritään kasvattamaan datan määrää esimerkiksi geometrisesti sekoittamalla näytteitä tai vaihtamalla niiden väritystä [30]. Tässä tutkielmassa ei toteutettu datan augmentointia tutkielman laajuuden ja datan haastavuuden takia.

Koneoppimisen ja tekoälyn hyödyntäminen lääketieteessä on hyvin ajankohtainen tutkimusala. Artikkeliin [5] on koottu kattavasti eri artikkeleissa julkaistuja tuloksia, jotka keskittyvät koneoppimismallien käyttöön sairauksien diagnosoinnissa eri lääketieteen aloilla. Myös konvoluutioneuroverkkojen käyttö kasvainten identifioinnissa on mainittu artikkelissa. Koneoppimista voidaan käyttää diagnoosien tukena, mutta se ei pysty kokonaan korvaamaan ihmistä. Lääketieteen alan ammattilaiset ovat edelleen keskeisessä roolissa sekä potilastyössä että uusien ja epäselvien tapausten diagnosoinnissa.

Viitteet

[1] Charu C. Aggarwal. *Neural networks and deep learning : a textbook*. Springer, Cham, second edition edition, 2023.

[2] Ethem Alpaydin. *Introduction to machine learning*. Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, fourth edition. edition, 2020.

[3] Gurvinder Singh Bumbrah and Rakesh Mohan Sharma. Raman spectroscopy – basic principle, instrumentation and selected applications for the characterization of drugs of abuse. *Egyptian journal of forensic sciences*, 6(3):209–215, 2016.

[4] Michele Ceccarelli, Sofie R. Salama, Ganiraju Manyam, Pietro Zoppoli, Mia Grifford, Arvind Rao, Ching C. Lau, Jason Huse, Siyuan Zheng, Kenneth Hess, Ganesh Rao, Rameen Beroukhim, Rehan Akbani, Margaret Wrensch, David Haussler, Kenneth D. Aldape, Samreen Anjum, Saianand Balu, Gene Barnett, Keith L. Black, Christopher A. Bristow, Qing-Rong Chen, Juok Cho, Eric Chuah, Sudha Chudamani, Mark L. Cohen, Marta Couce, Tanja Davidsen, Amy Davis, J. Bradley Elder, Jennifer M. Eschbacher, Ashley Fehrenbach, Martin Ferguson, Gregory Fuller, Jordonna Fulop, Luciano Garofano, Julie M. Gastier-Foster, Nils Gehlenborg, Mark Gerken, William J. Gibson, Angela Hadjipanayis, D. Neil Hayes, Joe Hilty, Alan P. Hoyle, Corbin D. Jones, Steven J.M. Jones, Zhenlin Ju, Alison Kastl, Ady Kendler, Raju Kucherlapati, Tara M. Lichtenberg, Pei Lin, Julia Y. Ljubimova, Yiling Lu, Yussanne Ma, Dennis T. Maglinte, Marco A. Marra, Shaowu Meng, C. Ryan Miller, Gordon B. Mills, Lisle E. Mose, Theresa Naska, Michael S. Noble, Ardene Noss, Cheryl Palmer, Michael Parfenov, Joel S. Parker, Charles M. Perou, Todd Pihl, Alexei Protopopov, Nilsa C. Ramirez, Gordon Saksena, Jacqueline E. Schein, Hui Shen, Yan Shi, Kristen Shimmel, Hugues Sicotte, Suzanne Sifri, Tiago Silva, Rosy Singh, Xingzhi Song, Camila Souza, Huandong Sun, Charlie Sun, Jiabin Tang, Felipe Amstalden Trevisan, David J. Van Den Berg, Doug Voet, Zhining Wang, John N. Weinstein, Daniel J. Weisenberger, Matthew D. Wilkerson, Felicia Williams, Lisa Wise, Andrew W. Xu, Lixing Yang, Travis I. Zack, Jianhua Zhang, Wei Zhang, and Erik Zmuda. Molecular profiling reveals biologically discrete subsets and pathways of progression in diffuse glioma. *Cell*, 164(3):550–563, 2016.

[5] G Jignesh Chowdary, Suganya G, Premalatha M, Asnath Victy Phamila Y, and Karunamurthy K. Machine learning and deep learning methods for building intelligent systems in medicine and drug discovery: A comprehensive survey, 2021.

[6] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark, 2022.

[7] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Knowledge Discovery and Data Mining*, 1996.

[8] Xiaqiong Fan, Wen Ming, Huitao Zeng, Zhimin Zhang, and Hongmei Lu. Deep learning-based component identification for the raman spectra of mixtures. *Analyst (London)*, 144(5):1789–1798, 2019.

[9] Mesfin Fassil B., Karsonovich Torin, and Al-Dhahir Mohammed A. *Gliomas*. StatPearls Publishing, Treasure Island (FL), 2024.

[10] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[11] Alejandra M. Fuentes, Apurva Narayan, Kirsty Milligan, Julian J. Lum, Alex G. Brolo, Jeffrey L. Andrews, and Andrew Jirasek. Raman spectroscopy and convolutional neural networks for monitoring biochemical radiation response in breast tumour xenografts. *Scientific reports*, 13(1):1530–12, 2023.

[12] Aurelien. Geron. *Hands-on machine learning with Scikit-learn, Keras, and TensorFlow : concepts, tools, and techniques to build intelligent systems*. O'Reilly, Sebastopol, 2nd ed edition, 2019 - 2019.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts ;, 2016.

[14] Joonas Haapasalo, Antti Hyartt, Minja Salmi, Kristiina Nordfors, Sirpa Liisa Lahtela, Marketta Kähkönen, Pauli Helén, and Hannu Haapasalo. Glioomien diagnoosi ja ennuste–molekyylidiagnostiikan mahdollisuudet, 2014.

[15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[16] Chen Kong and Simon Lucey. Take it in your stride: Do we need striding in cnns? 2017.

[17] Miroslav Kubat. *An Introduction to Machine Learning*. Springer International Publishing, Cham, 2nd ed. 2017. edition, 2017.

[18] Adrian Lita, Joel Sjöberg, David Păcioianu, Nicoleta Siminea, Orieta Celiku, Tyrone Dowdy, Andrei Păun, Mark R Gilbert, Houtan Noushmehr, Ion Petre, and Mioara Larion. Raman-based machine-learning platform reveals unique metabolic differences between idhmut and idhwt glioma. *Neuro-oncology (Charlottesville, Va.)*, 26(11):1994–2009, 2024.

[19] Adrian Lita, Joel Sjöberg, David Păcioianu, Nicoleta Siminea, Orieta Celiku, Andrei Păun, Mark R Gilbert, Houtan Noushmehr, Ion Petre, and Mioara Larion. Tumor detection in glioma samples using raman spectroscopy and unsupervised machine learning.

[20] Jinchao Liu, Margarita Osadchy, Lorna Ashton, Michael Foster, Christopher J Solomon, and Stuart J Gibson. Deep convolutional neural networks for raman spectrum recognition: a unified solution. *Analyst (London)*, 142(21):467–474, 2017.

[21] Farhad Maleki, Nikesh Muthukrishnan, Katie Ovens, Caroline Reinhold, and Reza Forghani. Machine learning algorithm validation: From essentials to advanced applications and implications for regulatory certification and deployment. *Neuroimaging clinics of North America*, 30(4):433–445, 2020.

[22] Soumita Modak. Finding groups in data: an introduction to cluster analysis finding groups in data: an introduction to cluster analysis , authored by leonard kaufman and peter j. rousseeuw, john wiley and sons, 2005, isbn: 0-47-1-73578-7: authored by leonard kaufman and peter j. rousseeuw, john wiley and sons, 2005, isbn: 0-47-1-73578-7. *Journal of applied statistics*, 51(8):1618–1620, 2024.

[23] Mehryar. Mohri, Afshin. Rostamizadeh, and Ameet. Talwalkar. *Foundations of machine learning*. Adaptive computation and machine learning series. MIT Press, Cambridge, MA, 2012.

[24] Hanna Mäenpää. Gliooma kuriin. *Duodecim*, 126(14):1669–1675, 2010.

[25] Mikael Mäkelä. Neural network training notebook. `https://github.com/MrMake1/CNN-for-Glioma-classification`, 2025.

[26] Malila Nea. Aikuisten keskushermoston pahanlaatuiset kasvaimet suomessa. *Duodecim*, 136(10):1199–1201, 2020.

[27] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. Accessed: 2025-04-27.

[28] Andrea Orlando, Filippo Franceschini, Cristian Muscas, Solomiya Pidkova, Mattia Bartoli, Massimo Rovere, and Alberto Tagliaferro. A comprehensive review on raman spectroscopy applications. *Chemosensors*, 9(9), 2021.

[29] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[30] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

[31] Meshal Shutaywi and Nezamoddin N. Kachouie. Silhouette analysis for performance evaluation in machine learning with applications to clustering. *Entropy (Basel, Switzerland)*, 23(6):759–, 2021.

[32] Joel Sjöberg, Nicoleta Siminea, Andrei Păun, Adrian Lita, Mioara Larion, and Ion Petre. Radar: Raman spectral analysis using deep learning for artifact removal.

[33] Alejandro Antonio Torres Garcia. *Biosignal processing and classification using computational learning and intelligence : principles, algorithms, and applications*. Academic Press, London, England, 2022.

[34] Olli Tynninen, Hannu Haapasalo, Soili Kytölä, and Anders Paetau. Pahanlaatuisuusaste ei yksin riitä – molekyylitutkimukset tarkentavat glioomien diagnostiikkaa. *Duodecim*, 136(10):1202–1208, 2020.

[35] Liqiang Yu, Xinyu Zhao, Jiaxin Huang, Hao Hu, and Bo Liu. Research on machine learning with algorithms and development. *Journal of Theory and Practice of Engineering Science*, 3:7–14, 12 2023.

[36] Zhi-Min Zhang, Shan Chen, and Yi-Zeng Liang. Baseline correction using adaptive iteratively reweighted penalized least squares. *Analyst (London)*, 135(5):1138–1146, 2010.