TURUN YLIOPISTO

Konvoluutioneuroverkkojen rakenne ja sovelluksia

Matematiikan
pro gradu -tutkielma

Laatija:
Teemu Sarikka

16.4.2025
Turku

Pro gradu -tutkielma

Tutkinto-ohjelma, oppiaine: Matematiikka
Tekijä: Teemu Sarikka
Otsikko: Konvoluutioneuroverkkojen rakenne ja sovellukset
Ohjaaja: Professori Ion Petre
Sivumäärä: 47 sivua
Päivämäärä: 16.4.2025

Tekoäly jaetaan usein kahteen erilaiseen tekoälyn muotoon, heikkoon tekoälyyn ja vahvaan tekoälyyn. Heikkoa tekoälyä ovat sellaiset tekoälyn muodot, joissa tekoäly pyrkii ratkaisemaan jotain yksittäistä ongelmaa ja kaikki tämänhetkiset tekoälytratkaisut ovat heikkoa tekoälyä. Vahvalla tekoälyllä tarkoitetaan sellaisia tekoälyn muotoja, joissa tekoäly pystyy suorittamaan minkä tahansa sellaisen älyllisen tehtävän, jonka ihminenkin pystyy suorittamaan.

Koneoppiminen on tekoälyn osa-alue, jossa pyritään kehittämään tilastollisia menetelmiä, joiden avulla tietokoneohjelma voidaan opettaa suorittamaan tehtäviä perustuen ohjelman saamaan aiempaan dataan. Koneoppimisalgoritmit luokitellaan yleisesti niille annettavan syötetiedon sisältämien ominaisuuksien sekä niistä saatavien tulosteiden luonteen perusteelle ohjatun oppimisen, ohjaamattoman oppimisen, osittain ohjatun oppimisen sekä vahvistusoppimisen algoritmeihin.

Syväoppiminen puolestaan on koneoppimisen osa-alue, jossa käsitellään joukkoa sellaisia koneoppimismenetelmiä, joissa hyödynnetään monikerroksisia neuroverkkoja. Monikerroksiset neuroverkot ovat nimensä mukaisesti neuroverkkoja, jotka koostuvat useista peräkkäisistä kerroksista toisiinsa yhdistettyjä keinotekoisia neuroneita. Syväoppivista neuroverkoista on kehitetty useita toisistaan rakenteellisesti erovia versioita, joilla on keskenään erilaisia etuja ja niitä käytetään erilaisiin sovelluskohteisiin. Yleisimpiä ja tunnetuimpia syväoppivia neuroverkkoja ovat eteenpäin kytketyt neuroverkot, takaisinkytketyt neuroverkot, konvoluutioneuroverkot sekä modernit transformer-arkkitehtuuriin perustuvat neuroverkot.

Tässä tutkielmassa käsitellään syväoppivia- ja erityisesti konvoluutioneuroverkkoja, niiden rakennetta ja sovelluksia. Tutkielmassa käydään läpi tekoälyn, koneoppimisen ja syväoppimisen perusteita, neuroverkkojen perusrakennetta ja neuroverkkojen kouluttamista. Tutkielmassa keskitytään hieman perusteellisemmin käsittelemään konvoluutioneuroverkkojen rakennetta ja lisäksi tutkielmassa käydään läpi esimerkki, jossa datajoukon luokitteluun rakennetaan konvoluutioneuroverkkomalli, monikerrosperseptroni sekä lisäksi datajoukon avulla koulutetaan logistinen regressioluokittelija ja XGBoost -malli, joiden tuottamia datan ennustetarkkuuksia verrataan koulutettujen syväoppimismallien ennustetarkkuuksiin.

Avainsanat: Syväoppiminen, Konvoluutioneuroverkot

Sisällysluettelo

1 Johdanto

Tekoäly, koneoppiminen ja syväoppiminen ovat termejä, jotka vilisevät nykypäivänä hyvin usein vastaan mitä moninaisimmissa lähteissä ja tilanteissa. Monelle nämä termit ovat enemmän tai vähemmän synonyymejä keskenään ja se, mitä niillä tarkkaan ottaen tarkoitetaan, on usein hieman epäselvää. Tekoälyn vallatessa mediatilaa ja erilaisten kuvan- ja tekstintunnistussovellusten yleistyessä myös termi neuroverkko on tullut monille tutuksi.

Tekoäly, koneoppiminen, syväoppiminen ja keinotekoiset neuroverkot eivät kuitenkaan ole välttämättä kovinkaan yksiselitteisiä asioita ja tämän tutkielman tarkoitus on käydä lyhyesti läpi, mitä koneoppiminen ja syväoppiminen oikeastaan tarkoittavat sekä esittää neuroverkkojen rakennetta ja toimintaperiaatteita yleisellä tasolla.

Tutkielmassa käsitellään hieman tarkemmin erityisesti kuvan- sekä tekstintunnistukseen usein sovellettavaa konvoluutioneuroverkkoteknologiaa ja käydään läpi lyhyt esimerkkisovellus, missä avoimen datajoukon avulla koulutetaan tiedon arviointiin soveltuva konvoluutioneuroverkkomalli.

2 Tekoälyn perusteita

Tekoäly sanana ja konseptina ymmärretään mitä moninaisimmilla tavoilla ja tekoälylle on muodostettu useita erilaisia määritelmiä toisistaan poikkeavien tahojen ymmärtäessä tekoälyn käsitteen hieman eri tavalla, lähestyessä sitä hieman erilaisesta näkökulmasta tai huomioiden siihen sisältyvän erilaisia asioita. Euroopan Unioni on määritellyt tekoälyn seuraavan kaltaisen määritelmän mukaisesti: ”Tekoälyllä tarkoitetaan koneen kykyä käyttää perinteisesti ihmisen älyyn liitettyjä taitoja, kuten päättelyä, oppimista, suunnittelemista tai luomista [1].” Kansainvälinen standardisoimisjärjestö ISO puolestaan on määritellyt tekoälyn vapaasti suomennettuna jotakuinkin seuraavasti: ”Tekoäly on tekniikan- ja tieteenala, joka käsittelee sellaisia järjestelmiä, mitkä tuottavat sisältöä, ennusteita, suosituksia tai päätöksiä tiettyyn ihmisten määrittelemään tarkoitukseen tai kohteeseen [2].”

Julkisoikeudellisten toimijoiden lisäksi myös tekoälyä kehittävät ja liiketoiminnassaan hyödyntävät suuret teknologiayritykset ovat luoneet omia määritelmiään tekoälylle. IBM määrittelee tekoälyn seuraavasti: ”Tekoäly on teknologiaa, joka mahdollistaa sen, että tietokoneet ja laitteet kykenevät jäljittelemään ihmisen oppimista, ymmärrystä, ongelman ratkaisu- ja päätöksentekokykyä, luovuttaa ja autonomiaa [3].” Amazon puolestaan antaa tekoälylle seuraavan määritelmän: ”Tekoäly on teknologiaa, jolla on ihmismäinen kyky ratkaista ongelmia. Tekoäly vaikuttaa kykenevän jäljittelemään ihmisen älykkyyttä – se kykenee tunnistamaan kuvia, kirjoittamaan runoja ja tekemään tietoon perustuvia päätöksiä [4].” Edellä esitellyt tekoälyn määritelmät poikkeavat jonkin verran toisistaan, mutta yhteistä kaikille edellä esitellyille, ja muillekin tekoälyä koskeville määritelmille on kuitenkin se, että kaikissa niissä tekoälyn määritellään enemmän tai vähemmän olevan koneelle opetettu tai koneen oppima kyky jäljitellä ihmisen ajattelua ja käytöstä. Itse menetelmien, mallien ja algoritmien käsittelyn lisäksi tekoäly konseptina tai määreenä kattaa muitakin osa-alueita, kuten esimerkiksi tekoälyyn liittyvät eettiset, yhteiskunnalliset ja muun muassa lainsäädännölliset kysymykset.

Tekoäly teknologiana jaetaan usein kahteen erilaiseen tekoälyn muotoon: heikko tekoäly ja vahva tekoäly. Heikoksi tekoälyksi luokitellaan kaikki sellaiset tekoälyn muodot, joissa tekoäly pyrkii ratkaisemaan jotain yksittäistä ongelmaa tai suorittamaan yksittäistä kapeaa tehtävää. Käytännössä kaikki tämänhetkiset teknologiset tekoälytratkaisut ovat luokiteltavissa heikoksi tekoälyksi. Vahvalla tekoälyllä puolestaan tarkoitetaan sellaisia, toistaiseksi

hypoteettisia, tekoälyn muotoja, joissa tekoäly pystyy suorittamaan minkä tahansa sellaisen älyllisen tehtävän, jonka ihminenkin pystyy suorittamaan [5].

2.1 Koneoppiminen

Koneoppiminen käsitteenä sisältyy tekoälyn käsitteeseen. Koneoppiminen on tekoälyn osa-alue, jossa pyritään kehittämään tilastollisia menetelmiä ja – algoritmeja, joiden avulla tietokoneohjelma voidaan opettaa suorittamaan tehtäviä perustuen ohjelman saamaan aiempaan dataan. Koneoppimisalgoritmit luokitellaan yleisesti niille annettavan syötetiedon sisältämien ominaisuuksien sekä niistä saatavien tulosteiden luonteen perusteella seuraaviin luokkiin: ohjattu oppiminen, ohjaamaton oppiminen, osittain ohjattu oppiminen sekä vahvistusoppiminen [6].

Ohjatun oppimisen luokka koostuu sellaisista koneoppimisalgoritmeista, joiden syötetiedot sisältävät tiedon alkioiden luokasta ja näiden luokkatietojen avulla algoritmi pyrkii muodostamaan funktion, jonka avulla malli kykenee tuottamaan ennusteita tulosteiden luokista. Olennainen osa ohjatun oppimisen algoritmeja on siis nimenomaan se, että syötetieto sisältää tiedot alkioiden luokista, joita algoritmin ja sen avulla muodostetun koneoppimismallin opetuksessa hyödynnetään [7].

Ohjatun oppimisen luokka jaetaan edelleen yleisesti kahden eri kategorian algoritmeihin: luokittelualgoritmeihin ja regressioalgoritmeihin. Luokittelualgoritmeja käytetään niissä tilanteissa, kun tulosteiden halutaan saavan jonkinlaisen kategorisen luokituksen eli, että tulosteet voidaan luokitella kahteen tai useampaan toisistaan erilliseen luokkaan [7]. Eräs esimerkki luokittelualgoritmista on sähköpostin roskapostisuodatin, jossa ohjatun oppimisen malli koulutetaan syötedatan avulla ennustamaan tai arvioimaan onko saapuva sähköposti roskapostia vai ei. Regressioalgoritmeista puhutaan puolestaan silloin, kun algoritmien tuottamat ennusteet ovat luonteeltaan reaalisia tai jatkuvia [7]. Esimerkki regressioalgoritmista on esimerkiksi algoritmi, joka ennustaa työntekijän saaman palkan suhdetta työntekijän kokemukseen.

Ohjatun oppimisen algoritmien avulla voidaan luoda yksinkertaisista monimutkaisiin vaihtelevia malleja, joiden avulla voidaan tehdä ennusteita suuresta määrästä erilaisia asioita ja tämän vuoksi niitä käytetään hyvin laajalti eri aloilla ja erilaisissa käyttökohteissa, kuten esimerkiksi terveydenhuollossa, markkinoinnissa ja finanssimarkkinoilla [7]. Yleisesti käytettyjä ohjatun oppimisen algoritmeja ovat mm. lineaarinen regressio, polynominen

regressio, polynominen regressio, K lähimmän naapurin algoritmi, Naiivi Bayes-algoritmi ja päätöspuut [8].

Ohjaamattomaksi oppimiseksi luokitellaan puolestaan sellaisia koneoppimisalgoritmeja, joissa algoritmi oppii sille syötetystä aineistosta ilman, että syötettävä data sisältää tietoja aineiston luokista. Algoritmista muodostettu malli opetetaan jakamaan aineisto luokkiin siten, että kunkin luokan alkiot muistuttavat enemmän toisiaan kuin muiden luokkien alkioita [9].

Kuten ohjatun oppimisen algoritmit, myös ohjaamattoman oppimisen algoritmit voidaan jakaa edelleen eri kategorian algoritmeihin, esimerkiksi klusterointialgoritmeihin sekä ulottuvuuden vähentämisalgoritmeihin [6].

Klusterointi on tekniikka, jonka avulla pyritään ryhmittelemään datajoukko eri ryhmiin perustuen datajoukon alkioiden keskinäiseen samankaltaisuuteen, joka klusteroinnin yhteydessä tarkoittaa etäisyyttä, esimerkiksi euklidista etäisyyttä. Klusterointi on yleisesti käytetyimpiä ohjaamattoman oppimisen menetelmiä ja sitä käytetään useissa eri käyttökohteissa, muun muassa asiakasdatan ryhmittelyssä, erilaisissa petosten havaintasovelluksissa sekä esimerkiksi kuva-analyyseissa [9].

Klusterointiin käytetään useampaa erilaista klusterointialgoritmia, muun muassa poissulkeva klusterointi (*exclusive*), päällekkäisklusterointi (*overlapping*), hierarkkinen klusterointi (*hierarchical*) ja todennäköisyysklusterointi (*probabilistic*). Poissulkevassa klusteroinnissa syötetieto ryhmitellään siten, että yksittäinen datapiste sijaitsee vain ja ainoastaan yhdessä ryhmässä. Poissulkeva klusterointialgoritmi on esimerkiksi K-means klusterointialgoritmi, joka ryhmittelee syötetiedon käyttäjän määrittelemään K määrään ryhmiä. Päällekkäisklusteroinnissa yksittäinen syötetiedon datapiste voi puolestaan kuulua useampaan kuin vain yhteen ryhmään kerrallaan. Hierarkkisessa klusteroinnissa syötetieto ensin ryhmitellään yksittäisten datapisteiden samankaltaisuuden mukaan ja sen jälkeen datajoukkoja yhdistetään perustuen datajoukkojen hierarkkisiin yhtäläisyyksiin. Todennäköisyysklusteroinnissa syötetieto puolestaan ryhmitellään perustuen todennäköisyyksiin, millä kukin datapiste kuuluu mihinkäkin joukkoon [9].

Ulottuvuuden vähentämisalgoritmit puolestaan ovat ohjaamattoman oppimisen tekniikka, jossa pyritään poistamaan syötetietojoukosta epärelevantteja luokkia, eli ulottuvuuksia, ja sitä kautta löytämään syötetiedosta luokittelun kannalta olennaisimmat ominaisuudet [9].

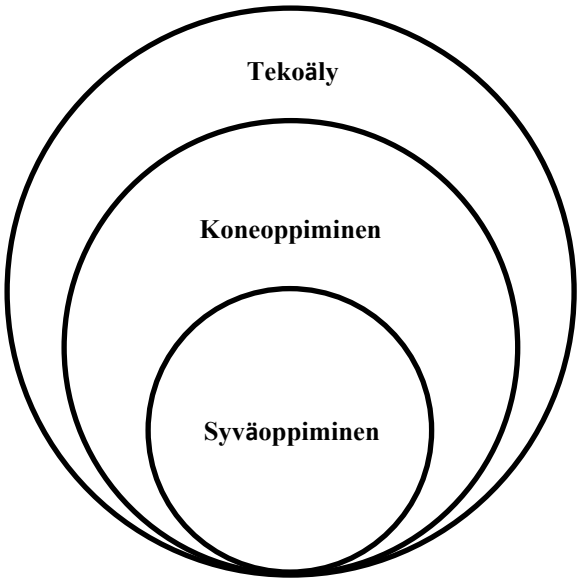
Erilaiset ohjaamattoman oppimisen algoritmit ovat tyypillisesti hyvin käyttökelpoisia tilanteissa, joissa syötetieto on monimutkaista ja se pitää saada esimerkiksi ryhmiteltyä tai tiedosta pitää löytää tiettyjä toistuvuuksia tai olennaisia luokkia vaikkapa myöhempää luokittelua varten.

Osittain ohjattu oppiminen on joukko koneoppimisalgoritmeja, joissa yhdistyy elementtejä ja ominaisuuksia sekä ohjatun oppimisen että ohjaamattoman oppimisen algoritmeista. Osittain ohjatun oppimisen algoritmeja sovelletaan yleisesti sellaisissa tilanteissa, missä haluttaisiin muuten soveltaa ohjatun oppimisen algoritmeja, mutta saatavilla olevasta tiedosta merkittävä osa on sellaista, joka ei sisällä erillistä luokkatietoa esimerkiksi sen vuoksi, että sen saaminen olisi todella hankalaa tai hyvin kallista. Mikäli merkittävä osa tiedosta ei sisällä luokkatietoa ja samanaikaisesti halutaan koneoppimismalli, jonka tarkoitus on pystyä muodostamaan tulosteena esimerkiksi ennuste, ei ohjatun oppimisen tai ohjaamattoman oppimisen algoritmi pystyisi yksinään tällaista tehtävää toteuttamaan [10].

Vahvistusoppiminen on koneoppimistekniikka, jossa algoritmia opetetaan tekemään päätöksiä, joiden avulla se saavuttaa itsensä kannalta mahdollisimman optimaalisen lopputuloksen. Vahvistusoppiminen jäljittelee erehdyksen kautta oppimisen prosessia, jota ihmisaivot käyttävät saavuttaakseen oppimistavoitteensa. Päätöksiä, joita tekemällä algoritmi lähestyy toivottua lopputulosta, vahvistetaan ja toisaalta ne päätökset, jotka aiheuttavat algoritmin päätyvän kauemmas toivotusta lopputuloksesta, jätetään huomiotta. Näin toimimalla algoritmi saadaan oppimaan jokaisesta tekemästään päätöksestä saamastaan palautteesta ja saadaan sitä kautta löytämään paras mahdollinen tapa saavuttaa haluttu lopputulos. Vahvistusoppimisalgoritmeja käytetään yleisesti tilanteissa, joissa algoritmin pitää pystyä toimimaan monimutkaisessa ja ennalta tuntemattomassa toimintaympäristössä [11].

2.2 Syväoppiminen

Syväoppiminen puolestaan on koneoppimisen osa-alue, josta puhuttaessa tarkoitetaan joukkoa sellaisia koneoppimismenetelmiä, joissa hyödynnetään monikerroksisia neuroverkkoja [6]. Syväoppiminen eroaa laajemmasta koneoppimisen termistä siinä mielessä, että syväoppimismenetelmissä käytetään monimutkaisempia rakenteita ja kerroksia, jotka pystyvät käsittelemään ja analysoimaan hyvinkin suuria määriä tietoa itsenäisesti sen jälkeen, kun ne on koulutettu. Koneoppiminen ja syväoppiminen ovat siis molemmat tekoälymenetelmiä ja kuvassa 1 on pyritty havainnollistamaan tekoälyn, koneoppimisen ja syväoppimisen keskinäistä suhdetta toisiinsa.



Kuva 1. Tekoälyn, koneoppimisen ja syväoppimisen suhde toisiinsa.

Idea ja perusperiaate monikerroksisiin neuroverkkoihin ja sitä kautta syväoppimisen tekniikkaan on saatu biologiasta ja neurotieteistä siten, että on keksitty pyrkiä replikoimaan tai toistamaan aivojen toiminnan periaatteita keinotekoisesti luomalla keinotekoisia neuroneita, verkottamalla niitä keskenään, pinoamalla useita neuroverkko päällekkäin kerroksiksi ja opettamalla näin muodostettua rakennetta käsittelemään tietoa [12]. Tämän työn aihe, konvoluutioneuroverkot, liittyy nimenomaisesti syväoppimistekniikoihin, joten tästä eteenpäin työssä keskitytään lähinnä syväoppimistekniikoiden käsittelyyn ja seuraavassa kappaleessa käydään läpi keinotekoisten neuroverkkojen rakennetta ja periaatteita.

3 Keinotekoiset neuroverkot

Keinotekoisten neuroverkkojen idea on mallintaa ihmisaivojen toimintaa ja niiden tieteellisen historian voidaan katsoa alkaneen vuodesta 1943, kun Warren McCullough ja Walter Pitts esittelivät ensimmäisen keinotekoisen neuronin matemaattisen mallin [13]. Artikkelissaan McCullough ja Pitts pyrkivät selittämään ihmisaivojen kykyä tuottaa monimutkaisia päätelmiä käyttäen hyväksi toisiinsa yhdistyneitä aivosoluja eli neuroneita ja eräs tästä työstä seuranneista päätelmistä liittyi binäärisen aktivaation ymmärtämiseen [12].

Useamman neuronin yhdistelmänä tehdyn keinotekoisen neuroverkon esitteli dokumentoidusti ensimmäisen kerran Frank Rosenblatt vuonna 1958 [14]. Rosenblatt vei McCullochin ja Pittsin ajatuksia asteen eteenpäin tuomalla mukaan neuronien väliset painot ja hän sai tietokoneen oppimaan erottelemaan reikäkortteja toisistaan reiän sijainnin perusteella [12]. Neuroverkkoja tutkittiin tästä eteenpäin aktiivisesti sekä neurotieteiden että tietojenkäsittelytieteen aloilla aina 1960 luvun lopulle asti, jolloin neuroverkkoihin liittyvän tutkimuksen suosio lopahti lähinnä yksikerroksisten neuroverkkojen rajallisuuden vuoksi [15].

Neuroverkkotutkimus heräsi uudelleen henkiin 1980 luvulla ja useat tutkijat saivat samanaikaisesti ideoita takaisinkytkennän käsitteestä ja Yhdysvalloissa Harvadin yliopistossa väitöskirjantekijä Paul Werbos yhdisti takaisinkytkennän käsitteen neuroverkkoihin vuonna 1974 [12], [16]. Edelleen vuonna 1989 Yann LeCun osoitti, kuinka takaisinkytkentää hyödyntävä neuroverkkoarkkitehtuuri pystyttiin opettamaan tunnistamaan käsin kirjoitettuja postinumeroita [12], [17]. 2000 luvun alun jälkeen neuroverkkotutkimuksen suosio on edelleen noussut valtaisasti johtuen paljolti näytönohjainten kehittyneiden grafiikkasuorittimien mukanaan tuomasta huomasta laskentatehon kasvusta [15].

3.1 Keinotekoisten neuroverkkojen rakenne

Yksinkertaisin ja yksi varhaisimmista keinotekoisesta neuroverkosta on nimeltään perseptroni, joka on käytännössä yksinkertainen neuroni aktivaatiofunktionaan alkujaan porrasfunktio, esimerkiksi

$$f(x) = \begin{cases} 1 & kun\ x \geq 0 \\ 0 & kun\ x < 0 \end{cases}.$$

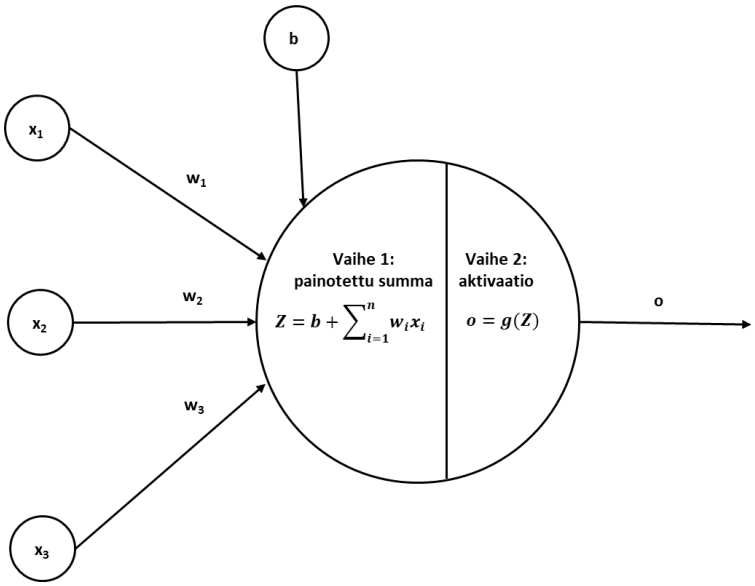
Moderneissa neuroverkoissa aktivaatiofunktio ei useinkaan enää ole porrasfunktio, mutta muutoin perseptronin käsite on alkuperäisen kaltainen. Perseptronin rakenne on esitetty kuvassa 2. Jokainen perseptroni koostuu syötteistä (x1 – xn), painoista (kuvassa w1 – wn) sekä vakioisesta virhetermistä (b). Perseptronin toimiessa kukin syöte kerrotaan sille määrätyllä painolla ja nämä tulot sekä virhetermi summataan yhteen:

$$Z = \sum_{i=1}^{n} w_i x_i + b.$$

Tämän jälkeen perseptronin tuloste määrittyy summan ja aktivaatiofunktion funktiona

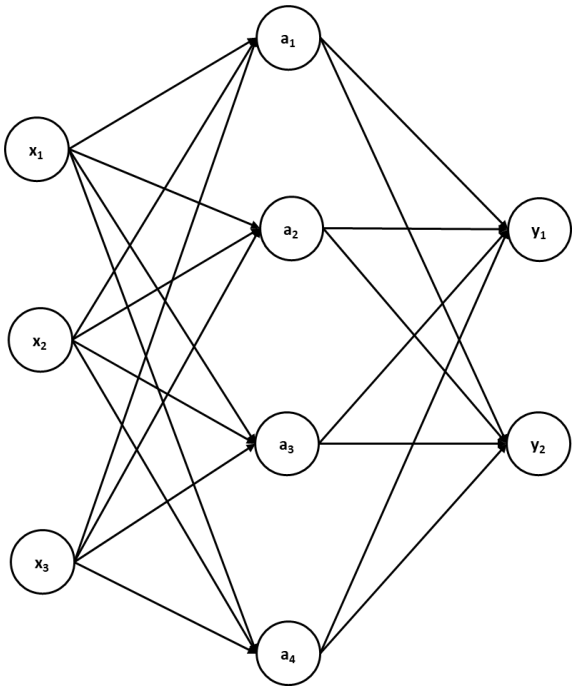
$$tuloste = f(Z)$$

ja perseptroni syöttää tulosteen edelleen eteenpäin [18].

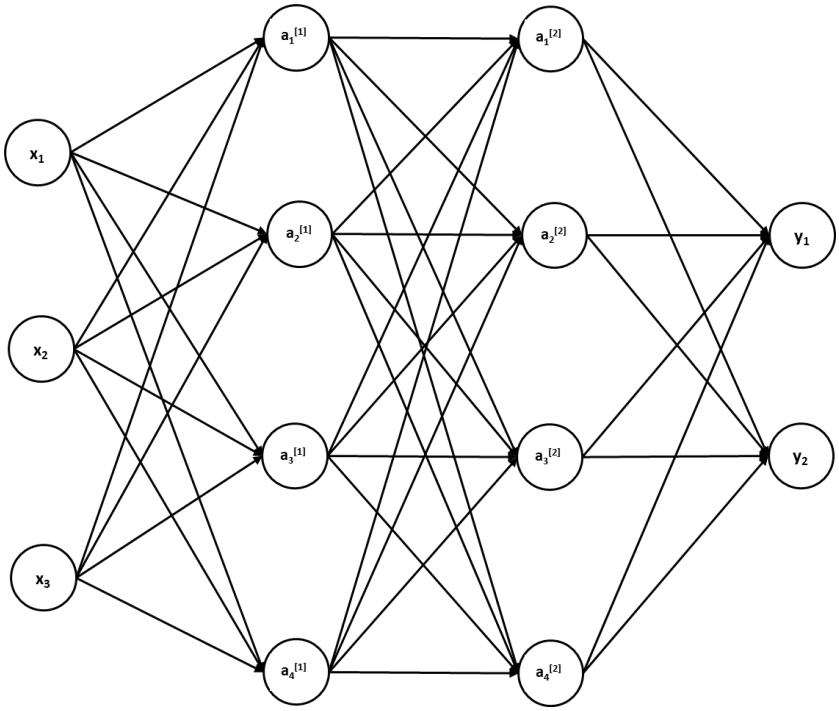


Kuva 2. Perseptronin rakenne [18]. Perseptroni kertoo saamansa syöteet painoilla, jonka jälkeen aktivaatiofunktio määrittää perseptronin tulosteen.

Kun perseptroneja laitetaan monta vierekkäin samaan kerrokseen, saadaan yksikerroksinen neuroverkko, jonka rakenne on esitetty Kuvassa 3. Kun edelleen jatketaan lisäämällä toinen kerros perseptroneja ensimmäisen kerroksen perään saadaan aikaiseksi yksinkertainen monikerroksinen neuroverkko, monikerrosperseptroni. Monikerroksisen neuroverkon periaatteellinen rakenne on esitetty kuvassa 4.



Kuva 3. Yksikerroksisen neuroverkon rakenne [18]. Neuroverkko koostuu syötekerroksesta ja tulostekerroksesta ja niiden välissä olevasta neurokerroksesta.



Kuva 4. Monikerroksisen neuroverkon rakenne [18]. Neuroverkko koostuu syötekerroksesta, tulostekerroksesta ja niiden välissä olevista useammista piilokerroksista, joita tässä tapauksessa on kaksi.

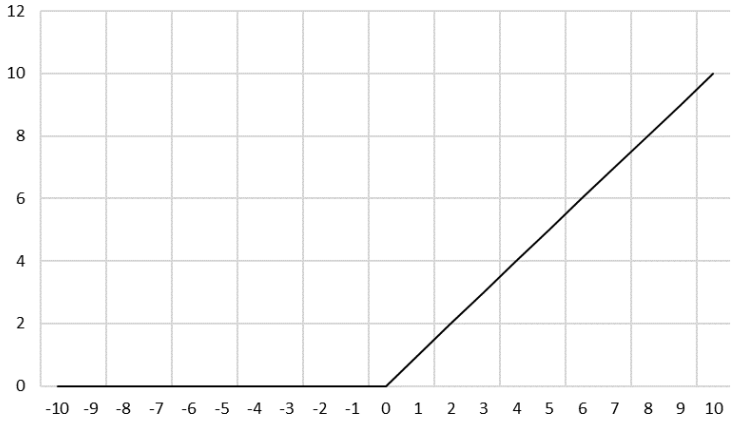
Syväoppiminen käsitteenä liittyy olennaisesti neuroverkkoihin ja syväoppimisella tarkoitetaankin joukkoa tekoälymenetelmiä, joissa käytetään arkkitehtuuriltaan erilaisia monikerroksisia neuroverkkoja. Neuroverkkoa, jossa on yhteensä enemmän kuin kolme kerrosta, voidaan pitää syväoppimisalgoritmina [12].

3.2 Aktivaatiofunktiot

Erittäin olennainen osa neuroverkon arkkitehtuuria on aktivaatiofunktio, jonka tehtävä on määrittää neuroverkon neuroneiden aktivoituminen ja määrittää aktivoituneiden neuroneiden tuottamat tulosteet. Vaihtoehtoisia aktivaatiofunktioita on useita ja valinta eri vaihtoehtojen välillä riippuu pääasiassa neuroverkon tyypistä, neuronikerroksen tyypistä ja sovelluskohteesta. Yksinkertaisimmillaan aktivaatiofunktio voi olla jokin lineaarinen funktio.

Eräs suosituimmista ja parhaita tuloksia tuottaneista aktivaatiofunktioista moderneissa neuroverkoissa on nimeltään ReLU-funktio (*rectified linear unit*) [19]. ReLU-funktion hyödyntämisen neuroverkon aktivaatiofunktiona esitteli ensimmäisenä Kunihiko Fukushima vuonna 1969 artikkelissaan "Visual feature extraction by a multilayered network of analog threshold elements" [20], [21], [22]. ReLU-funktio määritellään matemaattisesti seuraavasti ja funktion kuvaaja havainnollistetaan kuvassa 5:

$$ReLU(x) = \max(0, x).$$



Kuva 5. ReLU-funktion kuvaaja. Negatiivisilla syötteillä x funktion arvo on 0 ja positiivisilla syötteillä x funktion arvo on x.

ReLU-funktio on yksinkertainen ja laskennallisesti tehokas. Tavallisesta ReLU-funktiosta on sen suosion seurauksena kehitetty erilaisia aktivaatiofunktiomuunnelmia, kuten esimerkiksi LReLU (*leaky ReLU*) [23] ja PReLU (*parametric ReLU*) [24]. LReLU-funktio määritellään matemaattisesti seuraavalla tavalla

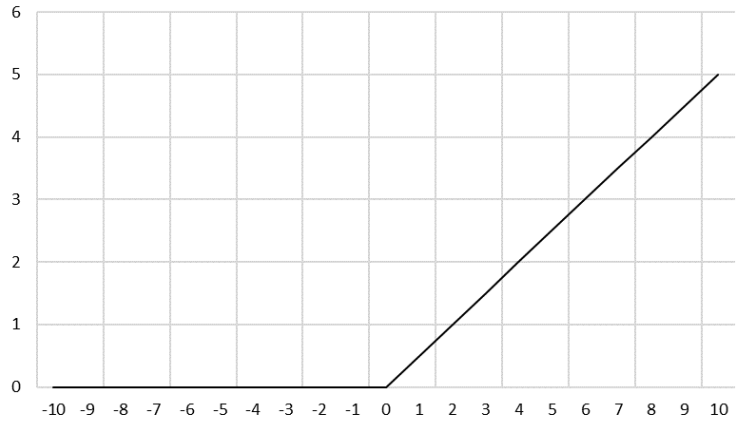
$$LReLU(x) = \max(\alpha x, x), \quad jossa\ \alpha\ on\ vakio$$

PReLU-funktio

$$PReLU(x) = \max(\alpha x, x), \quad jossa\ \alpha\ määritetään$$

kouluttamisen yhteydessä

ja LReLU- ja PReLU-funktioiden kuvaajat arvolla α = 0,5 on havainnollistettu kuvassa 6.

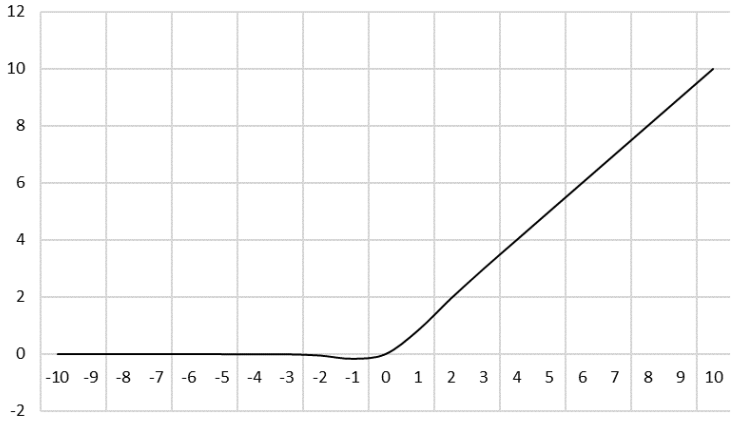


Kuva 6. LReLU- ja PReLU-funktioiden kuvaajat arvolla α = 0,5. Negatiivisilla syötteillä x funktion arvo on 0 ja positiivisilla syötteillä x funktion arvo on αx.

Eli toisin kuin alkuperäinen ReLU-funktio, LReLU ja PReLU-funktiot eivät nollaa neuronin tulostetta sen ollessa negatiivinen, vaan antavat sille alkuperäistä pienemmän negatiivisen arvon. ReLU-funktion idean pohjalta on kehitetty myös muita aktivaatiofunktiota, kuten ELU (*exponential linear unit*) [25], SELU (*scaled exponential linear unit*) [26], Swish [19] sekä hyvin moderneissa neuroverkkoratkaisuissa aktivaatiofunktiona käytetty periaatteiltaan stokastisempi GELU-funktio (*gaussian error linear unit*) [27], jonka matemaattinen määritelmä puolestaan on seuraava:

$$GELU(x) = xP(X \leq x) = x\Phi(x) = x \cdot \frac{1}{2}\left[1 + \tanh\left(\sqrt{\frac{2}{\pi}} \cdot (x + 0.044715x^3)\right)\right].$$

GELU-funktion on todettu toimivan erinomaisesti esimerkiksi moderneissa transformer-mallisissa neuroverkoissa [28], mutta se on toisaalta laskennallisesti vähemmän tehokas kuin yksinkertaisemmat ReLU-funktiot. GELU-funktion kuvaaja on esitetty kuvassa 7.



Kuva 7. GELU-funktion kuvaaja. GELU-funktion kuvaaja on hyvin samankaltainen kuin ReLU-funktion kuvaaja sillä merkityksellisellä erolla, että origon läheisyydessä GELU-funktion arvot poikkeavat ReLU-funktion arvoista.

ReLU-pohjaisten funktioiden lisäksi hyvin yleisesti neuroverkoissa aktivaatiofunktiona käytetään niin kutsuttua Softmax-funktiota [29], jonka määritellään seuraavasti:

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}.$$

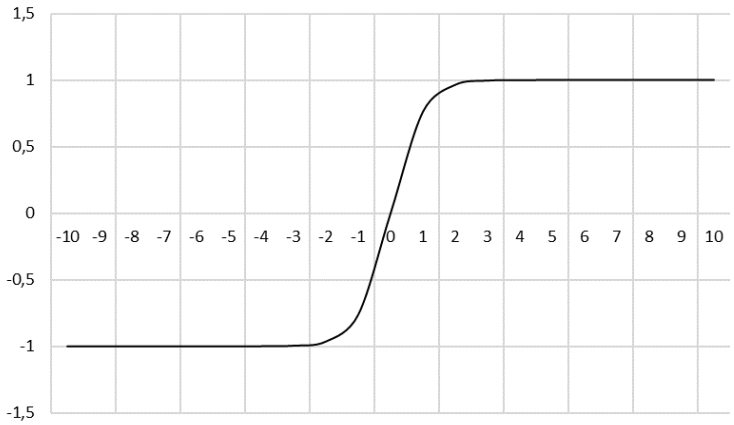
Softmax-funktio siis kääntää vektorimuotoisen syötteen todennäköisyysjakaumaksi ja mahdollistaa näin erityyppisten luokitteluongelmien ratkaisun. Neuroverkoissa on käytetty aktivaatiofunktiona myös sigmoid-funktiota

$$\sigma(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

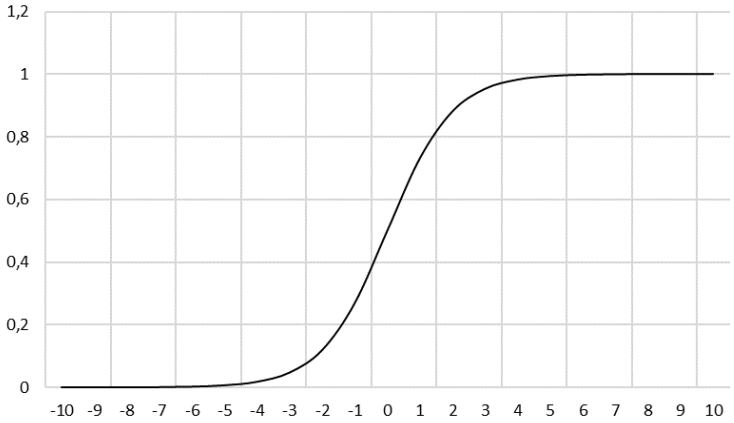
sekä hyperbolista tangenttifunktiota

$$tanh(x) = \frac{1}{1 + e^{-x}},$$

ja niiden kuvaajat on esitetty kuvissa 8 ja 9.



Kuva 8. Sigmoid-funktion kuvaaja. Sigmoid-funktion arvo vaihtelee -1 ja 1 välillä eroten -1 ja 1 merkittävästi lähellä origoa.



Kuva 9. Hyperbolisen tangenttifunktion kuvaaja. Hyperbolisen tangenttifunktion arvo vaihtelee 0 ja 1 välillä eroten -1 ja 1 merkittävästi lähellä origoa.

Sigmoid-funktion ja hyperbolisen tangenttifunktion käyttö on kuitenkin vähentynyt merkittävästi, koska nämä aktivaatiofunktiot kärsivät niin kutsutusta katoavan gradientin ongelmasta.

3.3 Kouluttaminen

Neuroverkkoarkkitehtuurin, eli neuronikerrosten tyypin ja määrän sekä aktivaatiofunktioiden lisäksi neuroverkot tarvitsevat myös oikein asetetut parametrit, käytännössä siis neuronien painot, joiden avulla neuroverkot pystyvät ratkaisemaan niille annettuja ongelmia ja tuottamaan esimerkiksi haluttuja ennusteita tai luokitteluja. Prosessia, jolla neuroverkon neuronien painot etsitään, kutsutaan neuroverkon kouluttamiseksi. Neuroverkon kouluttaminen on luonteeltaan iteratiivinen prosessi, jossa neuroverkolle syötetään koulutusdataa sykli kerrallaan ja neuroverkon parametrit lasketaan kunkin syklin yhteydessä uudelleen [30].

3.3.1 Virhefunktio

Neuroverkon kouluttaminen perustuu siihen, että neuroverkon tekemä virhe pyritään saamaan mahdollisimman pieneksi. Toisin sanoen pyritään siihen, että neuroverkon tekemän arvion ja todellisen vastaan arvon välinen ero, eli neuroverkon virhe, on mahdollisimman pieni. Tämä tapahtuu minimoimalla neuroverkolle asetettua virhefunktiota, joka voi olla esimerkiksi pienimmän neliösumman funktio [31], [32]:

$$J(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2.$$

Neuroverkkojen koulutuksessa yleisesti käytettyjä virhefunktioita ovat keskineliövirhe (*MSE*), keski-itseisvirhe (*MAE*), keski-itseisvirhe prosentteina ja Huber-virhe (*Huber loss*), joita käytetään yleisesti regressiosovelluksiin liittyvien neuroverkkojen koulutuksessa. Binääristä luokittelua vaativissa sovelluksissa käytetään usein binääristä ristientropiafunktiota (*binary cross-entropy*) ja moniluokkaisissa luokittelusovelluksissa puolestaan moniluokkaista ristientropiafunktiota (*multi-class cross-entropy*) [31].

Eräs hyvin keskeinen menetelmä virhefunktion minimoimiseksi on niin kutsuttu gradient descent algoritmi. Gradient descent algoritmi löytää ratkaisun moneen koneoppimisongelmaan nopeammin kuin satunnaiset arvaukset ja on merkittävästi tehokkaampi kuin esimerkiksi lineaarinen regressio. Gradient descent algoritmin keskeisenä ideana on se, että funktion käyttö aloitetaan syöttämällä funktiolle satunnaiset arviot syöteparametreista. Tämän jälkeen selvitetään, kumpaan suuntaan virhefunktio laskee enemmän, kun parametrejä muutetaan hieman. Sitten kaikkia optimoitavia parametreja muutetaan hieman siihen suuntaan, johon virhefunktion lasku oli suurin ja tätä toistetaan niin kauan, kunnes saavutetaan hyväksyttävä lopputulos, eli käytännössä kunnes on löydetty funktion lokaali tai globaali minimi [32]. Matemaattisesti gradient descent algoritmi toimii seuraavasti:

$$\theta = \theta - \alpha \nabla J(\theta),$$

jossa θ on optimoitava parametrivektori, J(θ) virhefunktio, ∇J(θ) virhefunktion gradientti ja α parametri oppimisnopeudelle. Gradient descent algoritmin käyttö aloitetaan arvaamalla parametrien arvot ja tämän jälkeen iteroimalla gradient descent algoritmia niin kauan, kunnes virhefunktion minimi saavutetaan [33]. Oppimisnopeusparametrin suuruus määrittelee nopeuden, jolla gradient descent algoritmi konvergoituu kohti virhefunktion minimiä, mutta

toisaalta liian suurella oppimisparametrin arvolla voidaan ajautua tilanteisiin, joissa algoritmi ei konvergoidu.

3.3.2 Vastavirta-algoritmi

Virhefunktion gradienttia lasketaan syväoppimissovelluksissa usein vastavirta-algoritmin (*backpropagation algorithm*) avulla [34]. Vastavirta-algoritmi on käytännössä tekniikka, jonka avulla esimerkiksi gradient descent algoritmia sovelletaan. Vastavirta-algoritmissa virhefunktion gradientti lasketaan rekursiivisesti tulostekerroksesta syötekerrokseen, eli neuroverkon vastavirtaan [35].

Vastavirta-algortmi on niin ikään iteratiivinen tekniikka, jossa virhefunktiota minimoidaan muuttamalla neuroverkon neuroneiden painoja ja mahdollisia lisättäviä vakiotermejä. Neuroverkon painoja muutetaan jokaisella syklillä virhefunktion gradientin laskevaan suuntaan ja algoritmi laskee virhefunktion gradienttia rekursiivisesti tulostekerroksesta syötekerrokseen ketjusäännön avulla.

Vastavirta-algoritmi toimii käytännössä niin, että aluksi neuroverkko vastaanottaa syötteen ja laskee sen eteenpäin pitkin neuroverkon kerroksia neuroverkon läpi. Kunkin kerroksen neuronit laskevat omat tulosteensa käyttäen painoja ja aktivaatiofunktioitaan. Sitten neuroverkon tulostetta verrataan oikeaan vastaukseen käyttämällä virhefunktiota, esimerkiksi keskineliövirhettä, jonka avulla mitataan tulosteen ja todellisen arvon ero. Tämän jälkeen virhe levitetään rekursiivisesti neuroverkon läpi takaisin neuroverkon syötekerrokseen ja kunkin neurokerroksen neuroneiden painoja muutetaan osittaisderivaattojen painojen suhteen siten, että neuroverkon tekemä virhe pienenee.

Näin kun virheen ja osittaisderivaattojen perusteella tiedetään, mitä ja miten neuroverkon painoja tulee säätää, painoja muutetaan edellisessä kappaleessa esitetyn gradient descent -algoritmin mukaisesti. Tätä prosessia toistamalla useita kertoja peräkkäin (*epoch*) koko koulutusaineiston läpi saadaan neuroverkkomallin painot säädettyä sellaisiksi, että neuroverkon tekemä virhe on mahdollisimman pieni [34].

Moderneissa neuroverkkosovelluksissa verkon kouluttamiseen käytetään traditionaalisen gradient descent algoritmin sijaan usein modernimpaa stokastista gradient descent (*stochastic gradient descent*) [36] tai Adam (*adaptive moment estimation*) -algoritmia. Stokastinen gradient descent algoritmi eroaa perinteisestä gradient descent algoritmista siten, että traditionaalista gradient descent algoritmia käytettäessä kukin peräkkäinen koulutuskierros

tehdään käyttäen kaikkea opetusdataa kerrallaan, mutta stokastisessa gradient descent algoritmissa kullakin iteraatiokierroksella valitaan pienempi osuus opetusdatasta satunnaisesti. Tämä eroavaisuus aiheuttaa sen, että stokastinen gradient descent algoritmi toimii nopeammin ja tehokkaammin, mutta voi olla joissain tilanteissa hieman epävakaampi. Lisäksi stokastisella gradient descent algoritmilla voidaan päästä pois funktion paikallisista minimeistä, koska algoritmin päivityskierroksissa on mukana satunnaisuutta [36]. Adam-algoritmi puolestaan on stokastisesta gradient descent algoritmista edelleen kehittyneempi versio, jossa gradientin suuntaa seurataan yli ajan ja jokaiselle verkon painolle lasketaan erikseen oppimisnopeus sen perusteella, kuinka paljon kunkin painon arvo on vaihdellut verkon koulutuksen edetessä [37].

3.3.3 Yli- ja alisovittaminen

Neuroverkkojen kouluttamiseen liittyy oleellisesti kysymys siitä, kuinka paljon neuroverkkomallia tulisi kouluttaa. Mikäli neuroverkkomallia ylikoulutetaan oppimaan opetusdatasta ominaisuuksia, jotka eivät liity datan sisältöön, kuten esimerkiksi datan sisältämää kohinaa tai poikkeavia taikka virheellisiä datapisteitä, neuroverkkomallin kyky analysoida vastaavaa muuta dataa heikkenee merkittävästi. Niin ikään, mikäli malli opetetaan sovittamaan tarpeettoman monimutkainen matemaattinen funktio kuvaamaan opetusdataa, joka todellisuudessa käyttäytyy huomattavasti yksinkertaisemmin, neuroverkkomallin kyky mallintaa kyseistä tapahtumaa heikkenee merkittävästi.

Edellä kuvattua ongelmaa kutsutaan ylisovittamiseksi, mikä käytännössä tarkoittaa sitä, että neuroverkkomalli oppii opetuksessa käytetyn datan ominaisuudet niin hyvin, että mallin suorituskyky muun vastaavan geneerisen datan arvioinnissa kärsii tästä. Yleisimpiä koneoppimismallin ylisovittamisen syitä ovat mallin ennusteiden turhan korkea varianssi ja matala bias, opetusdatan liian pieni määrä sekä mallin rakenteellisuuden turha monimutkaisuus [38].

Ylisovittamisen vastakohtana voidaan puolestaan pitää mallin alisovittamista. Alisovittamista tapahtuu silloin, kun koulutettu malli on liian yksinkertainen kuvaamaan dataa, kuten esimerkiksi tilanteessa, jossa eksponentiaalisesti käyttäytyvää dataa pyritään kuvaamaan lineaarisena. Päinvastoin kuin ylisovittavassa mallissa, alisovittavan mallin ennusteiden bias on turhan korkea ja varianssi liian matala. Mallin sisältämä alisovittaminen voi puolestaan olla seurausta esimerkiksi mallin liiasta rakenteellisesta yksinkertaisuudesta tai liian pienestä opetusdatan määrästä [38].

3.4 Eteenpäin kytketyt neuroverkot

Eteenpäin kytketyt neuroverkot ovat arkkitehtuuriltaan yksinkertaisimpia syväoppivia neuroverkkoja. Eteenpäin kytketyissä neuroverkoissa on syötekerros, jota seuraa vaihtelevan sisältöinen piilotettujen kerrosten sarja, minkä jälkeen neuroverkkoarkkitehtuurin lopussa on tulostekerros. Eteenpäin kytketyissä neuroverkoissa jokainen edeltävän kerroksen neuroni on kytketty jokaiseen seuraavan kerroksen neuroniin, eli kerrokset ovat täysin yhdistettyjä kerroksia, ja data kulkee yksisuuntaisesti muuttuen syötteestä tulosteeksi kulkiessaan neuronikerrosten läpi. Huolimatta siitä, että eteenpäin kytketyt neuroverkot ovat arkkitehtuuriltaan yksinkertaisimpia neuroverkkoja, syöte- ja tulostekerrosten välissä olevat piilokerrosrakenteet voivat silti olla huomattavan monimutkaisia. Eteenpäin kytkettyjä neuroverkkoja voidaan käyttää monenlaisiin käyttötarkoituksiin, esimerkiksi hahmon- ja kuvantunnistukseen, regressioanalyysiin sekä luokitteluun [6].

3.5 Takaisinkytketyt neuroverkot

Takaisinkytketyissä neuroverkoissa kunkin neuronikerroksen tulostetta voidaan käyttää verkkoarkkitehtuurissa aiempana olevien neuronikerrosten syötteenä. Toisin sanoen, toisin kuin eteenpäin kytketyissä neuroverkoissa, takaisinkytketyissä neuroverkoissa data kulkee kaksisuuntaisesti neuronikerrosten välillä, sekä eteen että taakse päin [6], [39].

Takaisinkytkettyjen neuroverkkojen keskeinen piirre on takaisinkytkentä, joka mahdollistaa sen, että neuroverkko muistaa edellisen tilansa ja hyödyntää sitä kyseistä tilaa seuraavien laskelmien yhteydessä. Tämä antaa neuroverkolle kyvyn käsitellä sekvenssejä ja säilyttää informaatiota aikaisemmista vaiheista. Takaisinkytkettyjä neuroverkkoja käytetäänkin usein tehtävissä, joissa tiedon aikarakenne tai sekvenssi on tärkeä.

Takaisinkytketyt neuroverkot ikään kuin hyödyntävät muistiaan antaessaan nykyisen syötetiedon lisäksi aiemman syötetiedon ja aiemman tulosteen vaikuttaa nykyiseen tulosteeseen. Traditionaaliset, eteenpäin kytketyt neuroverkot olettavat syötetiedon ja tulosteen olevan riippumattomia toisistaan, mutta takaisinkytkettyihin neuroverkkoihin tätä riippumattomuuden oletusta ei sisälly. Takaisinkytkettyjä neuroverkkoja käytetään usein luonnollisen kielen- ja puheentunnistussovelluksissa sen vuoksi, että takaisinkytketyt neuroverkot ovat hyvin käyttökelpoisia tilanteissa, jossa syötetieto on jaksoittaista taikka aikasarjamuotoista [40].

3.6 Generatiiviset kilpailevat verkot

Generatiiviset kilpailevat verkot (*generative adversarial networks, GAN*) ovat moderni neuroverkkoarkkitehtuuri, joissa kaksi erillistä neuroverkkoa kilpailevat toisiaan vastaan siten, että ensimmäinen neuroverkoista (generatiivinen verkko, *generator*) pyrkii luomaan syötedatan avulla autenttisia datapisteitä ja toinen neuroverkko (diskriminoiva verkko, *discriminator*) pyrkii erottelemaan ensimmäisen neuroverkon luomat datapisteet aidoista syötedatan datapisteistä. Tätä iterointia jatketaan niin kauan, kunnes toinen neuroverkko ei enää kykene erottamaan ensimmäisen neuroverkon luomia datapisteitä aidoista syötedatan datapisteistä.

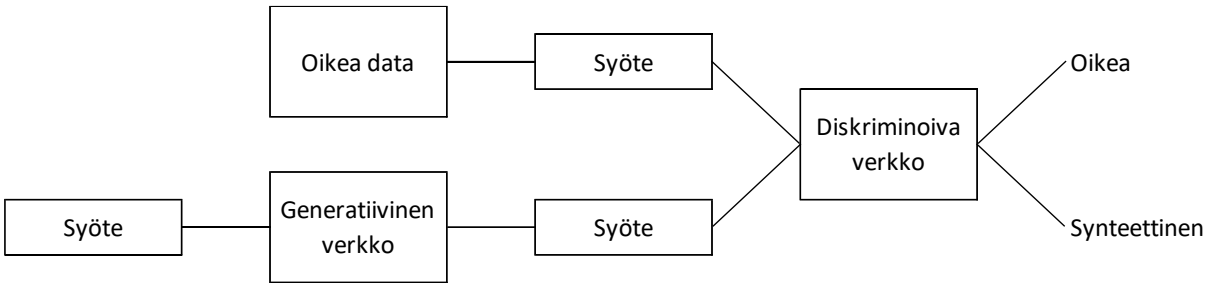
Käytännössä generatiivisen kilpailevan verkon toiminta tapahtuu seuraavan prosessin mukaisesti. Ensin generatiivinen verkko ottaa satunnaisen syötteen ja tuottaa sen perusteella väärennetyn tulosteen. Diskriminoiva verkko saa syötteenään sekä todellista dataa että generatiivisen verkon luomia tulosteita, jotka se yrittää edelleen erotella toisistaan. Näin molemmat verkot oppivat ja parantavat suoritustaan jatkuvasti siten, että generoiva verkko oppii luomaan yhä realistisempaa dataa, jota on vaikeampi erottaa todellisesta datasta ja diskriminoiva verkko puolestaan oppii yhä tarkemmin erottelemaan generatiivisen verkon luomia tulosteita todellisesta datasta. Tätä prosessia jatketaan edelleen niin kauan, kunnes generatiivisen verkon luoma data on niin realistista, että diskriminoiva verkko ei enää osaa erotella sitä todellisesta datasta.

Matemaattisesti generatiivisen kilpailevan verkon arkkitehtuuri perustuu peliteoriaan ja sen kouluttaminen koostuu virhefunktion minimoinnista ja peliteoreettisesta optimoinnista. Generatiivisen kilpailevan verkon virhefunktio voidaan matemaattisesti esittää seuraavasti [41]:

$$\min_G \max_D V(D,G) = E_{x\sim p_x(x)}[logD(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))],$$

missä $p_x(x)$ on todellisen syötedatan jakauma, $p_z(z)$ on generatiivisen verkon tuottaman datan jakauma, D(x) on diskriminoivan verkon arvio siitä, onko x oikeaa vai generatiivisen verkon tuottamaa dataa ja G(z) generatiivisen verkon tuottama data. Virhefunktiossa siis generatiivinen funktio pyrkii maksimoimaan omaa virhettään ja diskriminoiva funktio pyrkii minimoimaan omaa virhettään, joten diskriminoiva verkko päivittää omia painojaan maksimoidakseen V(D,G) funktion ja generatiivinen verkko vastaavasti päivittää omia painojaan minimoidakseen V(D,G) funktion. Näin ollen koulutuksen onnistuessa

optimaalisesti päädytään tilanteeseen, jossa D(G(z)) ≈ 0,5, jolloin generatiivinen verkko on oppinut tuottamaan niin realistista dataa, että diskriminoiva verkko ei enää osaa tehdä eroa sen ja todellisen datan välille [41]. Generatiivisen kilpailevan verkon rakenne on esitetty kuvassa 10.



Kuva 10. Generatiivisen kilpailevan verkon rakenne. Generatiivinen verkko luo syötteen diskriminoivalle verkolle ja diskriminoiva verkko pyrkii erottamaan generatiivisen verkon luoman syötteen oikeasta datasta.

Generatiivisten kilpailevien verkkojen käyttömahdollisuudet ovat hyvin laajat ja niiden avulla voidaan muun muassa luoda kolmiulotteisia malleja kaksiulotteisista kuvista, tuottaa erilaisia keinotekoisia kuvia, luoda syntetisoituja ääniä ja musiikkia tai luoda datajoukkoja toisten neuroverkkojen käyttöön [42], [43].

3.7 Transformer-arkkitehtuuri

Transformer-arkkitehtuuri on tässä käsitellyistä syväoppivista neuroverkkoarkkitehtuureista modernein ja sen esittelivät Googlen tutkijat artikkelissaan vuonna 2017. Transformer-arkkitehtuuri on suunniteltu käsittelemään erityisesti sekvenssidataa, kuten tekstiä, ja se on tullut tunnetuksi erityisesti kielenkäsittelyn ja konekääntämisen alueilla. Transformer-arkkitehtuuri koostuu kahdesta pääosasta, enkooderista (*encoder*) ja dekooderista (*decoder*). Molemmat näistä ovat monivaiheisia ja koostuvat useista neurokerroksista, mutta eroavat toisistaan toiminnallisesti [28].

Transformer-arkkitehtuurissa enkooderi ottaa syötteen ja muuttaa syötteenä saamansa tekstin numeeriseen muotoon vektoriksi. Dekooderi ottaa enkooderilta saamansa matemaattisen esityksen ja pyrkii jatkamaan vektoria sekä kääntää vektorin takaisin tekstimuotoon. Enkooderi-dekooderi rakenteen lisäksi olennainen osa transformer-arkkitehtuuria ovat niin kutsutut itsehuomio (*self-attention*) ja tarkkaavaisuusmekanismi (*multi-head attention mechanism*), joita hyödyntämällä dekooderin muodostamista useista mahdollisista vektorinjatkeista painotetaan ja valitaan sopivin sen perusteella, kuinka paljon kontekstia kukin jatke antaa ympärillään oleville sanoille [28].

Matemaattisesti transformer-arkkitehtuuri keskeinen on edellä mainittu itsehuomiomekanismi. Itsehuomiomekanismin tarkoituksena on laskea syötteen osien välille huomioarvo (*attention score*), joka lasketaan seuraavasti [28]:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V,$$

jossa Q on kysymysvektori, joka kuvaa mitä mallin tulee etsiä syötteestä, K on avainvektori, joka kuvaa syötteen osan merkitystä ja V on arvovektori, joka sisältää syötteen osan tiedot. QK^T on kysymys- ja avainvektoreista koostuva tekijä, joka kuvaa kysymyksen avaimen samanlaisuutta suhteessa toisiinsa ja d_k on avainvektorin pituus, joka normalisoi ja vakauttaa laskentaa. Huomioarvo siis kertoo sen, kuinka paljon jokainen avainvektori, eli syöte, vaikuttaa arvovektoriin, eli lopputulokseen. Kun tämä toimenpide tehdään edelleen kaikille koko syötteen osille, voidaan koko syöte arvioida ja syötteen osien suhteet toisiensa suhteen laskea.

Transformer-arkkitehtuuria kehitettiin pääasiassa luonnollisen kielen käsittelyyn, mutta sitä on sovellettu erittäin menestyksekkäästi muissakin tekoälysovelluksissa kuten esimerkiksi konenäköön liittyvissä sovelluksissa sen tehokkuuden, skaalautuvuuden ja kyvyn käsitellä pitkän matkan riippuvuuksia vuoksi. Transformer-arkkitehtuurin keskeinen innovaatio on tarkkaavaisuusmekanismi, joka mahdollistaa syötteen osien välisten suhteiden tehokkaan mallintamisen. Tämä arkkitehtuuri on perustana monille nykyaikaisille kielimalleille, kuten GPT, BERT ja T5. [44].

3.8 Konvoluutioneuroverkot

Tavallisia, täysin yhdistetyistä kerroksista koostuvia neuroverkkomalleja käytettäessä ennen pitkää törmätään kuitenkin ennen kaikkea erääseen ongelmaan. Neuroverkkomalliin sisältyvien painojen lukumäärä kasvaa nopeasti erittäin suureksi ja painojen oppimiseen tarvittavan datan määrä kasvaa niin valtavaksi, että sellaisen käsittely vaatii kohtuuttoman määrän laskentatehoa ja toisaalta sellaisia määriä dataa ei välttämättä ole saatavilla tai edes olemassa [45]. Tämän ongelman ratkaisuksi on löydetty konvoluutioneuroverkot. Konvoluutioneuroverkot ovat hyviä tunnistamaan kuvioita ja muotoja, jonka vuoksi ne ovat tärkeässä roolissa esimerkiksi tämänhetkisissä tietokonenäköön liittyvissä tekoälyratkaisuissa.

Konvoluutioneuroverkon muista neuroverkkoarkkitehtuureista erottaa pääasiassa se, että konvoluutioneuroverkon neuronit jakavat painoja sekä harha-arvoja muiden verkon neuronien

kanssa, toisin kuin neuronit eteenpäin kytketyissä sekä takaisinkytketyissä neuroverkoissa. Neuroneilla on keskenään samoja painoja sen vuoksi, että kunkin neuronin tarkoitus on suorittaa samaa tehtävää, kuten esimerkiksi tunnistaa jonkin muodon reunaa, eripuolella syötettä. Syöte ja tulostekerrosten lisäksi konvoluutioneuroverkoissa on vähintään kaksia erilaisia muista neuroverkoista poikkeavia konvoluutioneuroverkoille tyypillisiä kerroksia, konvoluutiokerroksia sekä yhdistämiskerroksia. Tämän työn keskeinen aihe on nimenomaan konvoluutioneuroverkot, joten niiden rakennetta ja ominaisuuksia käsitellään laajemmin seuraavissa kappaleissa.

4 Konvoluutioneuroverkot

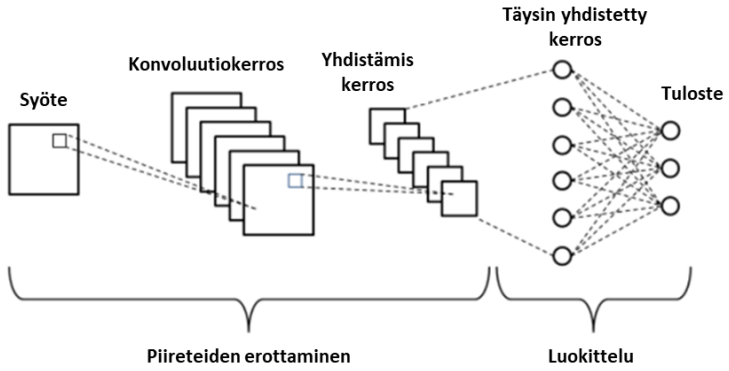
Koneoppimisen yhteydessä termillä konvoluutio tarkoitetaan tekniikkaa, jossa syötteenä käytettävä data syötetään konvoluutiokerroksen läpi, mikä muuttaa syötteen dimensioita. Koulutettaessa perinteistä kone-/syväoppimisalgoritmia, on sen koulutuksen yhteydessä opittava oma painonsa jokaiselle opetusdatan solulle. Esimerkkinä vaikkapa koneoppimisalgoritmin, joka opetetaan käyttäen 2 000 x 2 000 pikselin kuvia, täytyy sellaisenaan löytää 4 miljoonaa erillistä painoa, jotta se oppii tunnistamaan kyseisiä kuvia. Konvoluutioalgoritmin puolestaan on opittava painot jokaiselle konvoluutiodatan solulle. Ideana on siis toisin sanoen käyttää samoja painokertoimia monessa neuronissa, mutta eri syötteillä. Näin ollen konvoluutiosta saatava erittäin suuri hyöty on se, että konvoluution käyttö tekee koneoppimisalgoritmien käytöstä merkittävästi tehokkaampaa ja algoritmin opettamiseen vaadittavan datan määrä putoaa merkittävästi pienemmäksi kuin täysin yhdistetyssä tavallisessa neuroverkossa [45].

Konvoluutiokerrosten avulla voidaan tunnistaa erilaisia kuvapiirteitä, kuten erivärisiä muotoja, erisuuntaisia reunoja ja erilaisia kuvioita. Näiden kuvapiirteiden avulla voidaan edelleen tunnistaa abstraktimpia asioita, kuten esimerkiksi eläimiä tai liikennemerkkejä. Ilman konvoluutiokerroksia neuroverkon opettaminen tunnistamaan tällaisia asioita kuvasta olisi vaikeampaa ja enemmän aikaa ja tehoa vievää, sekä vaatisi huomattavan paljon enemmän opetusdataa. Koska tällaiset kohteet voivat sijaita missä tahansa päin kuvaa missä koossa tai kulmassa hyvänsä, kohteen tai kameran siirtäminen voi aiheuttaa tällaisten kohteiden sisältämien pikseliarvojen muuttumisen täysin erilaisiksi, vaikka kohde näyttää ihmisen silmään täysin samanlaiselta kuin ennenkin. Näin ollen tavallisen neuroverkon opettamiseen tarvitaan useita kuvia, joissa kukin tunnistettava kohde on kuvattu eripuolilla kuvaa sekä eri kuva kulmista. Konvoluutioneuroverkon avulla kohde voidaan tunnistaa kuvasta sen sijainnista riippumatta [46].

4.1 Konvoluutioneuroverkkojen rakenne

Kuten kaikki neuroverkot, konvoluutioneuroverkot koostuvat syötekerroksesta, vaihtelevasta määrästä piilokerroksia, sekä tulostekerroksesta. Piilokerrokset ja niiden kouluttaminen sisältävät kaikkien neuroverkkojen tapaan piilokerroksissa olevien neuronien painoja ja niiden optimoimista matriisitulojen avulla sekä neuronikerroksia seuraavien aktivaatiofunktioiden optimoimista [47]. Konvoluutioneuroverkon tapauksessa osa piilokerroksista, vähintään yksi,

mutta käytännön sovelluksissa aina useampi, on niin sanottuja konvoluutiokerroksia, eli kerroksia, joiden mukaan kyseinen neuroverkkoarkkitehtuuri on nimetty. Konvoluutiokerrosten lisäksi piilokerrokset sisältävät muita kerroksia, kuten yhdistämiskerroksia, normalisointikerroksia sekä täysin yhdistettyjä kerroksia. Kuvassa 11 on esitetty periaatekuva konvoluutioneuroverkon rakenteesta ja konvoluutioneuroverkossa esiintyvistä neurokerroksista.



Kuva 11. Konvoluutionneuroverkon rakenne [48]. Konvoluutioneuroverkko koostuu syöte- ja tulostekerroksista sekä vaihtelevasta määrästä konvoluutiokerroksia, yhdistämiskerroksia, normalisointikerroksia ja verkon lopussa olevia täysin yhdistettyjä kerroksia.

4.1.1 Konvoluutiokerrokset

Konvoluutiokerros on konvoluutioneuroverkon kerros, jossa varsinainen konvoluutio tapahtuu. Konvoluutioneuroverkoissa hyödynnetään matemaattisessa mielessä konvoluutio-operaatiota, joka voidaan matemaattisesti määritellä kahden jatkuvan reaalifunktion x ja y välille kaavalla [49]:

$$s(t) = (x * y)(t) = \int_{-\infty}^{\infty} x(a)y(t-a)da.$$

Koska käsiteltävä data on jatkuvan sijaan koneoppimisen yhteydessä diskreettiä, muuttuu konvoluution matemaattinen määritelmä konvoluutionneuroverkkojen tapauksessa diskreetiksi:

$$s(t) = (x * y)(t) = \sum_{a=-\infty}^{\infty} x(a)y(t-a).$$

Konvoluutiokerroksessa siis kerrotaan syötetietomatriisia konvoluutiosuotimella, joka on matriisi, millä on sama aste kuin syötematriisilla, mutta pienempi koko. Konvoluutiokerrosten

toiminta on yksinkertaista esittää esimerkin avulla. Ajatellaan seuraavaa 3x3 konvoluutiosuodinta:

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

Sovelletaan kyseistä konvoluutiosuodinta seuraavaan 4x4 syötematriisiin:

$$\begin{bmatrix} 5 & 5 & 2 & 1 \\ 3 & 9 & 8 & 5 \\ 0 & 1 & 4 & 2 \\ 1 & 5 & 9 & 2 \end{bmatrix}.$$

Kun konvoluutiosuodin asetetaan aluksi syötematriisin vasempaan yläkulmaan, syötematriisin konvoluutiosuotimen alle jäävät solut kerrotaan konvoluutiosuotimen vastaavilla soluilla ja tulot summataan yhdeksi skalaariksi, saadaan

$$1 * 5 + 0 * 5 + 1 * 2 + 0 * 3 + 1 * 9 + 0 * 8 + 1 * 0 + 0 * 1 + 1 * 4 = 20.$$

Konvoluutiosuodinta siirretään näin yksi askel kerrallaan ensin oikealle ja kun tullaan syötematriisin päätyyn, siirrytään yksi askel alapäin ja jatketaan näin koko syötematriisin läpi. Näin toimimalla saadaan tämän esimerkin mukaisesti neljän konvoluutio-operaation jälkeiseksi matriisiksi:

$$\begin{bmatrix} 20 & 17 \\ 22 & 25 \end{bmatrix}.$$

Edellä esitetty esimerkki on siis konvoluutioesimerkki käyttäen konvoluutiosuotimen kokoa 3x3, askelluksen (*stride*) arvoa 1 sekä zero-padding parametria, jolla tarkoitetaan sitä, että konvoluutiosuodin käy läpi syötematriisin aivan reunasta aivan toiseen reunaan. Konvoluutio-operaation lopputuloksena saatuja matriiseja kutsutaan ominaisuuskartoiksi.

4.1.2 Yhdistämiskerrokset

Konvoluutioneuroverkot sisältävät lähes aina konvoluutiokerrosten lisäksi yhdistämiskerroksia. Yhdistämiskerrosten ideana on vähentää datan ulottuvuutta yhdistämällä useamman neuronin tuloste edeltävältä neuronikerrokselta yhden neuronin syötteeksi seuraavassa neuronikerroksessa. Yhdistämiskerrokset voivat olla luonteeltaan joko lokaaleja- tai globaaleja yhdistämiskerroksia. Lokaalit yhdistämiskerrokset yhdistävät pienemmästä määrästä, tyypillisestä esimerkiksi 2x2, neuroneita tulevat tulosteet yhdeksi syötteeksi ja

globaalit yhdistämiskerrokset puolestaan yhdistävät koko ominaisuuskartan kaikkien neuronien tulosteet yhdeksi syötteeksi seuraavaan kerrokseen.

Konvoluutioneuroverkkojen yhdistämiskerroksessa käytettävä yhdistämistekniikka perustuu usein siihen, että yhdistämissuotimen alle jäävistä arvoista valitaan suurin (ns. max poolaus), mutta yhdistäminen voi perustua myös keskiarvoihin tai hieman monimutkaisempaan periaatteeseen, kuten esimerkiksi niin kutsuttuun Softmax-periaatteeseen.
Yhdistämiskerroksen toimintaa on myös helppo esitellä esimerkin avulla. Oletetaan jälleen sama 4x4 syötematriisi, kuin mitä käytettiin edellisen konvoluutioesimerkin tapauksessa:

$$\begin{bmatrix} 5 & 5 & 2 & 1 \\ 3 & 9 & 8 & 5 \\ 0 & 1 & 4 & 2 \\ 1 & 5 & 9 & 2 \end{bmatrix}.$$

Sovelletaan tähän syötematriisiin nyt yhdistämiskerroksen toimintaa käyttäen 2x2 yhdistämissuodinta sekä askellusarvoa 2. Tällöin yhdistämissuodin asetetaan jälleen matriisin vasempaan yläkulmaan ja sen alle jää seuraava osuus syötematriisista:

$$\begin{bmatrix} 5 & 5 \\ 3 & 9 \end{bmatrix}.$$

Kun yhdistämissuodinta siirretään tästä kaksi askelta kerrallaan ensin oikealle ja sitten alas, saadaan seuraavat matriisit:

$$\begin{bmatrix} 2 & 1 \\ 8 & 5 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 5 \end{bmatrix}, \begin{bmatrix} 4 & 2 \\ 9 & 2 \end{bmatrix}.$$

Oletetaan edelleen, että yhdistäminen perustuu suurimpaan suotimen alle jäävään arvoon ja näin saadaan yhdistämiskerroksen tulosteeksi seuraava matriisi:

$$\begin{bmatrix} 9 & 8 \\ 5 & 9 \end{bmatrix}.$$

4.1.3 Normalisointikerrokset

Normalisointikerrokset ovat neuronikerroksia, joiden tarkoitus on muokata syötetieto normalisointikerroksia seuraaville täysin yhdistetyille kerroksille sopivaksi. Tavalliset neuroverkot saavat syötetietonsa yleensä yksiulotteisena taulukkona (vektorina), mutta konvoluutioneuroverkon aiemmissa kerroksissa, konvoluutiokerroksissa ja yhdistämiskerroksissa, syötetieto on usein matriisimuodossa. Näin ollen

normalisointikerroksen tehtävä esimerkiksi kuvantunnistamiseen käytettävässä konvoluutioneuroverkossa on kääntää tieto matriisimuodosta vektorimuotoon.

4.1.4 Täysin yhdistetyt kerrokset

Täysin yhdistetyt kerrokset ovat järjestyksessään viimeinen osa konvoluutioneuroverkkoa ja ovat niin ikään erittäin tärkeä osa konvoluutioneuroverkkoarkkitehtuuria, sillä ne ovat neuroverkon osa, jossa itse oppiminen ja varsinainen luokittelu tapahtuu. Konvoluutioneuroverkon edeltävissä kerroksissa syötetietoa on käsitelty eri tavoin ja kuvista syötetiedosta on poimittu erilaisia tärkeitä tiedontunnistuksen kannalta olennaisia tiedon ominaisuuksia, mutta varsinainen tiedon tunnistaminen tai luokittelu tapahtuu täysin yhdistetyissä kerroksissa, josta tieto siirtyy edelleen tulostekerrokseen.

4.2 Aktivaatiofunktiot

Konvoluutioneuroverkkojen ja piilokerroksissa ja tulostekerroksessa käytetään usein hieman erityyppisiä aktivaatiofunktioita. Konvoluutiokerroksissa käytetään aktivaatiofunktiona hyvin usein ReLU-funktiota tai jotain sen modernimpaa muunnelmaa. ReLU-funktiota on käytetty esimerkiksi vuoden 2012 AlexNet konenäkömallissa [50] sekä vuoden 2015 ResNet (*residual neural network*) konenäkömallissa [51]. ReLU-funktiosta johdettua modernimpaa GELU:a käytetään esimerkiksi Googlen tutkijoiden kielentunnistusmallissa BERT:ssä (*bidirectional encoder representations from transformers*) [27].

ReLU-funktion tai jonkin sen muunnelman ollessa toimivia ratkaisuja piilokerrosten aktivaatiofunktioksi sen vuoksi, että ne ovat laskennallisesti tehokkaita ja niiden avulla neuroverkko kykenee oppimaan aineiston epälineaarisuuksia, tulostekerroksen aktivaatiofunktiolta vaaditaan erilaisia ominaisuuksia. Esimerkiksi luokittelua tekevässä neuroverkkomallissa on toivottavaa, että tulostekerros kykenisi esimerkiksi luokitteluun todennäköisyyksien avulla. Esimerkiksi AlexNet, ResNet sekä BERT mallien tulostekerroksissa käytettiin aktivaatiofunktiona Softmax funktiota [27], [50], [51].

4.3 Konvoluutioneuroverkkojen sovelluksia

Konvoluutioneuroverkkoja voidaan ominaisuuksiensa takia hyödyntää useissa tekoälyyn liittyvissä sovelluksissa. Varmaankin yleisin konvoluutioneuroverkkojen käyttökohde ovat erilaiset kuva- sekä videoanalytiikkaan liittyvät sovellukset, joissa konvoluutioneuroverkkoja on hyödynnetty erittäin laajasti jo pidemmän aikaa. Tällaisia sovelluksia ovat esimerkiksi

kuvantunnistus ja -luokittelusovellukset, objektintunnistus, kasvojentunnistus, videoanalyysi, kuvan manipulointi ja generointi, lääketieteellinen kuvantaminen, liikennemerkkientunnistus ja kuvahakusovellukset. Konvoluutioneuroverkkoja on hyödynnetty myös esimerkiksi kielentunnistussovelluksissa [52], peleissä [53], [54] ja muun muassa taloudellisissa aikasarja-analyysisovelluksissa [55].

Konvoluutioneuroverkot ovat erittäin tehokkaita erityisesti juuri kuvien ja muiden visuaalisten tietojen käsittelyssä, mutta niiden käyttöön liittyy myös useita rajoitteita. Konvoluutioneuroverkot toimivat hyvin vain euklidisen datan käsittelyssä, missä jokaisella datapisteellä on selkeä paikallinen sijainti. Sen sijaan epäeuklidisen datan käsittely konvoluutioneuroverkoilla on vaikeaa ja epäeuklidisen datan tapauksessa malliksi kannattaakin valita mieluummin jokin muu tekniikka. Konvoluutioneuroverkot ovat niin ikään perinteisesti herkkiä mallin ylisovittamiselle opetusdataan. Tämä johtuu konvoluutioneuroverkkojen monimutkaisesta rakenteesta ja niiden sisältämästä suuresta määrästä optimoitavia parametrejä, tosin sama pätee myös useille muille syväoppimismenetelmille. Ylisovittamisongelmaa pyritään konvoluutioneuroverkkojen tapauksessa hallitsemaan muun muassa riittävällä määrällä yhdistämiskerroksia, jotka yksinkertaistavat datan rakennetta mallissa ja mallin sisältämien parametrien määrää [56].

Konvoluutioneuroverkot voivat myös olla herkkiä syötedatan laadulle tai esimerkiksi pienille muutoksille syötedatassa. Sen seurauksena konvoluutioneuroverkot ovat alttiita adversiaarisille hyökkäyksille (*adversarial attack*), joissa tarkoituksena on syötedatan hienovaraisten muokkausten avulla saada tekoälymalli tuottamaan vääriä ennusteita. Konvoluutioneuroverkkojen antamien tulosten tulkinta ja neuroverkon toiminnan selittäminen voi myös olla osin vaikeaa. Tämä johtuu siitä, että konvoluutioneuroverkkojen sisältämien useiden suodattimien toiminnan sisäistäminen voi olla vaikeaa eikä aina kovin yksiselitteistä. Lisäksi konvoluutioneuroverkot tarvitsevat koulukseen erittäin suuria määriä dataa ja kouluttamiseen tarvitaan myös kohtuullisen paljon laskentatehoa, tosin tämä pätee myös useille muille syväoppimismenetelmille, ja useisiin muihin syväoppimismenetelmiin verrattuna konvoluutioneuroverkkojen kouluttaminen on erittäinkin tehokasta ja vähemmän opetusdataa vaativaa [57].

Konvoluutioneuroverkkojen, kuten myös yleisesti tekoälyn käyttöön erilaisissa sovelluksissa liittyy teknisten ja luonnontieteellisten kysymysten lisäksi myös useisiin eri muihin osa-alueisiin liittyviä olennaisia kysymyksiä. Eräänä esimerkkinä konvoluutioneuroverkkojen

kohdalla voidaan mainita oikeudelliset sekä eettiset kysymykset. Kuten edellä mainittu, konvoluutioneuroverkkoja käytetään paljon erilaisissa kuvantunnistus- ja konenäkösovelluksissa, kuten esimerkiksi kasvojentunnistuksessa, valvontajärjestelmissä tai vaikkapa lääketieteellisessä diagnostiikassa. Tällaisissa sovelluksissa oikeustieteellisiksi ja eettisiksi kysymyksiksi nousevatkin esimerkiksi kasvojentunnistuksen yhteydessä oikeus yksityisyyteen ja siihen liittyvät loukkaukset taikka erilaiset syrjintään liittyvät kysymykset. Lisäksi vastuu konvoluutioneuroverkkojen tekemistä virheistä tai esimerkiksi tekijänoikeuskysymykset ovat hankalia ja vaikeasti tulkittavia oikeudellisia konvoluutioneuroverkkojen käyttöön liittyviä kysymyksiä.

Konvoluutioneuroverkoista on kehitetty useita edelleen kehitettyjä erillisiä arkkitehtuuriversioita, joissa on omia erityisiä ominaisuuksiaan, kuten esimerkiksi edellä mainittu ResNet [51], DenseNet (*Densely Connected Convolutional Networks*) [58] ja MobileNet [59]. ResNet:ssä keskeisenä ominaisuutena ovat niin kutsutut jäännösyhteydet (*residual connection*), joiden ideana on mahdollistaa se, että tiettyjen kerrosten tulos ei välttämättä ohjaudu verkossa seuraavan olevaan kerrokseen syötteenä, vaan kerroksen tulos voidaan lisätä seuraavan kerroksen tulokseen. Näin neuroverkko sisältää ominaisuuden, jolla se voi ohittaa tai jättää huomiotta tarpeettomia kerroksia, minkä tarkoitus on ehkäistä katoavan gradientin ongelmaa syvissä verkoissa ja näin parantaa verkon oppimiskykyä. ResNet-verkot voivat olla erittäin monikerroksisia ilman että ne kärsivät monikerroksisuuden aiheuttamista tavallisista ongelmista, kuten katoavan gradientin ongelmasta, ja toisaalta jäännösyhteydet helpottavat ResNet-verkkojen koulutusta [51].

DenseNet-verkkojen keskeinen ominaisuus puolestaan on se, että verkon jokainen kerros on yhteydessä kaikkiin edellisiin kerroksiin. Näin ollen kukin verkon kerros saa syötteensä jokaiselta sitä edeltävältä verkon kerrokselta sen sijaan, että syöte tulisi vain edeltävältä kerrokselta. Tämä ominaisuus niin ikään auttaa estämään katoavan gradientin ongelmaa ja parantaa verkon tehokkuutta siten, että syvässä verkossa ei tarvitse käyttää niin montaa suodatinta eri kerroksissa, koska kukin kerros saa useampia syötteitä. Tällöin päästään pienempiä määrejä parametreja vaativiin, mutta silti hyvin toimiviin neuroverkkomalleihin [58].

5 Esimerkki datajoukon luokittelusta konvoluutioneuroverkon avulla

Tässä kappaleessa käsitellään esimerkkinä tiedon luokittelua konvoluutioneuroverkon avulla. Konvoluutioneuroverkkomalli rakennettiin luokittelemaan NFL Big Data Bowl 2025 datajoukon [60] sisältämää aineistoa. Kyseinen aineisto on amerikkalaisen jalkapallosarja NFL:n (*National Football League*) Kaggle-alustalle julkaisema datajoukko ja datajoukon analysointiin liittyvä kilpailu.

NFL on julkaissut kyseisen data-aineiston ja siihen liittyvän kilpailun vuosittain jo useamman vuoden ajan ja aineiston julkaisemisen sekä siihen liittyvän analysointikilpailun tarkoituksena on ollut saada luotua erilaisia metriikoita auttamaan sekä tuomaan uusia näkökulmia itse joukkueille että myös peliä seuraaville ja analysoiville tahoille, kuten medialle ja kannattajille. Data-aineiston julkaisussa ja kilpailussa pyritään keskittymään amerikkalaiseen jalkapalloon liittyviin suurin datamääriin, kuten esimerkiksi pelaajien liikkumiseen kentällä, pelitilanteisiin sekä peliin liittyviin tilastoihin. NFL Big Data Bowl datasetti sisältääkin muun muassa palaajien sijaintitietoja, nopeuksia, suuntia, pelitilanteita, yksittäisten pelien lopputulemia, otteluiden tuloksia, pelaajien tilastoja ja niin edelleen.

Kuluneiden vuosien aikana NFL Big Data Bowl kilpailussa on tehty analyyseja esimerkiksi liittyen yksittäisen pelaajan tai useiden pelaajien liikkeen ja reitin ennustamiseen, pelisuunnitelmien optimointiin ja strategiseen analyysiin, pelaajien suorituskyvyn arviointiin, puolustuksen ja hyökkäyksen vahvuuksien ja heikkouksien arviointiin, pelitilanteiden todennäköisyyksien arviointiin ja riskianalyyseihin sekä pelaajien loukkaantumisriskien ennustamiseen.

Pelaajien liikkeen ja reitin ennustamiseen liittyvien analyysien tarkoituksena on ollut saada kehitettyä malleja, joiden avulla pelaajien tulevan liikkeen suuntaa ja reittiä voidaan arvioida perustuen pelaajien aiempiin sijainteihin, nopeuksiin ja liikesuuntiin. Pelisuunnitelmien optimoinnin ja strategisen analyysin tarkoituksena on ollut analysoida, millaiset pelisuunnitelmat ja strategiat ovat tehokkaimpia eri tilanteissa ja miten esimerkiksi hyökkäyksen ja puolustuksen linjat reagoivat toisiinsa ja millaiset pelintilanteet ja muodostelmat johtavat suuriin pelinonnistumismahdollisuuksiin.

Pelaajien suorituskykyä arvioimalla on pyritty tutkimaan pelaajien yksilöllistä suorituskykyä pelissä kerätyn datan avulla ja saamaan tieto esimerkiksi siitä, kuinka nopeasti

keskushyökkääjä saavuttaa suuren nopeuden tietynlaisessa tilanteessa tai kuinka hyvin keskushyökkääjä pystyy rikkomaan puolustajien tekemiä taklauksia tietynlaisessa nopeudessa. Loukkaantumisriskien ennustamiseen liittyvissä analyyseissa on puolestaan pyritty ennustamaan pelaajien loukkaantumisriskiä liikkumismallien perusteella ja pyritty tunnistamaan pelitilanteita, joiden yhteydessä tai seurauksena pelaajien loukkaantumiset ovat yleisiä.

NFL Big Data Bowl datalla on edellä esitellyn mukaisesti tehty useampana vuonna erittäin suuri määrä mitä erilaisimpia analyyseja, joilla on pyritty arvioimaan pelin kehittymistä ja luomaan erilaisia mittareita, joiden avulla pelin kehittymistä voisi seurata paremmin. Lisäksi NFL Big Data Bowl on ollut monelle data-analyytikolle tilaisuus opiskella data-analytiikkaa ja testata ja oppia kehittämään uusia ennustemalleja. Vuoden 2025 aineiston ja kilpailun fokukseksi on määritelty erityisesti se, mitä tapahtuu ennen kuin peli käynnistyy ja miten tiedon avulla voidaan luoda analyyttisia arvioita joukkueen hyökkäyksen tai puolustuksen pelikutsusta ja pelistä perustuen ennen kunkin pelin aloitusta (*”ball snap”*) tapahtuviin tapahtumiin.

5.1 Tutkimuksen tarkoitus

Tässä työssä valittiin kaksi eri datan luokitteluun perustuvaa kysymystä, joihin analyysilla pyrittiin vastaamaan. Ensimmäiseksi analysoitiin sitä, miten hyvin sekä hyökkäyksen että puolustuksen pelaajien muodostelmasta ennen kunkin yksittäisen pelin alkua pystytään ennustamaan, onko kyseessä juoksu- vai heittopeli.

Amerikkalaisessa jalkapallossa on samanaikaisesti kentällä 11 hyökkäyksen pelaajaa sekä 11 puolustuksen pelaajaa, joten mallin koulutukseen käytettävä aineisto, ja siten mallin ennustekyky, perustui 22 eri pelaajan sijaintiin karteesisessa koordinaatistossa ennen pelin käynnistymistä. Sijainnin lisäksi analyysissa käytettiin hyväksi tietoja pelaajien hetkellisestä nopeudesta, kiihtyvyydestä ja orientaatiosta.

Laajuudestaan huolimatta aineisto on kohtuullisen monimutkainen ja massiivisuudestaan huolimatta tällaiseen käyttötarkoitukseen rajallinen ja tämän vuoksi edellä esitetyn lisäksi analysoitiin, että miten hyvin sekä hyökkäyksen että puolustuksen pelaajien muodostelmasta sekä liikkeestä välittömästi aloituksen jälkeen pystytään ennustamaan, onko kyseessä juoksu- vai heittopeli. Tämän analyysin tarkoituksena oli tuottaa tarkkuudeltaan ensimmäistä

analyysia parempia tuloksia, koska ensimmäisen analyysin tuottaman arviointitarkkuuden ei ennakoitu olevan kovinkaan suuri tämän tyyppisessä tehtävässä tämän kaltaisella aineistolla.

Analyysitarkkuuden voidaan olettaa jälkimmäisessä analyysissa olevan parempi sen vuoksi, että pelin jo käynnistyttyä pelissä tyypillisesti tapahtuvat tiettyjen pelaajaryhmien, kuten esimerkiksi hyökkäyksen linjapelaajien, liikkeet paljastavat usein hyvin sen, onko kyseessä juoksu- vai heittopeli. Ensimmäisen tutkimuskysymyksen tapauksessa ongelmaksi muodostuu se, että mallin pitäisi pystyä luokittelemaan peli käytännössä pelkästään pelaajien aloituskuviossa olevan sijainnin ja orientaation perusteella, koska pelaajat eivät pääsääntöisesti käytännössä ole liikkeessä ennen pelin käynnistymistä.

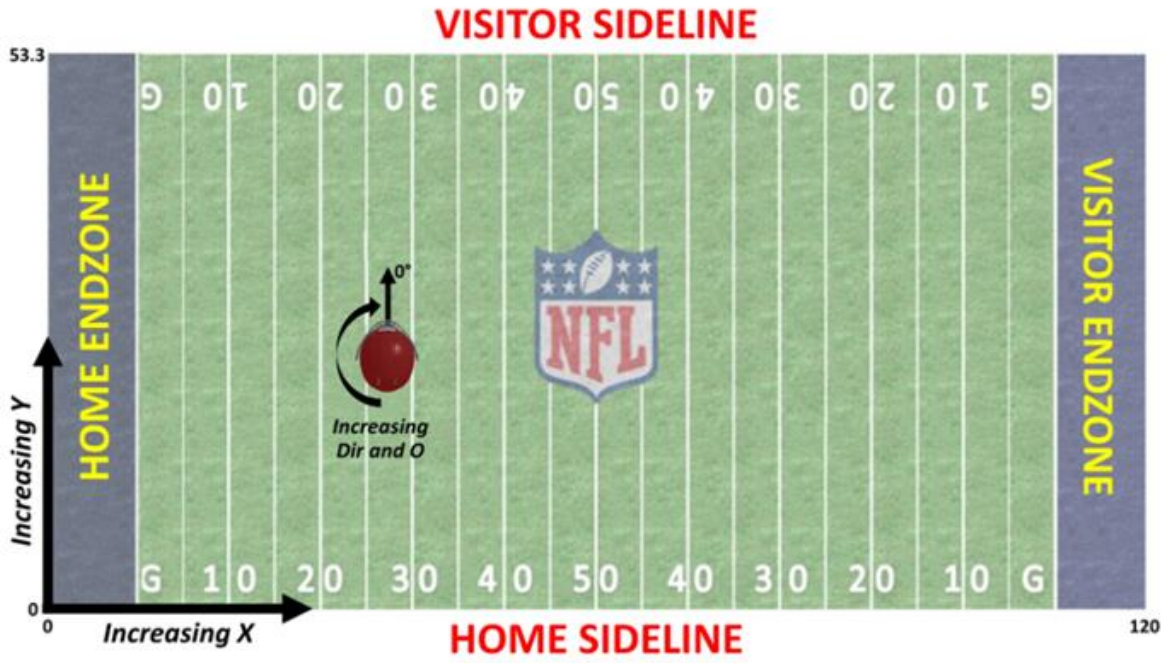
5.2 Tutkimuksessa käytetty alusta sekä tutkimusdata

Tutkimuksessa käytetty tutkimusaineisto on julkaistu ja saatavilla Kaggle-alustalta [60] ja koostuu kokonaisuudessaan 13 kappaleesta csv-muotoisia tiedostoja, jotka sisältävät yhteensä 8,17 GB numeerista dataa. Tiedostot sisältävät erilaista analytiikkadataa NFL-peleistä, pelien yksittäisistä pelitapahtumista sekä pelaajista useampien peliviikkojen ajalta.

Myös konvoluutioneuroverkko rakennettiin Kaggle-alustalle. Datan käsittelyssä ja neuroverkon ohjelmoinnissa käytettiin Python 3 ohjelmointikieltä ja Python 3 kielen datankäsittelyyn ja -kuvantamiseen, matriisioperaatioihin sekä koneoppimismenetelmiin ja tekoälyyn liittyviä yleisiä ohjelmointikirjastoja, kuten Pandas, Numpy ja Matplotlib. Tekoälykirjastona käytettiin Googlen Tensorflow-kirjastoa. Konvoluutioverkkojen ohjelmointiin käytettiin apuna aiemman NFL Big Data Bowl -kilpailun voittajaksi valitun työn Python-ohjelmaa nimeltään The Zoo [61].

Konvoluutioneuroverkon kouluttamista varten dataa muokattiin siten, että mallin koulutukseen ja validointiin käytettävä datajoukko sisälsi kunkin datasetin erillisen pelin osalta tiedot kunkin peliin osallistuvan 22 pelaajan sijainnista koordinaatistolla (x- ja y-koordinaatit), hetkellisestä nopeudesta, hetkellisestä kiihtyvyydestä, edellisestä mittapisteestä liikutun matkan pituudesta, pelaajan orientaatiosta sekä pelaajan liikkeen kulmasta.

Tarkempi esitys kunkin pelaajan seurantadatan koordinaateista on esitetty vielä kuvassa 12 [60]. Lisäksi pelaajien x-koordinaatin mukainen sijainti muutettiin analyysia varten siten, että suoran x-koordinaatin sijaan käytettiin sijaintia suhteessa pelin aloituslinjaan, jolloin x-koordinaatti vastasi pelaajan sijaintia suhteessa kunkin pelin aloituspaikkaan.



Kuva 12. Pelaajien koordinaatit ja seurantadatan ominaisuudet NFL Big Data Bowl 2025 datajoukon [60]. Data sisältää kutakin pelaajaa sekä palloa kohden tiedot pelaajan sijainnista x ja y koordinaatistossa, pelaajan nopeuden ja kiihtyvyyden sekä liikesuunnan ja liikekulman.

Pelaajien sijaintiin ja liikkeeseen liittyvien tietojen lisäksi datajoukko sisälsi tunnistetiedot ottelusta sekä yksittäisestä pelistä, johon kyseiset sijaintitiedot liittyvät sekä tiedon siitä, onko kyseinen peli ollut juoksu- vai heittopeli. Ensimmäisen analyysin tekoon käytettävä hetkellinen data otettiin 0,5 sekuntia ennen pelin käynnistymistä. Toisen analyysin tekoon käytettävä data puolestaan otettiin vastaavasti 0,5 sekuntia pelin käynnistymisen jälkeen.

Datan muokkauksen jälkeen data jaettiin koulutus- ja validointidataksi. Mallin koulutukseen datasta käytettiin 80 % ja loput 20 % datasta käytettiin mallin validointiin. Näin ollen koulutukseen käytettävä datajoukko sisälsi kaikkiaan 12 849 sattumanvaraisesti valittua erillistä peliä ja validointiin käytettävä datajoukko sisälsi 3 213 niin ikään sattumanvaraisesti valittua erillistä peliä.

Konvoluutioneuroverkkomalleja rakennettiin kaksilla eri malliparametreilla, toinen 1-ulotteisia konvoluutiokerroksia käyttäen, jolloin mallin kouluttamiseen käytetty koulutusdata syötettiin mallille 154 muuttujaa sisältävässä vektorimuodossa ja toinen 2-ulotteisia konvoluutiokerroksia käyttäen, jolloin mallin kouluttamiseen käytetty koulutusdata syötettiin mallille 22 kertaa 7 muuttujaa sisältävässä matriisimuodossa. Malleja testattiin käyttäen taulukoiden 1 ja 2 mukaisia konvoluutiokerrosten ja yhdistämiskerrosten yhdistelmien määriä.

Taulukko 1. 1-ulotteisia konvoluutiokerroksia käyttävän konvoluutioneuroverkkomallin testatut kerrosmäärät.

Kerros
1 konvoluutiokerros
1 konvoluutionkerros ja 1 yhdistämiskerros
2 konvoluutionkerrosta ja niiden välissä 1 yhdistämiskerros
2 konvoluutionkerrosta ja 2 yhdistämiskerrosta
3 konvoluutionkerrosta ja niiden välissä 2 yhdistämiskerrosta
3 konvoluutionkerrosta ja 3 yhdistämiskerrosta

Taulukko 2. 2-ulotteisia konvoluutiokerroksia käyttävän konvoluutioneuroverkkomallin testatut kerrosmäärät.

Kerros
1 konvoluutiokerros
1 konvoluutionkerros ja 1 yhdistämiskerros
2 konvoluutionkerrosta ja niiden välissä 1 yhdistämiskerros

Konvoluutioneuroverkkomallien aktivaatiofunktioksi konvoluutiokerroksiin valittiin ReLU-funktio. Konvoluutio- ja yhdistämiskerrosten lisäksi konvoluutioneuroverkoissa käytettiin lopussa datan luokittelua varten kahta peräkkäistä täysin yhdistettyä kerrosta, joista ensimmäisen aktivaatiofunktiona käytettiin ReLU-funktiota ja jälkimmäisen aktivaatiofunktiona Softmax-funktiota. Mallin virhefunktiona käytettiin ristientropia (*sparse categorical cross-entropy*) virhefunktiota.

Konvoluutioneuroverkon tuottaman analyysitarkkuuden havainnollistamiseksi ja konvoluutioneuroverkkotekniikan harjoitusdataan sopivuuden testaamiseksi datan analysointia varten rakennettiin myös syväoppiva monikerrosperseptroni. Monikerrosperseptroni muodostettiin niin ikään testaamalla sopivaa määrää piilokerroksia taulukon 3 mukaisesti, käyttäen piilokerrosten aktivaatiofunktiona ReLU-funktiota ja viimeisen neuronikerroksen aktivaatiofunktiona Softmax-funktiota. Virhefunktiona käytettiin niin ikään ristientropia (*sparse categorical cross-entropy*) virhefunktiota.

Taulukko 3. Monikerrosperseptronin testatut kerrosmäärät.

Kerros
2 täysin yhdistettyä kerrosta
3 täysin yhdistettyä kerrosta
4 täysin yhdistettyä kerrosta
5 täysin yhdistettyä kerrosta

Edellä esitettyjen syväoppimismenetelmien lisäksi data analysoitiin käyttäen logistista regressiomenetelmää sekä päätöspuihin perustuvaa logistista XGBoost-menetelmää. Logistisen regression ja XGBoost-menetelmien avulla pyrittiin vertaamaan käytetyillä syväoppimismenetelmillä saatuja analyysitarkkuuksia muilla menetelmillä saatuihin analyysitarkkuuksiin ja siten selvittämään, että kuinka hyvin käytetyt syväoppimismenetelmät sopivat tämän kyseisen datan analysointiin.

Linkit eri menetelmiin liittyviin Python-koodeihin on esitetty liitteessä 1.

5.3 Tulokset, niiden analysointi ja johtopäätökset

Ensimmäistä tutkimuskysymystä varten rakennetun eli 0,5 sekuntia ennen pelin käynnistymistä kerätyn datan avulla koulutetun 1-ulotteisen konvoluutionneuroverkon analyysitarkkuudeksi (accuracy) parhaiten toimivalla yhdistelmällä konvoluutio- ja yhdistämiskerroksia saatiin 0,6772 ja vastaavasti toista tutkimuskysymystä varten rakennetun eli 0,5 sekuntia pelin käynnistymisen jälkeen kerätyn datan avulla koulutetun 1-ulotteisen konvoluutioneuroverkon analyysitarkkuudeksi saatiin 0,7504. Datasta määritetyt F1-testiluvut (*F1-score*) malleille olivat ennen pelin käynnistymistä kerätystä datasta mitattuna juoksupelille 0,5293 ja heittopelille 0,7544 ja pelin käynnistymisen jälkeen kerätystä datasta mitattuna juoksupelille 0,6573 ja heittopelille 0,8037. Vastaavien konvoluutioneuroverkkojen ennusteiden ja testidatan avulla muodostetut hämmennysmatriisit (*confusion matrix*) olivat:

$$\begin{bmatrix} 583 & 671 \\ 366 & 1593 \end{bmatrix} \quad ja \quad \begin{bmatrix} 769 & 485 \\ 317 & 1642 \end{bmatrix}.$$

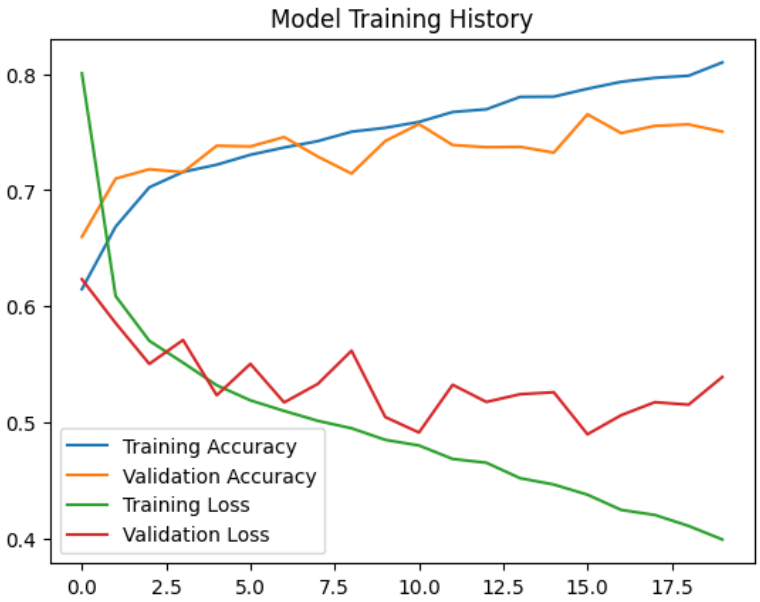
Hämmennysmatriisien tulokset siis tarkoittavat sitä, että pelin käynnistymisen jälkeen kerätyn datan tapauksessa koulutettu malli ennusti 769 kertaa oikein juoksupelin olevan juoksupeli ja 1642 kertaa oikein heittopelin olevan heittopeli. Vastaavasti malli ennusti 485 kertaa heittopelin olevan juoksupeli ja 317 juoksupelin olevan heittopeli. Mallia testattiin taulukon 1

mukaisilla vaihtoehdoilla piilokerroksia ja rakenteita ja parhaiten ennustava malli saatiin muodostettua taulukon 4 mukaisella rakenteella.

Taulukko 4. Harjoituksessa koulutetun 1-ulotteisen konvoluutioneuroverkon rakenne.

Kerros	Tulosteen muoto	Parametrien määrä
1D Konvoluutiokerros	(0, 153, 128)	384
Yhdistämiskerros (max)	(0, 76, 128)	0
1D Konvoluutiokerros	(0, 75, 64)	16 448
Yhdistämiskerros (max)	(0, 37, 64)	0
1D Konvoluutiokerros	(0, 36, 64)	8 256
Normalisointikerros	(0, 2 304)	0
Täysin yhdistetty kerros	(0, 32)	73 760
Täysin yhdistetty kerros	(0, 4)	132

Tällä rakenteella muodostettuja 1-ulotteisia konvoluutioneuroverkkoja koulutettiin 20 kierrosta, minkä havaittiin olevan suurin piirtein optimaalinen määrä koulutusta. Tähän asti mallin kouluttaminen lisäsi mallin tarkkuutta, mutta suurempien kierrosmäärien käyttäminen alkoi selkeästi aiheuttaa sen, että mallissa esiintyi selkeää ylisovittamista, jolloin mallin tekemä virhe testausaineiston kohdalla alkoi selkeästi kasvamaan. Kuvassa 13 on esitetty koulutettujen konvoluutioneuroverkkomallien opetushistoria.



Kuva 13. Pelin käynnistymisen jälkeen kerätyn datan avulla koulutetun 1-ulotteisen konvoluutioneuroverkkomallin koulutushistoria.

2-ulotteisia konvoluutiokerroksia käyttävällä konvoluutioneuroverkkomallilla analyysitarkkuuksiksi saatiin 0,5945 ennen pelin käynnistymistä kerätyn datan avulla ja

0,7242 pelin käynnistymisen jälkeen kerätyn datan avulla. Datasta määritetyt F1-testiluvut malleille olivat ennen pelin käynnistymistä kerätystä datasta mitattuna juoksupelille 0,5673 ja heittopelille 0,6184 ja pelin käynnistymisen jälkeen kerätystä datasta mitattuna juoksupelille 0,5943 ja heittopelille 0,7911. Vastaavat hämmennysmatriisit olivat:

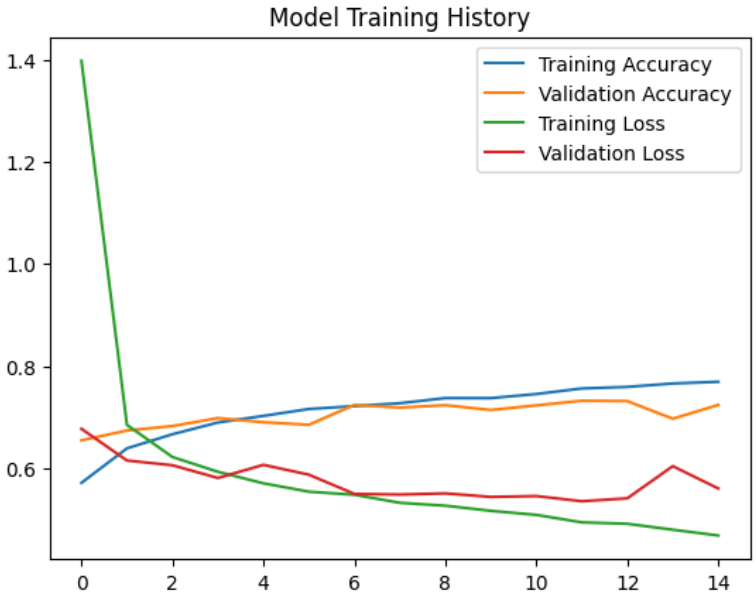
$$\begin{bmatrix} 854 & 400 \\ 903 & 1056 \end{bmatrix} \quad ja \quad \begin{bmatrix} 649 & 605 \\ 281 & 1678 \end{bmatrix}.$$

Pelin käynnistymisen jälkeen kerätyn datan tapauksessa koulutettu malli siis ennusti 649 kertaa oikein juoksupelin olevan juoksupeli ja 1678 kertaa oikein heittopelin olevan heittopeli. Vastaavasti malli ennusti 605 kertaa heittopelin olevan juoksupeli ja 281 juoksupelin olevan heittopeli. Mallia testattiin taulukon 2 mukaisilla vaihtoehdoilla piilokerroksia ja rakenteita ja parhaiten ennustava malli saatiin muodostettua taulukon 5 mukaisella 1 konvoluutiokerroksen rakenteella.

Taulukko 5. Harjoituksessa koulutetun 2-ulotteisen konvoluutioneuroverkon rakenne.

Kerros	**Tulosteen muoto**	**Parametrien määrä**
2D Konvoluutiokerros	(0, 21, 6, 22)	110
Normalisointikerros	(0, 2 772)	0
Täysin yhdistetty kerros	(0, 32)	88 736
Täysin yhdistetty kerros	(0, 4)	132

2-ulotteisia konvoluutiokerroksia sisältävää konvoluutioneuroverkkomallia koulutettiin 15 kierrosta, minkä havaittiin tässä tapauksessa olevan suurin piirtein optimaalinen määrä koulutusta. Tähän asti mallin kouluttaminen lisäsi mallin tarkkuutta, mutta suurempien kierrosmäärien käyttäminen alkoi selkeästi aiheuttaa ylisovittamista, jolloin mallin tekemä virhe testausaineiston kohdalla alkoi selkeästi kasvamaan. Kuvassa 14 on esitetty 2-ulotteisen pelin käynnistymisen jälkeen kerätyn datan avulla koulutetun konvoluutioneuroverkkomallin opetushistoria.



Kuva 14. Pelin käynnistymisen jälkeen kerätyn datan avulla koulutetun 2-ulotteisen konvoluutioneuroverkkomallin koulutushistoria.

Monikerrosperseptronilla vastaaviksi mallin tarkkuuksiksi saatiin 0,6091 ennen pelin käynnistymistä kerätyn datan avulla ja 0,7345 pelin käynnistymisen jälkeen kerätyn datan avulla. Datasta määritetyt F1-testiluvut malleille olivat ennen pelin käynnistymistä kerätystä datasta mitattuna heittopelille 0,7571 ja pelin käynnistymisen jälkeen kerätystä datasta mitattuna juoksupelille 0,6619 ja heittopelille 0,7815. Juoksupelin F1-testilukua ennen pelin alkua kerätystä datasta ei pysty määrittämään. Vastaavien monikerrosperseptronien ennusteiden ja testidatan avulla muodostetut hämmennysmatriisit puolestaan olivat:

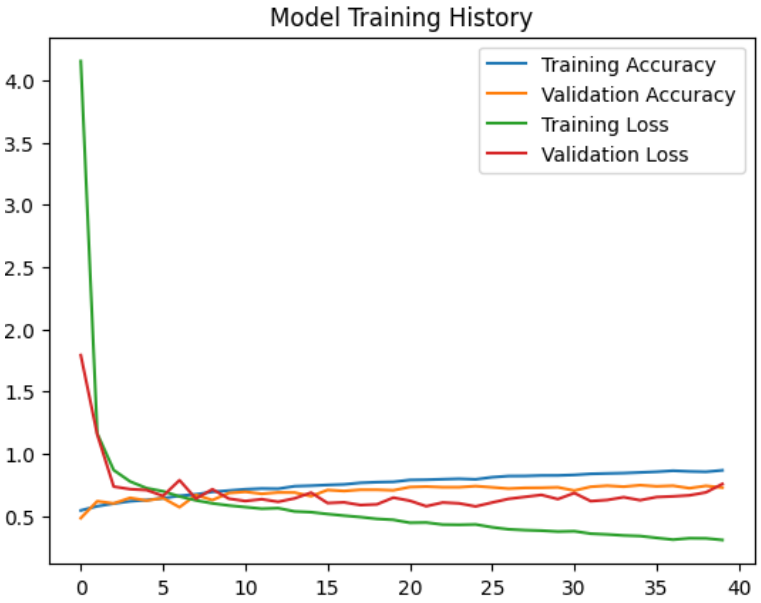
$$\begin{bmatrix} 0 & 1254 \\ 2 & 1957 \end{bmatrix} \quad ja \quad \begin{bmatrix} 835 & 419 \\ 434 & 1525 \end{bmatrix}.$$

Pelin käynnistymisen jälkeen kerätyn datan tapauksessa koulutettu malli siis ennusti 835 kertaa oikein juoksupelin olevan juoksupeli ja 1525 kertaa oikein heittopelin olevan heittopeli. Vastaavasti malli ennusti 419 kertaa heittopelin olevan juoksupeli ja 434 juoksupelin olevan heittopeli. Monikerrosperseptronia testattiin taulukon 3 mukaisilla vaihtoehdoilla piilokerroksia ja rakenteita ja parhaiten ennustava malli saatiin muodostettua taulukon 6 mukaisella rakenteella.

Taulukko 6. Harjoituksessa koulutetun monikerrosperseptronin rakenne.

Kerros	Tulosteen muoto	Parametrien määrä
Täysin yhdistetty kerros	(0, 128)	19 840
Täysin yhdistetty kerros	(0, 64)	8 256
Täysin yhdistetty kerros	(0, 10)	650

Monikerrosperseptronia koulutettiin 40 kierrosta, minkä havaittiin tässä tapauksessa olevan suurin piirtein optimaalinen määrä koulutusta. Tähän asti mallin kouluttaminen lisäsi mallin tarkkuutta, mutta suurempien kierrosmäärien käyttäminen alkoi selkeästi aiheuttaa ylisovittamista. Kuvassa 15 on esitetty pelin käynnistymisen jälkeen kerätyn datan avulla koulutetun monikerrosperseptronin opetushistoria.



Kuva 15. Pelin käynnistymisen jälkeen kerätyn datan avulla koulutetun monikerrosperseptronin koulutushistoria.

Logistisella regressiolla luodun mallin tarkkuudeksi saatiin ennen pelin käynnistymistä otetusta datasta 0,6134 ja pelin käynnistymisen jälkeen otetusta datasta 0,6293. Datasta määritetyt F1-testiluvut malleille olivat ennen pelin käynnistymistä kerätystä datasta mitattuna juoksupelille 0,1753 ja heittopelille 0,7476 ja pelin käynnistymisen jälkeen kerätystä datasta mitattuna juoksupelille 0,3495 ja heittopelille 0,7408. Vastaavat hämmennysmatriisit olivat:

$$\begin{bmatrix} 132 & 1122 \\ 120 & 1839 \end{bmatrix} \quad ja \quad \begin{bmatrix} 320 & 934 \\ 257 & 1702 \end{bmatrix}.$$

XGBoost-menetelmällä luodun mallin tarkkuudeksi puolestaan saatiin 0,6990 ennen pelin käynnistymistä otetusta datasta ja 0,8335 pelin käynnistymisen jälkeen otetusta datasta. Datasta määritetyt F1-testiluvut malleille olivat ennen pelin käynnistymistä kerätystä datasta mitattuna juoksupelille 0,5566 ja heittopelille 0,7722 ja pelin käynnistymisen jälkeen kerätystä datasta mitattuna juoksupelille 0,7913 ja heittopelille 0,8615. Vastaavat hämmennysmatriisit olivat:

$$\begin{bmatrix} 607 & 647 \\ 320 & 1639 \end{bmatrix} \quad ja \quad \begin{bmatrix} 1014 & 240 \\ 295 & 1664 \end{bmatrix}.$$

XGBoost-menetelmällä päästiin näin ollen edellä esitellyistä menetelmistä parhaimpaan analyysitarkkuuteen. Logistisella regressiolla analyysitarkkuus puolestaan jäi kohtuullisen vaatimattomaksi ja niin logistinen regressio kuin osin myös monikerrosperseptroni yliarvioivat heittopelien määrää suhteessa juoksupeleihin, erityisesti silloin kun ne koulutettiin ennen pelin alkua kerätyllä datalla.

6 Johtopäätökset

Tässä tutkielmassa tarkasteltiin tekoälyn, koneoppimisen ja syväoppimisen perusteita sekä syväoppivia neuroverkkoja, erityisesti konvoluutioneuroverkkoja, niiden rakennetta ja ominaisuuksia. Lisäksi tutkielmassa demonstroitiin konvoluutioneuroverkkomallin soveltuvuutta valitun NFL Big Data Bowl datajoukon sisältämän datan kuvaamiseen käsittelemällä datajoukko neuroverkkomallin kouluttamiseen sopivaksi syötteeksi ja kouluttamalla datan avulla kaksi eri arkkitehtuurin konvoluutioneuroverkkomallia. Lisäksi dataa käytettiin kouluttamaan monekerrosperseptroni, logistinen regressiomalli sekä päätöspuupohjainen XGBoost-malli. Näiden mallien rakentamisen tarkoituksena oli verrata konvoluutioneuroverkkomallin soveltuvuutta tämän kaltaisesti muokatun datan käsittelyyn tässä sovelluksessa.

Tekoäly, koneoppiminen ja syväoppiminen liittyvät konseptuaalisesti toisiinsa. Tekoäly on käsitteistä laajin ja kattaa tekoälyyn liittyvät matematiikan ja teknologian käsittelyn lisäksi paljon muitakin osa-alueita, kuten esimerkiksi tekoälyyn liittyvät eettiset ja yhteiskunnalliset kysymykset. Koneoppiminen on tekoälyn osa-alue, jossa pyritään kehittämään tilastollisia menetelmiä, joiden avulla tietokoneohjelma voidaan opettaa suorittamaan tehtäviä perustuen ohjelman saamaan aiempaan dataan. Syväoppiminen puolestaan on koneoppimisen osa-alue, josta puhuttaessa tarkoitetaan joukkoa sellaisia koneoppimismenetelmiä, joissa hyödynnetään monikerroksisia neuroverkkoja.

Syväoppivia neuroverkkoja on kehitetty monenlaisilla arkkitehtuureilla ja mitä moninaisimpiin sovelluksiin, kuten esimerkiksi tietokonenäkösovelluksiin tai syväoppiviin kielimalleihin. Konvoluutioneuroverkkojen erinomaisuus syväoppimissovelluksissa verrattuna muihin syväoppiviin neuroverkkoihin perustuu niiden kykyyn oppia opetusdatassa olevia rakenteellisia kuvapiirteitä, kuten muotoja, reunoja ja erilaisia kuvioita. Tunnistettujen kuvapiirteiden avulla voidaan edelleen tunnistaa abstrakteja asioita, kuten vaikkapa eläimiä. Esimerkiksi eteenpäin kytketyn täysin yhdistetyistä kerroksista koostuvan neuroverkon opettaminen tunnistamaan tällaisia on vaikeampaa ja enemmän aikaa ja tehoa vievää sekä huomattavan paljon enemmän opetusdataa vaativaa.

Tutkielmassa esitetty NFL Big Data Bowl data-aineistoon perustuva analyysi oli idealtaan hyvin yksinkertainen ja kyseisellä aineistolla on tehty ja jaettu valtava määrä erilaisia, huomattavasti monimutkaisempia ja edistyneempiä analyysejä. Tässä tutkielmassa tehdyn

analyysin tarkoitus oli kuitenkin lähinnä demonstroida syväoppivan neuroverkon, tässä tapauksessa konvoluutioneuroverkon rakentamista ja käyttöä ja osoittaa, minkälaisia tuloksia konvoluutioneuroverkon avulla voidaan tämän kaltaisesta tiedosta analysoimalla saada. Huomattavasti tätä analyysiä monimutkaisempia analyyseja käytetäänkin tällä hetkellä jo hyväksi monenlaisessa urheiluanalytiikassa ja lienee oletettavaa, että tekoälyn käyttö urheilulajien analysoinnin yhteydessä tulee sekä syvenemään, että myös sen hyödyntäminen useammissa lajeissa tullee yleistymään.

Tutkielman esimerkki oli sikäli toimiva, että molemmilla demonstroiduilla konvoluutioneuroverkoilla päästiin kohtuulliseen tarkkuuteen pelityypin ennustamisessa. Tämä kuitenkin pätee lähinnä pelin aloituksen jälkeen kerätyn datan avulla koulutetuille neuroverkkomalleille, ennen pelin alkua kerätyn datan avulla koulutettujen mallien tuottama ennustekyky jäi selkeästi matalammaksi. Tämä toisaalta on oletettuakin, sillä ennen pelin alkua kerätyssä datassa pelaajat ovat vielä aloitusmuodostelmassa pääosin paikallaan, jolloin mitkään pelaajien liikkeeseen liittyvät parametrit eivät ole mallin käytettävissä. Lisäksi aloitusmuodon yhteydessä sekä puolustavan että hyökkäävän joukkueen on tarkoituskin pyrkiä peittämään sitä, minkälainen pelisuunnitelma on käytössä, joten pelkästä aloitusmuodostelmasta pelin lopputulokset ennustaminen ei varmastikaan kovinkaan tarkkaa vastausta anna.

Mallien tarkkuuksista voidaan toisaalta myös päätellä, että tässä harjoituksessa rakennetut 1- ja 2-ulotteiset konvoluutioneuroverkkomallit eivät ole täysin optimoituja oppiakseen datan ominaisuuksia riittävän hyvin, koska XGBoost-menetelmällä päästiin jonkin verran korkeampaan analyysitarkkuuteen kuin mihin harjoituksessa rakennettujen konvoluutioneuroverkkomallien avulla päästiin. Tämä voi lisäksi tarkoittaa, että konvoluutioneuroverkko ei välttämättä ole optimaalisin mahdollinen koneoppimismalli tämän kaltaisen datan analysointiin tällaisella datankäsittelyllä sovellettuna ylipäätään, koska tässä tutkielmassa muokattu data ei välttämättä sisältänyt riittävän selkeitä rakenteellisia kuvapiirteitä, joita konvoluutioneuroverkkomallin avulla pystyttäisiin hyödyntämään.

Tämän kaltaisen analyysin tekeminen konvoluutioneuroverkkojen avulla olisi varmasti selkeästi kattavampaa ja loisi mallin, jonka ennustekyky olisi selkeästi parempi, jos konvoluutioneuroverkkomalli koulutettaisi pelistä otettujen hetkellisten kuvien sijaan videoilla tai useiden peräkkäisten kuvien sarjoilla. Tällöin konvoluutioneuroverkon kouluttamisessa saataisiin hyödynnettyä kaikki mahdollinen pelitilanteiden sisältämä tieto

mukaan lukien tarkat tiedot siitä, miten pelaajien sijainnit, nopeudet ja suunnat ovat kehittyneet pelin edessä, ei vain hetkellisessä tilanteessa.

Lähteet

[1] Euroopan parlamentti, [Online]. Available: https://www.europarl.europa.eu/pdfs/news/expert/2020/9/story/20200827STO85804/20200827STO85804_fi.pdf. [Haettu 28 11 2024].

[2] Kansainvälinen standardointijärjestö, ”ISO/IEC 22989:2022”.

[3] International Business Machines Corporation, [Online]. Available: https://www.ibm.com/topics/artificial-intelligence. [Haettu 28 11 2024].

[4] Amazon, [Online]. Available: https://aws.amazon.com/what-is/artificial-intelligence/. [Haettu 28 11 2024].

[5] P. Bory, S. Natale ja C. Katzenbach, ”Strong and Weak AI Narratives: An Analytical Framework,” *AI & Society,* 2024. htpps:://doi.org/10.1007/s00146-024-02087-8.

[6] E. Alpaydin, Introduction to Machine Learning, Fourth Edition toim., The MIT Press, 2020, p. 712. ISBN: 9780262043793.

[7] Google LLC, [Online]. Available: https://cloud.google.com/discover/what-is-supervised-learning. [Haettu 2 12 2024].

[8] Google LLC, [Online]. Available: https://cloud.google.com/discover/deep-learning-vs-machine-learning#artificial-intelligence-vs-machine-learning-vs-deep-learning. [Haettu 3 12 2024].

[9] Google LLC, [Online]. Available: https://cloud.google.com/discover/what-is-unsupervised-learning. [Haettu 5 12 2024].

[10] International Business Machines Corporation, [Online]. Available: https://www.ibm.com/topics/semi-supervised-learning. [Haettu 7 12 2024].

[11] Amazon, [Online]. Available: https://aws.amazon.com/what-is/reinforcement-learning/. [Haettu 5 12 2024].

[12] International Business Machines Corporation, [Online]. Available: https://www.ibm.com/topics/neural-networks. [Haettu 18 12 2024].

[13] W. McCulloch ja W. Pitts, ”A Logical Calculus of Ideas Immanent in Nervous Activity,” *Bulletin of Mathematical Biophysics,* osa/vuosik. 5, pp. 115-133, 1943. https://doi.org/10.1007/bf02478259.

[14] F. Rosenblatt, ”The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain,” *Psychological Review,* osa/vuosik. 65, nro 6, pp. 386-408, 1958. https://doi.org/10.1037/h0042519.

[15] L. Hardesty, ”Explained: Neural networks,” MIT News Office, 2017.

[16] P. Werbos, *Beyond Regression: New tools for prediction and analysis in the behavioral science,* Harvard University, 1974.

[17] Y. LeCun, ”Backpropagation Applied to Handwritten Zip Code Recognition,” *Neural Computing,* osa/vuosik. 1, nro 4, pp. 541-551, 1989. https://doi.org/10.1162/neco.1989.1.4.541.

[18] P. Madan ja S. Madhavan, ”An Introduction to deep learning,” [Online]. Available: https://developer.ibm.com/learningpaths/get-started-with-deep-learning/an-introduction-to-deep-learning/. [Haettu 12 12 2024].

[19] P. Ramachandran, B. Zoph ja Q. Le, ”Searching for Activation Functions,” 2017. arXiv:1710.05941.

[20] K. Fukushima, ”Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements,” *IEEE Transactions on Systems Science and Cybernetics,* osa/vuosik. 5, nro 4, pp. 322-333, 1969. htpps://doi.org/10.1109/tssc.1969.300225.

[21] J. Schmidhuber, ”Deep learning in neural networks: An overview,” *Neural Networks,* osa/vuosik. 61, pp. 85-117, 2015. https://doi.org/10.1016/j.neunet.2014.09.003.

[22] S. Sonoda ja N. Murata, ”Neural network with unbounded activation functions is universal approximator,” *Applied and Computational Harmonic Analysis,* osa/vuosik. 43, nro 2, pp. 233-268, 2017. https://doi.org/10.1016/j.acha.2015.12.005.

[23] A. Maas, H. Awni ja A. Ng, ”Rectifier Nonlinearities Improve Neural Network Acoustic Models,” tekijä: *International Conference on Machine Learning*, 2013.

[24] K. He, X. Xhang, S. Ren ja J. Sun, ”Delving Deep into Rectifiers: Surpassing Human-Level Performance on Image Net Classification,” tekijä: *Proceedings of the IEEE international conference on computer vision*, 2015. https://doi.org/10.1109/iccv.2015.123.

[25] D.-A. Clevert, T. Unterthiner ja S. Hochreiter, ”Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs),” *ICLR,* 2015.

[26] G. Klambauer, T. Unterthiner, A. Mayr ja S. Hochreiter, ”Self-Normalizing Neural Networks,” tekijä: *NeurIPS*, 2017.

[27] D. Hendrycks ja K. Gimpel, ”Gaussian Error Linear Units (GELUs),” 2016.

[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser ja I. Polosukhin, ”Attention is All you Need,” *Advances in Neural Information Processing Systems,* osa/vuosik. 30, 2017.

[29] C. M. Bishop, Pattern Recognition and Machine Learning, osa/vuosik. 4, Springer, 2006, pp. 75. ISBN: 0-387-31073-8.

[30] OpenAI, [Online]. Available: https://openai.com/index/techniques-for-training-large-neural-networks/. [Haettu 13 1 2025].

[31] Google Developer, [Online]. Available: https://developers.google.com/machine-learning/crash-course/neural-networks/backpropagation. [Haettu 20 3 2025].

[32] V. Bushaev, ”TDS Archive,” 2017. [Online]. Available: https://towardsdatascience.com/how-do-we-train-neural-networks-edd985562b73. [Haettu 7 1 2025].

[33] D. S. Wizards. [Online]. Available: https://medium.com/@datasciencewizards/a-simple-guide-to-gradient-descent-algorithm-60cbb66a0df9. [Haettu 17 1 2025].

[34] GeegsforGeegs. [Online]. Available: https://www.geeksforgeeks.org/backpropagation-in-neural-network/. [Haettu 17 1 2025].

[35] G. Dreyfus, Neural Networks: Methodology and Applications, osa/vuosik. third edition, Springer, 2005. ISBN: 978-3642061875.

[36] L. Bottou, Online Algorithms and Stochastic Approximations, Cambridge University Press, 1998.

[37] D. P. Kingma ja J. L. Ba, ”Adam: A Method for Stochastic Optimization,” *ICLR 2015,* p. 15, 2015.

[38] T. Mucci. [Online]. Available: https://www.ibm.com/think/topics/overfitting-vs-underfitting. [Haettu 20 3 2025].

[39] J. Schmidhuber, ”Annotated History of Modern AI and Deep Learning. Technical Report IDSIA-22-22,” 2022.

[40] International Business Machines Corporation, [Online]. Available: https://www.ibm.com/topics/deep-learning. [Haettu 18 12 2024].

[41] D. Xu, Y. Wang, S. Xu, K. Zhu, N. Zhang ja X. Zhang, ”Infrared and Visible Image Fusion with a Generative Adversarial Network and a Residual Network,” *Applied Sciences,* osa/vuosik. 10, 2020. https://doi.org/10.3390/app10020554.

[42] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville ja Y. Bengio, ”Generative Adversarial Nets,” tekijä: *NeurIPS Proceedings*, 2014.

[43] Coursera, [Online]. Available: https://www.coursera.org/articles/neural-network-architecture. [Haettu 16 12 2024].

[44] OpenAI, [Online]. Available: https://openai.com/index/chatgpt/. [Haettu 12 2 2025].

[45] Google LLC, [Online]. Available: https://developers.google.com/machine-learning/glossary/#convolutional_neural_network. [Haettu 15 12 2024].

[46] International Business Machines Corporation, [Online]. Available: https://www.ibm.com/topics/convolutional-neural-networks. [Haettu 14 1 2025].

[47] L. G. Wright, T. Onodera, M. M. Stein, T. Wang, D. Schachter, Z. Hu ja P. L. McMahon, ”Deep Physical Neural Networks Trained with Backpropagation,” *Nature,* osa/vuosik. 601, nro 7894, pp. 549-555, 2022. https://doi.org/10.1038/s41586-021-04223-6.

[48] V. H. Phung ja E. J. Rhee, ”A Deep Learning Approach for Classification of Cloud Image Patches on Small Datasets,” *Journal of information and communication convergence engineering,* osa/vuosik. 16, pp. 173-178, 2018.

[49] I. Goodfellow, Y. Bengio ja A. Courville, Deep Learning, Cambridge: MIT Press, 2017. ISBN: 9780262035613.

[50] A. Krizhevsky, I. Sutskever ja G. E. Hinton, ”ImageNet classification with deep convolutional neural networks,” *Communications of the ACM,* osa/vuosik. 60, nro 6, pp. 84-90, 2017. https://doi.org/10.1145/3065386.

[51] K. He, X. Zhang, S. Ren ja J. Sun, ”Deep residual learning for image recognition,” tekijä: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016. https://doi.org/10.1109/cvpr.2016.90.

[52] R. Collobert ja J. Weston, ”A unified architecture for natural language processing,” tekijä: *Proceedings of the 25th international conference on Machine learning*, New York, 2008. https://doi.org/10.1145/1390156.1390177.

[53] K. Chellapilla ja D. B. Fogel, ”Evolving neural networks to play checkers without relying on expert knowledge,” *IEEE transactions on neural networks,* osa/vuosik. 10, nro 6, pp. 1382-1391, 1999. https://doi.org/10.1109/72.809083.

[54] C. Clark ja A. Storkey, ”Teaching Deep Convolutional Neural Networks to Play Go,” 2014.

[55] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj ja A. Iosifidis, ”Forecasting Stock Prices from the Limit Order Book Using Convolutional Neural Networks,” tekijä: *2017 IEEE 19th conference on business informatics (CBI)*, Thessaloniki, 2017.

[56] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever ja R. Salakhutdinov, ”A Simple Way to Prevent Neural Networks from Overfitting,” *Journal of Machine Learning Research,* osa/vuosik. 15, nro 56, pp. 1929-1958, 2014.

[57] K. Simonyan ja A. Zisserman, ”Very Deep Convolutional Networks for Large-Scale Image Recognition,” 2014. https://doi.org/10.48550/arXiv.1409.1556.

[58] G. Huang, Z. Liu, L. Van Der Maaten ja K. Q. Weinberger, ”Densely Connected Convolutional Networks,” *Proceedings of the IEEE conference on computer vision and pattern recognition,* pp. 4700-4708, 2017.

[59] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto ja A. Hartwig, ”MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” *arXiv preprint arXiv:1704.04861,* 2017.

[60] M. Lopez, B. Thompson, A. Blake, P. Mooney ja H. Addison, ”NFL Big Data Bowl 2025,” Kaggle, 2024.

[61] D. Gordeev ja P. Singer, ”The Zoo,” Kaggle.

Liitteet

Liite 1. Linkit tutkimuksessa käytettyihin Python 3-koodeihin

Tutkielmassa rakennettujen konvoluutioneuroverkkojen, monikerrosperseptronin sekä logistisen regressiomallin ja XGBoost-mallin Python 3 -koodit löytyvät seuraavista osoitteista:

1-ulotteinen konvoluutioneuroverkko:

https://www.kaggle.com/code/teemusarikka/cnn-gradu-aftersnap/notebook

2-ulotteinen konvoluutioneuroverkko:

https://www.kaggle.com/code/teemusarikka/cnn2d-gradu-aftersnap/notebook

Monikerrosperseptroni:

https://www.kaggle.com/code/teemusarikka/mult-gradu-aftersnap/notebook

Logistinen regressiomalli:

https://www.kaggle.com/code/teemusarikka/logreg-gradu-aftersnap/notebook

XGBoost:

https://www.kaggle.com/code/teemusarikka/xgboost-gradu-aftersnap/notebook